

U.S. DEPARTMENT OF THE INTERIOR

U.S. GEOLOGICAL SURVEY

**DATA REPORT AND DESCRIPTION OF TECHNIQUES
FOR THE 1993 SEASON OF THE SAN FRANCISCO BAY AREA
REGIONAL BROAD-BAND TRANSECT, CALIFORNIA
(PASSCAL EXPERIMENT P9220)**

by

John R. Evans¹, John M. Coakley¹,
and the Broadband Transect Working Group
(H.M. Iyer¹, Stan Ruppert², Susan Schwartz³, John Vidale^{1,3}, and George Zandt²)

Open-File Report 94-433

This report is preliminary and has not been reviewed for conformity with U.S. Geological Survey editorial standards or the North American Stratigraphic Code. Any use of trade, product, or firm 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.

¹U.S. Geological Survey, Menlo Park, California

²Lawrence Livermore National Laboratory, Livermore, California

³University of California, Santa Cruz, California

I have made this letter longer than usual, because I lack the time to make it short.
—Blaise Pascal, *Lettres Provinciales*, #16, 1656–1657

Contents

1. Introduction	3
2. Description of Vaults	3
2.1 Vault Temperature Stability	4
3. Field Procedures	5
3.1 GPS Timing Stability <i>versus</i> WWVB Radio	5
4. Description of the Data	6
5. Acknowledgements	7
6. References	7
7. Appendix A: Vault Construction Checklist	8
8. Appendix B: Standard Field Notes for Transect	12
9. Appendix C: Script <i>cutpaste</i> and Examples of Its Use	19
10. Appendix D: Script <i>seltran</i> and Program <i>sum2cut</i> Source Code	22
11. Appendix E: Program <i>dz</i> Example and Source Code	27
12. Appendix F: Manual Page and Source Code for <i>ahwwwvb</i>	42
13. Appendix G: Manual Page and Source Code for <i>ahclk</i>	77
14. Appendix H: Manual Pages and Source Code for Portions of <i>librmf</i> and <i>libq</i>	88
15. Tables	118
16. Figure Captions and Figures	139

1. Introduction

We are searching for the San Andreas fault. The surface trace of the fault is very well mapped, of course, but the location of deeper shearing, particularly in the Earth's upper mantle, is poorly known. The location of this shearing is a matter of ongoing conjecture and study (e.g., *Furlong et al.*, 1989), and is important because it constrains and may modify interpretations and modeling of Earth-strain measurements. These, in turn, affect the evaluation and mitigation of earthquake hazards, particularly for areas with long quiescent periods between earthquakes.

We are using a polarization effect called "shear-wave splitting" in the waves of distant earthquakes to identify and characterize deforming regions in the mantle (cf. *Silver and Chan*, 1991). Succinctly, the slow, shearing distortions of the upper mantle line up crystals in those rocks. These crystal alignments affect seismic wave velocities, making them depend strongly on the polarization of the waves and the direction of wave travel. One result is that the shear wave splits into two waves of slightly different speeds. The time lag between these waves' arrivals and the polarization direction of the waves allow us to measure the crystal alignment, and from that infer the location, direction, and amount of shearing in the mantle. This mantle shearing distorts the Earth's overlying crust, causing earthquakes.

This "Transect" experiment was to be the prototype of its genre, to demonstrate the technique's efficacy in a relatively simple region possibly sporting a large lateral offset between surface and mantle shearing locations. The southern part of the San Francisco Bay Area has a small number of subparallel major faults (San Andreas, Hayward, and Calaveras) and some authors (e.g., *Furlong et al.*, 1989) propose that the mantle shearing is occurring near the Hayward and Calaveras faults, some 40-60 km east of the San Andreas. Hence, we designed a linear array of seismographs running perpendicular to the fault traces (Figure 1a).

Because of uncertainty in the distribution of mantle shearing, and to evaluate older and neighboring tectonism that could affect our interpretations, we spanned the whole region from near the coast, north of Santa Cruz, California, to the foothills of the Sierra Nevada, near Columbia, California (a State historic park in the Mother Lode). We planned seven to eight broadband-seismograph vaults along this line, spaced to obtain the maximum detail available from these (long wavelength) shear waves. Our array also takes advantage of two permanent broadband seismographs operated by the University of California, Berkeley (stations MHC at Mount Hamilton, and CMB at Columbia). Each of our vaults was to be occupied by continuously recording digital seismographs and modern broadband seismometers whenever such equipment was available. We planned a total of about 12 months recording, distributed over about three years, and mostly from Autumn into Spring, due to the limited availability of such equipment.

This project was begun in December, 1993, then cancelled unexpectedly in October, 1994. It is not clear whether it will be restarted by the USGS or others in this working group in a later year. In our one year of operation, we installed three of the vaults (Table 1), obtained preliminary data from two of them, developed and refined all major field procedures and major data processing methods and software, obtained most materials required for the remaining vaults, and were close to receiving permission for two more vault sites. Indeed, permitting sites proved the most onerous part of the study, since each vault's construction disrupts a moderately large area (about 10×10 m) and requires dirt-road travel in the wet season, while USGS policy and budget do not permit substantial compensation to landowners. We are deeply indebted to the landowners who allowed us to construct these vaults.

This report describes the methods we developed, releases the data obtained to date, and releases some related software, particularly a reliable, robust automatic WWVB time-code reader.

2. Description of Vaults

The necessity of using broadband instruments (which require both hard rock sites to mitigate site-response anomalies, and thermal stability to reduce long-period instrumental noise); anticipated rainy season operations over a period of several years; and the high value of the equipment to be used (~\$40k per site), drove the decision to build semi-permanent, secure, winter-rated, hard-rock vaults. We explored numerous design alternatives, and the practices of other investigators using temperature- and tilt-sensitive equipment. Budgetary limitations necessitated a simple, relatively inexpensive design—our solution is illustrated in Figure 1b. Indeed, construction of these vaults proved to be largely a one-day task, with an expenditure of about \$2k per site in materials

and contracts.

The vault consists of two short sections of 42-inch diameter (107-cm) galvanized steel culvert placed on end in an excavation, with at least the longer of them bottoming on bedrock (the "sensor vault"). A four-inch diameter (10-cm) ABS plastic pipe connects the culverts below ground, so that signal and power lines may pass between the "sensor vault" and the "recorder vault". In the most recent variant, standard T and U fittings and two-inch (5-cm) ABS pipe create a rain-resistant cable route to the surface near the recorder vault, to support external solar panels and time-code receivers. All three ends of the ABS fixture are pressure fit with removable polyethylene-foam stoppers, to reduce air circulation, water-vapor condensation, and invasion by animals such as endemic black widow spiders and cable-chewing rodents. The entire assembly, including the joints in the culvert, is sealed with industrial-grade silicon rubber caulking. Both culverts are floored by four to six inches of sidewalk-grade premixed concrete (10-15 cm; about $\frac{3}{4}$ ton). A few strands of heavy-wire reinforcing are used in any recorder vault not bottoming on bedrock. Once adequately cured (i.e., about a week after pouring), the concrete is sealed with a commercial mix of LatexTM and Portland cement—a widely available basement sealer. In addition to choosing sites with relatively deep water tables, the various sealant efforts to date have been effective in eliminating water seepage into the vaults, reportedly a major problem with some vault designs. Even so, some condensation, particularly on the upper parts of the vault walls and on the inside of the lid was observed, and more so in the less-insulated recorder vault.

Each culvert section is sealed by a custom-made steel lid (\$140 each, \$280 per site) fabricated by a local metal-spinning shop. They are of eighth-inch thick (3.2 mm) galvanized steel, with handles of bent stainless steel bar welded to two opposite sides. The inside of each lid is covered with a two-inch thick (5-cm) layer of polyethylene foam for thermal insulation, lid-vibration damping, and the vault air-and-animal seal. Hardened steel chain passes over the lid, through each handle, and down into the concrete on either side for security. The chain is pulled tight to prevent wind damage and lid rattling. They are bolted to the culvert at the bottoms and again about one foot (30 cm) below the lid, the latter to prevent the chain from being pulled off to the side of the lid, and the vault opened. The chains are locked with hardened padlocks having boron-steel hasps. This, and any other security system, can be overcome—site-access control and low site visibility from traveled roads and trails remain the principal sources of security. Remote areas on ranches often are optimal in this regard.

Inside the sensor culvert, a $\frac{1}{4}$ -inch plywood inner lid supported by "2×4" (inch) studs maintains free air-space around the seismometer, and holds up bags of polystyrene foam between the inner and outer lids. This foam and the surrounding soil are the thermal insulation for the sensor. The thermal "mass" of underlying rock and concrete, and of the sensor package, damp remaining temperature variations. The foam layer should be at least 24 inches thick (61 cm), but was only about 18 inches thick (46 cm) in early vaults, including the one tested for thermal stability (Section 2.1).

Vault construction requires about one day of labor by about three people, plus about six hours (and travel time) for a full-sized backhoe *equipped with a narrow bucket and "rock teeth"*. A standard 90-psi jackhammer also may be required for more massive rock—the contractor should bring several spare bits since they are likely to dull or break in such use. Bids for backhoe services ranged from \$60/hour to \$100/hour in central California, with \$65-\$75/hour being the norm. Jackhammer rentals typically ranged from \$140 to \$150 per day, with compressor. We paid about \$800-\$900 per vault for these services. An additional day of labor by two people is required to prepare the culvert vault modules and other items (cleaning, sealing, edge preparation, and installing lids and chains), and to load two pickup trucks with the required items. Finally, another two hours are needed about a week later, after the concrete cures, for applying concrete sealant and affixing metal sensor baseplates (which we attached with concrete-patch material), and for curing of these items to the point where an instrument can be installed. Our materials and equipment checklist for these operations is in Appendix A.

2.1 Vault Temperature Stability

To test the sensor vault for thermal stability, we used a Type-K thermocouple with an inexpensive commercial adaptor (FlukeTM 80TK) designed for use with digital volt-meters. Output of the adaptor (1 mV/°C) was recorded at a low continuous sample rate on an available seismograph channel. The RefTekTM 72A-07 used has a ~ 0.5 M Ω input impedance, while FlukeTM DVMs have 10 M Ω input impedance. Based on conversations with FlukeTM engineers and a simple shunt resistor test on a FlukeTM DVM, this difference depressed temperature readings about 0.4°C on the RefTekTM at room temperature—less than the precision of the thermocouple ($\pm 1.1^\circ\text{C}$) and the 80TK ($\pm 2^\circ\text{C}$) at 20°C. On a RefTekTM 72A-02, with ~ 2 M Ω input impedance, the reading would be depressed about 0.1°C. The internal 9-V battery of the 80TK appears to operate it continuously for

several weeks, at least. (Note that the battery-check mode of the Fluke™ 80TK fails completely for anything but a 10 MΩ impedance.)

We made several abortive attempts to use this setup, which yielded erratic voltage anomalies but also suggested large ($\sim 6^{\circ}\text{C}$) diurnal fluctuations in vault air temperature near the seismometer. We guessed that condensation on the exposed thermocouple tip might be altering leakage-current resistances, and subsequently immersed the thermocouple tip in heat-sink silicon grease. We also put the thermocouple tip in thermal, but not electrical, contact with the dome of the Streckeisen™ STS-2 inside the vault. Because of the large suspected diurnal cycle, at this point we also stacked two wooden freight pallets on top of the vault lid and covered them with a canvas tarpaulin, open on the north but shading the vault lid fully. (The alternative is burying the vault lid under 20 cm of soil.) In combination, these techniques seemed to work, providing credible data and a greatly reduced diurnal cycle. The 5.8-day long record for this arrangement is shown in Figure 2. It indicates a one-day exponential cool-down period, followed by a -0.3°C peak-peak diurnal cycle, plus a long linear drift that may be due to outside weather changes (the drift is -0.2°C/day). The diurnal cycle has a maximum of about 0.4°C peak-peak and RMS of 0.084°C .

With the 36-inch long (91-cm) culvert section of our early sensor-vault design, protruding about 3 inches (8 cm) from the ground, a 12-inch (30-cm) air gap at the bottom, and four inches (10-cm) of concrete, there are only 17 inches (43 cm) left for styrene insulation from the ground surface down to the inner lid. We conclude that this 43-cm insulation layer is inadequate without additional surface insulation. The vault design in Figure 1b shows a 42-inch long (107 cm) culvert section for the same reason. We have not determined whether this culvert length (yielding a 58-cm insulation layer) is sufficient. Providing shade to the vault lid is a reasonable precaution in all cases, only being careful not to create any wind-driven rattle in these surface-insulating materials.

3. Field Procedures

For several reasons, we chose a very cautious and thorough operational procedure. This study was the first-author's first with broadband seismometers and IRIS/PASSCAL RefTek™ recorders (called a "DAS" by PASSCAL). The vaults are few, expensive, and widely spaced. Suitable teleseisms ($\Delta \approx 100^{\circ}$ and $m_b \geq 6.0$) are rare. Most of those recorded in the western United States have northwest, southwest, or southeast azimuths—about parallel or perpendicular to anticipated anisotropy axes. So they will excite only one of the mantle's polarization directions and cannot generate the other polarization of shear wave, and there is no shear-wave splitting to measure. Hence, it is desirable that all stations run with high reliability, to be certain of capturing the rare useful events. To obviate events lost by triggering errors, we recorded continuously. Available disk sizes allowed a maximum of 10 to 20 samples per second, which is sufficient for teleseismic work but not really adequate for the local and regional earthquakes that we also recorded. Servicing intervals ranged from one to two weeks, depending on sample rate.

To speed field operations and allow more reliable data recovery, we swapped disks rather than copying data in the field. Lastly, we anticipated that some servicing personnel would be relatively inexperienced with these instruments. Our resulting field procedure is described best by our field notes form (Appendix B). It is methodical, detailed, and deliberately redundant.

3.1 GPS Timing Stability versus WWVB Radio

Because of reported problems with timing in some older PASSCAL instruments, particularly with Omega radios, we chose to use the new GPS time-base receivers, which are used to correct the internal RefTek™ TCXO at hourly intervals whenever a GPS lock can be obtained. We also chose to record WWVB radio time-code as a backup. We recorded 80-s bursts of WWVB at 250 samples per second every three hours, from True Time™ 86-3 receivers. The recording window began 10-s before the minute, yielding one full code cycle and 10-s end buffers. Because this receiver first detects the signal, then attempts to clean it up with a comparator circuit and signal-following threshold, the time of signal transitions (which mark the seconds) can be more variable than in the raw, detected signal. One of us (J. Evans) earlier developed a successful WWVB time-code reader (*ahwwvb*—Appendix F) that preprocesses the signal through running-median filters (Evans, 1982) and an edge-finding algorithm. This processing cleans the signal in a manner that preserves the transitions times of the input signal, and is highly robust to radio noise typical of WWVB. Hence, we chose to record only the detected output of the receiver (pin E, "DC test", of the 86-3).

Reading these time-code bursts with program *ahwwvb*, and comparing these automatically derived time readings to the GPS-tracking internal RefTek™ time base, yielded very hopeful results (Figure 3). No GPS errors were detected, and the two time bases tracked each other very closely—nearly always within one sample interval of the WWVB time-code (4 ms). The new GPS time base receivers appear to be very reliable in the field, with all detected errors attributable to noise and reading errors of the WWVB time-codes.

Several WWVB segments were unreadable because of a large spike very near the minute mark. This spike may be caused by power cycling in the RefTek™, from which the WWVB receiver draws its power; hence, it may be appropriate to move the WWVB recording window off the even hour. However, we have not pursued the cause of these occasional spikes.

Bob Busby of PASSCAL reports (E-mail, 05 January 1994): "We have compared the accuracy of Omega and GPS external clocks to a pulse input from a GOES and find the PHASE ERRORS reported by the DAS to be correct, when properly interpreted. There are rare cases during a hardware failure of the clock that things go awry. There also remain occasional mislabeling of packets by the DAS which can result in a file start time mislabeled by a sample or two—these errors are tracked in the .err file *ref2segy* produces." Those phase errors are generally well under 1 ms.

4. Description of the Data

In 1993, we obtained data at two vault sites (SFT2 and SFT5, Table 1) during two brief recording periods, one in the Spring and one in the Summer (Table 2). The most useful data set is from the Summer recording period, which was mostly with Streckeisen™ STS-2 broadband sensors, and was largely simultaneous at the two vaults. Events extracted from these continuous records are described in Table 3. Time bases for both recording periods were GPS clocks; the Spring period had WWVB backup, successfully verifying GPS accuracy.

In the field, data disks were swapped to save time and reduce field-operations errors. In the lab, these data were double copied to ExaByte™ tapes, first with *wrendump*, a simple image of the disk, and then with *ref2segy* (to disk) and *tar* (disk to ExaByte™), making a set of PASSCAL SEG-Y files in day-sorted subdirectories. These SEG-Y files were then checked with the *pql* program for quality control and event discovery or event evaluation. As the experiment progressed, we developed partially automated techniques for event identification. (An effort to start a triggered channel to collect a local trigger list was abandoned. It was impeded by the high microseism levels on broadband instruments, difficulties in making the triggered stream work consistently, and our wish to extract every conceivably useful event for coauthors and the IRIS DMC. Further efforts in trigger tuning might be effective, but the ready availability of near-real-time catalogs for this region made such efforts redundant.)

We recording all three components continuously at 10 sps (samples per second) when using the RefTek™ 72A-02 (Spring recording period) and at 20 sps when using the RefTek™ 72A-07 (Summer), since the latter does not support 10 sps (it lacks the DSP chip required to compute the necessary decimation filters). Lists of events were gathered from the CalNet regional 1-Hz network operated by the USGS, and from the USGS National Earthquake Information Service (NEIS), which monitors several world-wide and regional networks. Data from both centers are available on-line. We used the QED service of NEIS (Appendix E) via *telnet*, and the *qfetch* program supported here in Menlo Park for CalNet (Appendix D). We produced software to sift each data set and produce input for our event-extraction script, *cutpaste* (Appendix C), which runs several PASSCAL-supported programs to extract events from the continuous data stream. It remains necessary to move continuous files between day-directories manually when events overlap day boundaries in the continuous stream.

The IRIS DMC, with this Open-file Report, receives for general distribution copies of the *wrendump* tapes and *tar* tapes of all extracted events, the latter in both SEG-Y and *xdr-ah* format.

Our philosophy and the details of event extraction evolved over the course of this experiment. We eventually settled on extracted generous windows around every QED teleseism with any *pql*-visible phase on station SFT5 (often only a weak surface wave), plus local and regional CalNet events as follows: two annuli are taken about SFT5, extracting all events of $M_L \geq 2.0$ from 0 to 200 km away, and all events of $M_L \geq 3.5$ from 200 to 400 km away. A few smaller local events noted during *pql* scans were also extracted, but this latter effort was intermittent. Teleseisms were extracted for windows from one minute before the earliest *iasp91* phase to at least one minute after the last visible surface wave. This is a very generous extraction policy, so many of the extracted events are very small. Only one large SKS phase at appropriate distance was obtained (event "93219.001221")

during the Summer run), which has about 25 s of *SKSac* and *SKKSac* before the *S* arrival (Figure 4). The large event near Guam also was captured (event "93220.084555"). Both these events were recorded with STS-2 sensors at both SFT2 and SFT5 at 20 sps.

5. Acknowledgements

We wish to thank the generous landowners who let us build and operate these vaults (Helen Gerber and the Wells family at SFT5, J. Howard Craven at SFT2, and Gary Stoddard at SFT4), and in all cases showed special interest, thoughtfulness, and forgiveness of our learning curves. Such people make this work possible and often a joy. In addition, we would like to thank the many people who helped us find vault sites or install the vaults. Their names are numerous, but Larry Cox, Arlene Evans, Bud McCrary, Satish Sheth, Roberta Smith, Gene Tobar, David Toshikian, Jim Weaver, and Gerry Weber stand out.

We thank the IRIS consortium and PASSCAL program for the use of their instrumentation and the generous assistance supplied by both PASSCAL instrumentation centers, particularly Bob Busby, Marcos Alvarez, Steve Michnick, and Paul Friberg. Bernard Chouet gave us liberal access to his StreckeisenTM STS-2 throughout this recording season, and Joe Fletcher gave us full use of his two low-power STS-2's during the Summer recording interval. Barbara Romanowicz and many others at the University of California, Berkeley and Santa Cruz, gave advice and support. Bob Urhammer performed noise analysis for SFT2 and SFT5. Malcolm Johnson and Doug Myren provided extensive and invaluable advice on thermally and geometrically stable vault design issues from their broad experience with tilt meters.

Several stalwarts helped us in the heavy construction of these vaults, and the sometimes arduous preparations for it. They are Marcos Alvarez, Bill Lutter, Gonzalo Mendoza, Angus Miller, and Alwyn Ross.

The following are trade names, trade marks, or model names of various organizations. They are cited subject to the disclaimers on the title page of this report: 72A-02, 72A-07, 80TK, 86-3, Brunton, DAS, EHT, Exabyte, Fluke, IRIS, Latex, PASSCAL, RefTek, SCSI, SEG-Y, Sharpie, Streckeisen, STS-2, Trimble, and True Time. Some of the software names used are also recognized product names of various organizations, cited as above (at least, *fsc*, *ftp*, *pql*, *QED*, *qfetch*, *qselect*, *ref2seg*, *segymerge*, *seg2ah*, *tar*, *telnet*, *troff*, and *wrendump*).

Lastly, we would like to thank Mother Earth, which put on the most remarkable display of California Spring wildflowers we have ever seen—and the first two authors are California natives.

6. References

- Evans, J. R., Running median filters and a general despiker, *Bull. Seismol. Soc. Am.*, **72**, 331–338, 1982.
- Furlong, K. P., Hugo, W. D., and Zandt, G., Geometry and evolution of the San Andreas fault zone in northern California, *J. Geophys. Res.*, **94**, 3100–3110, 1989.
- Silver, P. G., and Chan, W. W., Shear wave splitting and subcontinental mantle deformation, *J. Geophys. Res.*, **96**, 16,429–16,454, 1991.

7. Appendix A: Vault Construction Checklist

The following file is available via anonymous *ftp* on computer "andreas.wr.usgs.gov", in file "~ftp/pub/outgoing/evans/equipment_list".

TOOLS AND MATERIALS TO TAKE FOR THE CONSTRUCTION OF ONE SF-BAY TRANSECT VAULT (August 1993)

(C = vault Construction; I = vault finish and instrument
Installation; S = regular Servicing of instrument)

Class:	For:		
Item	C I S	Quantity	
Materials:			
Tubs (with chains & locks attached)	* _ _	2 each	
Lids (fit checked, insulation in)	* _ _	2 each	
A389 locks (hardened)	* _ _	2 each	
ABS pipe (4" ID)	* _ _	1 8-foot length	
Foam plugs for ABS pipe	* _ _	2 each	
Inner lid	_ * _	1 each	
dry-wall screws	_ * _	9 each	
legs	_ * _	3 each	
Styrofoam peanuts	_ * _	20 cubic feet	
Small, thick garbage bags	_ * _	20 each	
Recorder vault inner lid	_ * *	as needed	
Resources:			
Vehicles (with spark arrestors)	2 1 1	as shown	
Tie-down rope and rug scraps	* _ _	as needed	
People	4 4 1	as shown	
Making concrete:			
Sidewalk concrete premix (90# bags)	* _ _	20 bags	
Wheel barrow	* _ _	1 each	
or, mixing barrel	* _ _	1 each	
Hoe (for mixing)	* _ _	1 each	
Concrete trowel	* _ _	1 each	
5-gallon water jugs	* 1 _	10 each	
Concrete sealant	_ * *	100 square feet	
Brush for concrete sealant	_ * *	1 each	
Turpentine or paint thinner	_ * *	1 quart	
Concrete patch	_ * *	20 square feet	
Wire brush (surface preparation)	_ * _	1 each	
Scrub brush (surface preparation)	_ * _	1 each	
Digging:			
Backhoe and operator	* _ _	1 each	
Jackhammer, compressor, operator	* _ _	1 each	
Small jackhammer	* _ _	1 each (if rock is fragile)	
Sledge (3#)	* * *	1 each	
Sledge (10#)	* _ _	1 each	
Pick (large; good fit)	* _ _	1 each	

Breaker/Digger bar	* * *	1 each
Cold chisels (rock)	* * *	1 set

General Tools:

MacCloud tool	* * *	1 each
Shovels	* * *	as needed
Crosscut saw for ABS	* - -	1 each
12 V water pump and hoses	* * *	1 set
Metal-cutting saw for culvert	* * -	1 each
Caulking gun	* * *	1 each
Caulking (clear silicon rubber)	* * *	4 tubes
Drill (hand)	* * -	1 each
Bits	* * -	1 each
Crescent wrench	* * *	2 each
Files (metal, flat)	* * *	2 each
Files (metal, round)	* * *	1 each
Rasp (fine, ABS)	* - -	1 each
Solvent (alcohol) and wipes	* * *	4 bottles + 4 boxes
General tool box	* * *	1 set
Soft brush, broom, dust pan	* - -	1 set
4-foot 2X4 (straight edge)	* * *	1 each
Level (2-3' long)	* * -	1 each
Bucket (1-2 gal)	* * *	1 each
Mixing stick	* * *	1 each
Scoop (small)	* * *	1 each
Sponges	* * *	2 large
Duct tape	* * *	1 roll

Site protection:

Plywood covers	* - -	8 small
Pallet (for concrete bags)	* - -	1 each
Tarpaulin (for concrete bags)	* - -	1 each

People protection:

Ear protection	* * -	1 per person
Eye protection	* * -	1 each
Tick repellent and/or insect spray	* * *	2+ bottles/cans
Sun hat	* * *	1 each per person
Sun screen	* * *	1 bottle
Helmets (when near backhoe)	* - -	1 per person
Gloves	* * *	1 pair per person
Hand soap for cleanup	* * *	1 each
3/4" plywood, ~3X6' (to lie on)	* * *	1 each

Fence making (as needed):

Barbed wire	* * -	1 spool
"T-bar" Posts	* * -	15+ each
Post Driver	* - -	1 each
Pliers (fencing)	* * -	1 each
Pliers (locking)	* * *	1 each
Pliers (regular)	* * *	1 each
Wire ties	* * -	60 each (at least)
Wire stays	* * -	24 each (at least)

Instrumentation:

Seismometers	* _	1 3-component set
Jug line	* _	1 3-component set
Low-noise amps	* _	as needed (STS-2)
Battery, 12V, 100 AH	* *	as needed (STS-2)
Solar panel, 40W	* _	1 each
Power regulator for solar	* _	1 each
Terminal strip and leads	* *	1 set
Battery cover	* *	1 each
Spare battery and cable for retrieving a DAS still powered up	* *	1 set
Base plate(s)	* _	1 set
Epoxy for leg anchor	* *	1 set
Manual(s)	* *	1 set
DAS	* _	1 each
Disk, 230 MBytes	* *	1 each
SCSI cable, DAS-Disk	* _	1 each
Power cable, DAS-Disk	* _	1 each
Power cable, Battery-DAS	* _	1 each
Power cable, Battery-Solar	* _	1 each
Grommets for cables	* _	1 per exterior cable
WWVB radio	* _	1 each
Antenna	* _	1 each
Cables (antenna, DAS)	* _	2 each
Lead brick to hold down cables	* _	1 each
GPS clock	* _	1 each
Cable, DAS-GPS	* _	1 each
Battery, 12V, 100 AH	* *	1 each
Solar panel, 40W	* _	1 each
Power regulator for solar	* _	1 each
Thumb screws	* *	1 set
Battery cover	* *	1 each
Manual(s)	* *	1 set
Wooden blocks--DAS off concrete	* _	1 set
Maps, site and adjacent	* *	>=1 set
Master clock	* *	1 each
Box-cable	* *	1 each
WWV radio and antenna	* *	1 set
Desiccant	* *	as needed
Glass tape for desiccant	* *	1 roll
Camera and film	* *	as needed
Brunton compass	* *	2 each
Recording thermometer or thermocouple ..	* *	1 set, as needed
Grommets for antenna leads	* _	2 each
Blank field notes	* *	>=1 per DAS
Last trip's field notes	* _	1 per DAS
Gardening knee pad	* *	>=1 each
Small stool or vault-rim seat	* *	1 each
DVM	* *	1 each
EHT, with external-battery adaptor	* *	1 each
Lap-top PC with fsc software	* *	1 each, if available
Labels for used disks	* _	plenty
Pallet and plywood for shade	* *	2 sets

Office:

Calculator (with trig)	* * *	1 each
Drawing compass	_ * *	1 each
Protractor	_ * *	1 each
Triangle	_ * *	1 each
Notebook for general use	* * *	1 each
Clip board	_ * *	1 each
Labeling pens (Sharpie, etc.)	* * *	several

Other:

Canvass/plastic mat	* * *	as needed
Mirror (small)	* * *	as needed
Mirror (large)	* * _	as needed
Sun umbrella, IR blocking	* * *	1 each
Umbrella stand	* * *	1 each
Backpack for regular gear	_ * *	1 each

8. Appendix B: Standard Field Notes for Transect

The following field notes are designed very conservatively since these are high-value sites that may be serviced by relatively inexperienced personnel. They are part instruction manual and part data sheet. They are specific to our procedures (continuous broad-band recording with dual GPS and WWVB time sources). A copy of the *troff* input script for these notes is in anonymous *ftp* on computer "andreas.wr.usgs.gov", in file "~ftp/pub/outgoing/evans/DAS_SFTransect_data_sheet". It may be a useful starting point for some other PASS-CAL studies. This Open-File Report serves to release them to the public, subject to the limitations cited on the title page.

PASSCAL Station-Visit Instructions and Notes

(form last modified 02 August 93; Evans and Coakley, USGS)

1a. Arrive: ____M/____D/1993 = ____ Julian, ____:____ UTC (all dates and times are to be in UTC).

Purpose: ☐ Regular visit; ☐ Install (do 18a and 20 now); ☐ Remove; Other: _____

1b. Plug in EHT, and then do COMMNCTNS—RECV PRMS: ☐, or explain why not _____

2a. Verify STATION—EXP NAME: SF BAY LP TRANSECT; STATION—EXP CMMNT: P9220 _____

2b. Verify STATION—STN NUMBR; STATION—STN NAME (code); STATION—STN CMNT (full name):

☐ #1, SFT1, SANTA CRUZ VAULT

☐ #5, SFT5, INGRAM CANYON VAULT

☐ #2, SFT2, FERN PEAK VAULT

☐ #7, SFT7, WOODWARD RESERVOIR VAULT

☐ #4, SFT4, SAN ANTONIO VALLEY VAULT

or # ____, ____, _____

3. Weather:

Wind	Temp.	Clouds	Humidity	Precip.	Lightning
<input type="checkbox"/> still	<input type="checkbox"/> cold	<input type="checkbox"/> clear	<input type="checkbox"/> low	<input type="checkbox"/> none	<input type="checkbox"/> none
<input type="checkbox"/> light	<input type="checkbox"/> cool	<input type="checkbox"/> scattered clouds	<input type="checkbox"/> moderate	<input type="checkbox"/> light	<input type="checkbox"/> nearby
<input type="checkbox"/> moderate	<input type="checkbox"/> mild	<input type="checkbox"/> partly cloudy	<input type="checkbox"/> high	<input type="checkbox"/> heavy	<input type="checkbox"/> distant
<input type="checkbox"/> strong	<input type="checkbox"/> warm	<input type="checkbox"/> mostly cloudy			<input type="checkbox"/> active
<input type="checkbox"/> gale+	<input type="checkbox"/> hot	<input type="checkbox"/> fully overcast			<input type="checkbox"/> rare

4. Sensor: ☐ STS-2 (S/N: _____) ☐ Fluke 80TK (set for °C only ☐)

☐ SV/H-1 (S/N's: V=_____, N=_____, E=_____)

S/N(s) read from ☐ sensor, ☐ DAS, or _____

For your information: Expected Channels and Streams

Signal	Channel 1	Channel 2	Channel 3	Channel 4	Channel 5	Channel 6
Vertical	72A-07			72A-02		
North		72A-07			72A-02	
East			72A-07			72A-02
WWVB		72A-02				
Temperature			-02 or -07			

Stream	Signals Recorded	Rate (sps) 72A-02 (-07)	Data Size 72A-02 (-07)	Trigger Type	Comment (expected events/day; cf. step 7)
1	V, N, E	10 (20)	32 (32)	Continuous	3600-s blocks (24)
2	V, N, E	1 (N/A)	16 (N/A)	Continuous	3600-s blocks (24)
3	raw WWVB	250 (N/A)	16 (N/A)	Timed	Every 3 hours (8)
4	V, N, E	100 (20)	32 (32)	Triggered	For locals and regionals (~20?)
5	Fluke 80TK	1 (20)	16 (32)	Continuous	3600-s blocks, °C (24)

- 5a. EHT software version: _____ (UTILITIES—VERSN NOS—FST)
5b. DAS serial number: _____ (UTILITIES—VERSN NOS—ID = DAS faceplate: ☐)
5c. DAS software version: _____ (UTILITIES—VERSN NOS—CPU)
5d. DSP software version: _____ (UTILITIES—VERSN NOS—DSP)

6. Check battery voltages *via* DVM:

Primary DAS supply: _____ VDC (DAS will fail below 11.5 VDC).

This supply is a ☐ 72A-04 (Pins D-A) or ☐ deep-discharge marine battery (measured at lugs).

Solar panel: _____ VDC or ☐ none installed.

Sensor supply: _____ VDC or ☐ no sensor supply installed. ☐ solar panel also installed for sensor.

WWVB supply: _____ VDC or ☐ none or powered by DAS (for 72A-02, circle two as found):

For 72A-02, want DIAGNOSTICS—ACQUISITION—C1—12V. Got C1: OFF CS 12V BOTH

For 72A-07, N/A. Got C4: OFF CS 12V BOTH

7. Check DAS status (UTILITIES—DAS STATE). Record:

KIN OMGA ☐ 500, ☐ 50, ☐ 5; or ☐ NO UTC CLOCK, or UTC _____

ACQ STRT ON (circle one each column)

STOP OFF

EVENTS: _____ (Is "EVENTS" reasonable (estimate *via* stream table, p. 1)? ☐ yes, ☐ no)

BLKS US: _____ Streams: ☐ are as expected,

BLKS AV: _____ or _____

If no Master Clock is available: DAS time is ☐ visually close, or is ☐ fast, ☐ slow by

~ _____ w.r.t. reference standard of _____

8. Check external clock status (UTILITIES—XCLK STAT)

☐ Omega or ☐ GPS; Time of last lock: _____ Julian, ____:____ UTC. TYPE _____

S/N of radio (from box): _____

For Omega clock, verify that leap second is 18 (STATION—XCK SETUP—LEAP SECS): ☐.

(Have you received parameters from this DAS so the value is meaningful?)

9. If Master Clock is available, use it to check and reset DAS clock: TIME—SET TIME—enter next minute—set up Master Clock (5V positive going pulse, thumb wheel to same next-minute)—connect to COMM, press PULSE (but not more than 30-s before the minute), wait for that minute, then:

Old time: _____ Y: _____ J: _____ H: _____ M: _____ S

New time: _____ Y: _____ J: _____ H: _____ M: _____ .000s

Disconnect Master Clock and reconnect GPS, if in use: ☐

10. Verify mass-position offsets: for STS-2, measure VDC with a DVM at host-box Monitor connector (notice the order); for SV/H-1, draw it as you see it relative to the ticks:

STS-2 (old)	Element U: (Pins F-T)	Element W: (Pins F-U)	Element V: (Pins F-V)
STS-2 (new†)	Element U: (Pins F-T)	Element W: (Pins F-U)	Element V: (Pins F-V)
SV/H-1 (old)	Vertical: Connector at this end	North: Connector at this end	East: Connector at this end

☐ † did or ☐ did not recenter STS-2.

11. For STS-2, recenter if any element is more than ± 1.0 VDC. Depress the button on the "host box" for at least 0.5 s. You should hear clicks and motor sounds from the sensor, if it is exposed. The recentering sequence lasts about 30 s. Re-check offsets and recenter again if necessary. Repeat up to 8 times; after that tilting one support legs away from bubble-center may be necessary as a work-around (turn power off and lock all three elements first!), though this is a rarely needed option.

Never open the sensor package in the field (or at all if you haven't been instructed in that risky operation). For that matter, never move an unlocked STS-2, or significantly bump even a locked one—\$20K and a 15-month waiting list—the manufacturer will not repair these instruments.

For SV/H-1, if you doubt centering (due to a dead channel, for example) open the vault and recenter mass to tick marks *via* the knurled support leg of the SH-1 and the knurled spring-tension knob of the SV-1.

☐ vault opened before calibration ☐ vault opened after calibration ☐ vault not opened

Recentered SV/H-1: ☐ Vertical, ☐ North, ☐ East.

Other actions taken: _____

12. If acquisition has stopped and RAM is full, or if installing site: check here ☐ and wait to do this step until after step 24 (that is, after restarting the DAS and shortly before leaving the site).

Calibrate the sensor (72A-02 only): DAS *must* be acquiring data, that is, "START ON" (verify: ☐.

For STS-2 install the jumper plug in the Monitor connector of the host box (☐), press DIAGNOSTICS—ACQUISITION—C4 (it responds "CS"—note this time below) (☐), and leave it on for 15 minutes while you remain *very still* or leave the vicinity (at least 10 m and be quiet). (The "free period" of this instrument is 2 minutes, and the sensor is very tilt sensitive. The C4 relay connects the DAS "test bus" to the input-connector of channels 4, 5, and 6 and starts the calibration sequence. The DAS performs a series of mass-drop calibrations *via* the calibration coils.) After the allotted time, press C4 *three* times, until it responds "OFF" (☐). Lastly, *remove jumper* (☐). (Standard settings: duration 60 s, step ON, interval 360 s, step size 361 s, amplitude 8.0 V, output to COIL.)

For SV/H-1 sites, there is no jumper. Press DIAGNOSTICS—ACQUISITION—C4 (☐). Wait *very quietly* for 2 minutes. Then turn C4 to "OFF" *via* three strokes (☐). (Standard settings: duration 120 s, step ON, interval 30 s, step size 31 s, amplitude 1.0 V, output to COIL.)

Calibration start time: _____ Julian, _____:_____UTC

Turn off EHT until 30 s from end (or it shuts itself off at the most inconvenient moments).

13. **Stop acquisition.** (From the main menu, do COMMNCTNS—STOP ACQ—YES, and wait for it to return to COMMNCTNS menu with a beep)

Time of the beep: _____ Julian, ____:____:____ UTC

14. Verify reasonable sensor signals on active channels (MONITOR—enter 15 s—CHAN, 72A-02: 2,4,5,6; 72A-07: 1,2,3; Temperature: 3):

Channel	1	2	3	4	5	6
Max/Min and Sketch and/or Comment	/	/	/	/	/	/

(Press "X" to exit back to CHAN menu.)

15. Check SCSI status (UTILITIES—SCSI STAT). Spins disk and indicates length used.

Write all 7 digits of length used: _ _ _ _ _ _ _

Do 16a and 16b for 72A-02 OR 16a and 16c for 72A-07:

- 16a. Label old disk (for 72A-02) or transfer disk (for 72A-07) with:

- ☐ Station name (and, if huddled, which sensor(s))
☐ DAS S/N
☐ Time of STOP ACQ (Year:Julian day:hour:minute UTC)
☐ Your initials

- 16b. **Swap disks** (☐ Yes or ☐ N/A) (for 72A-02): remove power cable, remove SCSI cable, attach power cable, attach SCSI cable. Take home the old disk—it contains the data!

Write down the size of the *new* disk you attach: _____ MBytes (typically 230 MBytes).

(If unsure, try UTILITIES—SCSI STAT and write down **total** + 1000.)

- 16c. Copy data from internal disk to transfer disk (for 72A-07) (☐ Yes or ☐ N/A): connect the transfer disk to its battery, wait 15 s for it to spin up, connect its SCSI cable to DAS, do UTILITIES—NEXT MENU—COPY SCSI—(COPY DISK TO) DISK—YES, wait until "WAITING ..." goes off and EHT beeps, disconnect SCSI cable, power down transfer disk. **Remember not to bump or move the transfer disk while it is spinning.**

17. Swap batteries as needed:

☐ none changed

Primary DAS supply: _____ VDC (DAS will fail below 11.5 VDC).

This supply is a ☐ 72A-04 (Pins D-A) or ☐ deep-discharge marine battery (measured at lugs).

If in use, is solar panel properly connected? ☐ Yes

Sensor supply: _____ VDC or ☐ no sensor supply installed. ☐ solar panel also installed for sensor.

18a. Only upon initial site installation (☐) , DAS replacement (☐) , or serious DAS trouble (☐) (check one): Consider doing COMMNCTNS—RECV PRMS, then do a rigorous DAS initialization (UTILITIES—NEXT MENU—SYS INIT, then give password "IRIS"). **Warning:** this function clears *everything* in the DAS, including wiping out your parameter set—everything is set to factory default values. If you do this step (18a), then skip step 18b but **do step 20** (reprogramming the DAS).

Or

18b. Clear the DAS's RAM (UTILITIES—NEXT MENU—CLEAR RAM—YES): ☐

19. Format (that is, erase) the disk (UTILITIES—NEXT MENU—FRMT SCSI—DISK—YES): ☐
(You may see "—WREN" instead, which is an anachronism meaning "—DISK"). Should hear clicks.

20. If necessary, reprogram the DAS, typically as follows:

- (a) Load existing parameters from DAS to EHT (COMMNCTNS—RECV PRMS) OR ☐ done in step 1b.
Upon site installation or other SYS INIT, the EHT *may or may not* have a more correct set of parameters to start from (☐). You *may* need to do COMMNCTNS—LOAD PRMS first (☐).
- (b) Modify parameters on EHT as needed (e.g., add or change sensor S/N's, which are generally unknown to the author of an fsc file). Write any changes in "Synopsis", below.
- (c) Send modified parameters from EHT to DAS (COMMNCTNS—SEND PRMS) (☐).

Synopsis of parameter changes: ☐ none, ☐ all new (fsc file: _____), and/or: _____

21. Verify that you still have reasonable sensor signals (DIAGNOSTICS—OFFSET—set 0:15—CHAN, 72A-02: 2,4,5,6; 72A-07: 1,2,3; Temperature: 3) (At your discretion, instead use MONITOR—enter 15 s—CHAN to get an actual picture of the data):

Channel	1	2	3	4	5	6
Max						
Min						
Ave; or Sketch and/or Comment						

+++++

Perform steps 22 and 23 within 3 minutes of one another (limited by the WWVB stream). This restriction is a work-around for a bug that eats up pre-event memory in DAS software versions 2.46 and 2.47. (For the fastest sampling rate possible, the lag would need to be under 10 s.)

22. Perform system reset (UTILITIES—SYS RESET—YES—CONTINUE): ☐

23. **Start acquisition** (COMMNCTNS—START ACQ—set 0:0—YES and wait for the DAS STATE display).

Time STRT ON is verified by beep and status display: _____ Julian, ____:____:____ UTC

EVENTS: _____

BLKS US: _____

BLKS AV: _____

Streams: ☐ are as expected, or _____

+++++

24. Restart power to the WWVB radio (for 72A-02, circle two as you *actually* see them):

For 72A-02, want DIAGNOSTICS—ACQUISITION—C1—12V. Got C1: OFF CS 12V BOTH

For 72A-07, N/A. Got C4: OFF CS 12V BOTH

Remember to do step 12 (calibration) now, if it was deferred.

25. **Remember not to unplug anything from the DAS except the EHT.**

26. Depart at ____ M/ ____ D/1993 = _____ Julian, ~ ____:____ UTC.

27. Operator(s) name(s):

☐ John M. Coakley; ☐ John R. Evans; ☐ Gonzalo Mendoza

Others present: _____

Comments, and any troubles noted (as needed, use margins (except left margin), and use backs of sheets):

9. Appendix C: Script *cutpaste* and Examples of Its Use

The following listing is the cut-and-paste event-extraction script "*cutpaste*", used to generate event files from continuous-record files. A copy of the *cutpaste* script is in anonymous *ftp* on computer "andreas.wr.usgs.gov", in file "~ftp/pub/outgoing/evans/event_extraction/cutpaste/cutpaste". This Open-File Report serves to release them to the public, subject to the limitations cited on the title page. Following the *cutpaste* listing is a short example of a script that calls *cutpaste* for the desired events. Such calling scripts form a sufficient record of extraction activities and comments (e.g., Table 3 is derived from our own *cutpaste*-calling scripts).

Listing of *cutpaste* Script

```
#!/bin/sh
#
# Merge file for STS-2, SV/H-1, or CMG-3ESP data.
#
# Usage:
# cutpaste ( 1 2 3 | 4 5 6 ) dddd ( sts2 | svh1 | 3esp ) ssss YY JJJ hh mm ss rrr "blah blah"
# argc:          1 2 3          4          5          6 7 8 9 10 11 12 13
#
# where "1 2 3" or "4 5 6" are the triplet of channels to convert,
# "dddd" is the 4-digit serial number of the DAS, "sts2", "svh1", or
# "3esp" are the respective instrument type, "ssss" is the station name,
# "YY JJJ hh mm ss" is the start time of the file, and "rrr" is the
# length of record to produce (seconds), and "blah blah" is a comment.
#
# Produces cut and pasted SEG-Y files (one per channel), named
# "YYMDD.hhmmss.<sts2|svh1|3esp>.ssss.<1|2|3|4|5|6>" and a single
# cut and pasted ah file (containing three channels), named
# "YYMDD.hhmmss.<sts2|svh1|3esp>.ssss.ah".
#
# Bugs: does no checking, just hammers away. Assumes you are asking for
# something reasonable and are in the right directory.
#
# SEG-Y filename format: R<Julian day>.<stream number>/hh.mm.ss.dddd.c
#
# Changed output filename format (reversed station and instrument fields)
# on 01 Sept 93 to make segy2ah pick up the correct station name (which
# it takes from the second-to-last "."-separated field). No change to
# input format (still jug then site).
#
dest=/passcal10/data/SFT/extracts      # Destination directory
#
prog=$0
V=$1
shift ; N=$1
shift ; E=$1
shift ; DAS=$1
shift ; jug=$1
shift ; site=$1
shift ; YY=$1
shift ; JJJ=$1
shift ; hh=$1
shift ; mm=$1
shift ; ss=$1
shift ; len=$1
shift ; comment=$1
```

```
#
echo "-----'pwd'-----"
echo "$prog $V $N $E $DAS $jug $site $YY:$JJJ:$hh:$mm:$ss $len"
#
if [ -f 'ls *.SDAS.[SVNSE] | head -1' ]; then
    echo "Processing..."
    segymerge -c *.SDAS.$V -s $YY:$JJJ:$hh:$mm:$ss -l $len -t 1 > $YY$JJJ.$hh$mm$ss.$jug.$site.$V
    segymerge -c *.SDAS.$N -s $YY:$JJJ:$hh:$mm:$ss -l $len -t 1 > $YY$JJJ.$hh$mm$ss.$jug.$site.$N
    segymerge -c *.SDAS.$E -s $YY:$JJJ:$hh:$mm:$ss -l $len -t 1 > $YY$JJJ.$hh$mm$ss.$jug.$site.$E
#
    if [ -f 'ls $YY$JJJ.$hh$mm$ss.$jug.$site.[SVNSE] | head -1' ]; then
        segy2ah $YY$JJJ.$hh$mm$ss.$jug.$site.[SVNSE]
        cat $YY$JJJ.$hh$mm$ss.$jug.$site.[SVNSE].ah > $YY$JJJ.$hh$mm$ss.$jug.$site.ah
        /bin/rm $YY$JJJ.$hh$mm$ss.$jug.$site.[SVNSE].ah
#
    # Should discard 0-length files here...
    echo "Moving to $dest:"
    ls -la $YY$JJJ.$hh$mm$ss.$jug.$site.[SVNSE] $YY$JJJ.$hh$mm$ss.$jug.$site.ah
    mv -i $YY$JJJ.$hh$mm$ss.$jug.$site.[SVNSE] $YY$JJJ.$hh$mm$ss.$jug.$site.ah $dest
fi
fi
```

Example of *cutpaste* Usage

```
#!/bin/csh
# Run a series of the cutpaste script, starting from directory above
# those (e.g., R216.01) created by ref2segy.
#
#####
# L=local, R=regional, T=teleseism, (L/R at about S-P = 10 s;
# R/T at about distance 3-6 degrees, based on frequency content).
#
# L/R generated from CalNet listings via shell "seltran", which
# calls "qfetch" and "sum2cut" (Appendix D).
# R/T generated from USGS QED (NEIS) system lists, via program "dz"
# (Appendix E).
#####
#
# Start 216:20:13:47
#
# 04 August 93
cd R216.01
#
cutpaste 1 2 3 7064 sts2 sft5 93 216 20 20 07 099 "v sm CalNet L/R, m2.7 d1.57 h4.9"
#
# 05 August 93
cd ../R217.01
#
cutpaste 1 2 3 7064 sts2 sft5 93 217 10 19 31 120 "sm CalNet L/R, m2.8 d2.00 h9.1"
cutpaste 1 2 3 7064 sts2 sft5 93 217 18 43 58 094 "sm CalNet L/R, m1.9 d0.75 h0.9"
#
# 06 August 93
cd ../R218.01
#
```

```

cutpaste 1 2 3 7064 sts2 sft5 93 218 00 31 45 150 "CalNet L/R, m3.1 d2.06 h15.4"
# Unidentified surface wave ~03:34 (similar to offshore N. CA events of late)
cutpaste 1 2 3 7064 sts2 sft5 93 218 11 01 42 098 "sm CalNet L/R, m2.6 d1.45 h8.4"
cutpaste 1 2 3 7064 sts2 sft5 93 218 23 34 31 093 "CalNet L/R, m2.6 d0.54 h7.2"
#
# SPANS DAY BREAK (002320 070*20):
cutpaste 1 2 3 7064 sts2 sft5 93 219 00 12 21 6100 "nice T, @NORTHEAST OF TAIWAN"
#
# 07 August 93
cd ../R219.01
#
cutpaste 1 2 3 7064 sts2 sft5 93 219 06 16 00 1560 "T, OFF COAST OF JALISCO, MEXICO"
cutpaste 1 2 3 7064 sts2 sft5 93 219 13 43 30 120 "CalNet L/R, m3.5 d1.39 h18.4"
cutpaste 1 2 3 7064 sts2 sft5 93 219 18 03 00 15420 "T, @SOUTH OF FIJI ISLANDS"
#
# 08 August 93
cd ../R220.01
#
# Ignore "T, @MINAHASSA PENINSULA, SULAWESI m5.4/4.9 d112 h62; weak spectral-only surface wave"
cutpaste 1 2 3 7064 sts2 sft5 93 220 08 45 55 23650 "T, @SOUTH OF MARIANA ISLANDS (GUAM)"
cutpaste 1 2 3 7064 sts2 sft5 93 220 20 14 40 4340 "T, @MARIANA ISLANDS"
#
# 09 August 93
cd ../R221.01
#
cutpaste 1 2 3 7064 sts2 sft5 93 221 02 19 15 195 "CalNet L/R, m3.0 d2.07 h15.4"
cutpaste 1 2 3 7064 sts2 sft5 93 221 02 49 00 900 "Unidentified surface wave"
cutpaste 1 2 3 7064 sts2 sft5 93 221 05 32 45 150 "CalNet L/R, m3.0 d2.46 h2.5"
cutpaste 1 2 3 7064 sts2 sft5 93 221 09 26 42 4160 "T, @MARIANA ISLANDS m5.3/5.5 d84 h60"
cutpaste 1 2 3 7064 sts2 sft5 93 221 11 50 00 12300 "T, @HINDU KUSH REGION"
cutpaste 1 2 3 7064 sts2 sft5 93 221 21 40 18 094 "CalNet L/R, m2.1 d0.69 h5.4"
cutpaste 1 2 3 7064 sts2 sft5 93 221 23 08 41 099 "v sm CalNet L/R, m2.1 d1.53 h12.9"
#
# 10 August 93
cd ../R222.01
#
cutpaste 1 2 3 7064 sts2 sft5 93 222 01 04 57 15603 "T, @OFF W. COAST OF S. ISLAND, N.Z"
cutpaste 1 2 3 7064 sts2 sft5 93 222 05 52 56 101 "CalNet L/R, m2.6 d1.98 h0.1"
cutpaste 1 2 3 7064 sts2 sft5 93 222 09 04 50 12970 "T, @MARIANA ISLANDS"
#
# Stop 222:18:41:27

```

10. Appendix D: Script *seltran* and Program *sum2cut* Source Code

The script *seltran* runs USGS program *qfetch*, which finds the most current CalNet local-earthquake information for a specified date range. This result is piped to USGS program *qselect* to extract two distance-magnitude sets. The selected event summary lines are finally translated by program *sum2cut*, listed below, into the form used by shell *cutpaste* to extract events from the continuous data stream.

Both the source code and the script are available *via* anonymous *ftp* from "andreas.wr.usgs.gov" in directory "~ftp/pub/outgoing/evans/event_extraction". This Open-File Report serves to release them to the public, subject to the limitations cited on the title page.

Script *seltran*

```
#!/bin/csh
# Shell to gather events for extraction from SFT continuous data.
#
# Run two annuli about SFT5, taking all events of ML>=2.0 out to
# 200 km, and ML>=3.5 from 200 to 400 km. Then run sum2cut on
# each file to convert formats and set time windows.
#
# USAGE: seltran date1 date2 > output_file
#          $1      $2
# Format of dates is that of qfetch: [CC]YYMMDDhhmmss[+-offset]
#
# John R. Evans, USGS, Menlo Park, 09/93.
#
rsh -n andreas "qfetch -time $1 $2 | \
  qselect -magnitude 2.0 10.0 -delta 37.505053 -121.328514 0.0 200.0" > \
  seltran.temp1.xxx
#
rsh -n andreas "qfetch -time $1 $2 | \
  qselect -magnitude 3.5 10.0 -delta 37.505053 -121.328514 200.01 400.0" >> \
  seltran.temp1.xxx
#
sort seltran.temp1.xxx > seltran.temp2.xxx
/bin/rm seltran.temp1.xxx
#
sum2cut < seltran.temp2.xxx
/bin/rm seltran.temp2.xxx
```

Source Code of *sum2cut* Follows:

```

/*
 * SUM2CUT (SUMmary card to CUTpaste formatter)
 *
 * USAGE:  seltran < summary_cards > output_file
 *
 * BUGS:
 *
 * Update CENTURY appropriately.
 *
 * Station coordinates list hardwired.  Change STA structure, and
 * NSTA appropriately.
 */

#include <stdio.h>
#include <math.h>
#include <string.h>
#include <local/local.h>
#include <local/stdtyp.h>
#include <local/mathconst.h>
#include <local/date_time.h>

#define MAX(a,b) (((a)>(b)) ? (a) : (b))      /* Larger value */

#define MAXBUF      40      /* decode.h buffer size */
#include <local/decode.h>

#define STR_LEN      256      /* Maximum scratch-string length */

#define CENTURY      1900
/* Coordinates of possible station(s) from
   which to calculate delta and azimuth */
#define SFT2NAM      "sft2"
#define SFT2LAT      37.153042
#define SFT2LON      -121.812232
#define SFT2ELEV      518

#define SFT4NAM      "sft4"
#define SFT4LAT      37.38883
#define SFT4LON      -121.49450
#define SFT4ELEV      660

#define SFT5NAM      "sft5"
#define SFT5LAT      37.505053
#define SFT5LON      -121.328514
#define SFT5ELEV      312

#define NSTA          3

typedef struct {
    char    code[5];      /* Station information */
    double  lat;          /* Name */
    double  lon;          /* Location (+N, degrees) */
    double  elev;         /* (+E, degrees) */
    /* meters */
} STA;

typedef struct {
    TIME    ot;           /* Station information */
    double  lat;          /* Origin time */
    double  lon;          /* Epicenter (+N, degrees) */
    /* (+E, degrees) */
} EVENT;

#define strneq !strncmp

void    datetime();      /* -lq (YMDhms <- epochal) */
void    delaz();         /* -lq (geocen.c) */
void    error();         /* -lq (error.c) */
void    exit();          /* UNIX C library */
bool    isleap();        /* -lq */
void    read_ev();       /* Defined below */
void    refpt();         /* -lq (geocen.c) */
TIME    timvar();        /* -lq (YMDhms -> epochal) */

main(argc, argv)
    int    argc;
    char    **argv;
{
    char    in_str[STR_LEN]; /* Input buffer */

    int     styr, stmo, stday, sthr, stmn;
    double  stsec;
    double  delta, az0, az1;

    int     yr, mo, day, hr, mn, jday;
    double  sec, lat, lon;
    double  depth, mag;

    double  Ptime;

```

```

int      seci, dur;
TIME     start_t;

int      ii;          /* Dummy */

STA      stations[10]; /* Station list */

/* Build station list the dumb way */

(void)strncpy(stations[0].code, SFT2NAM, 5);
stations[0].lat = SFT2LAT;
stations[0].lon = SFT2LON;
stations[0].elev = SFT2ELEV;

(void)strncpy(stations[1].code, SFT4NAM, 5);
stations[1].lat = SFT4LAT;
stations[1].lon = SFT4LON;
stations[1].elev = SFT4ELEV;

(void)strncpy(stations[2].code, SFT5NAM, 5);
stations[2].lat = SFT5LAT;
stations[2].lon = SFT5LON;
stations[2].elev = SFT5ELEV;

if (argc > 1)
    error ("Usage: %s < summary_cards > output_file", argv[0]);

/* Main loop--read summary card, output NSTA cutpaste lines */
while (fgets(in_str, sizeof(in_str), stdin) != NULL) {
    read_ev(in_str, &yr, &mo, &day, &hr, &mn, &sec,
            &lat, &lon, &depth, &mag);

    /* Loop over stations for output */
    for (ii = 0 ; ii < NSTA ; ii++) {

        /* Infer record window */
#define PAD_BEG 30.0 /* Pad record window at front */
#define PAD_END 60.0 /* Pad record window at rear */
#define DEG2KM 111.2
#define P_VEL 5.0 /* Nominal P velocity (km/s) */
#define LR_VEL 28.16 /* Nominal surface-wave velocity (s/degree) */

        refpt((stations[ii].lat * RAD), (stations[ii].lon * RAD));
        delaz(lat * RAD, lon * RAD, &delta, &az0, &az1);
        delta *= DEG;
        az0 *= DEG;
        az1 *= DEG;

        /* Start at ~P time - PAD_BEG */
        Ptime = delta * DEG2KM / P_VEL;
        start_t = timvar(CENTURY + yr, mo, day, hr, mn, sec,
                        GREGORIAN);
        start_t += Ptime - PAD_BEG;
        (void)datetime(start_t,
                        &styr, &stmo, &stday, &sthr, &stmn, &stsec,
                        GREGORIAN);
        jday = yrdays(stmo, stday, isleap(styr, GREGORIAN));
        seci = stsec;

        /* End at nominal surface-wave time + PAD_END */
        dur = (delta * LR_VEL - Ptime) + PAD_BEG + PAD_END;

        /* Output */
        (void)printf(
            "cutpaste 1 2 3 dddd jjjj %s ",
            stations[ii].code);
        (void)printf(
            "%2d %03d %02d %02d %02d %03d \"CalNet L/R, ",
            sty, jday, sthr, stmn, seci, dur);
        (void)printf(
            "m%3.1lf d%2.1lf h%1.1lf, @%5.2lfN%7.2lfE\\n\\n",
            mag, delta, depth, lat, lon);
    }
}

return(0);
}

/*
 * Read an event line from QEDs defaulting numbers to zero and
 * strings to null if not present. Returns TRUE if it seems to
 * be an event line, otherwise FALSE.
 */
930408 0032 53.41 37 21.36 121 43.04 8.24 1.6 42 58 3. 0.07 0.1 0.4 A2
930726 0310 18.27 36 50.92 121 35.26 9.16 2.3 60 39 3. 0.16 0.2 0.4 B2
*/

```



```
void
read_ev(in_str, yr, mo, day, hr, mn, sec, lat, lon, depth, mag)
    char    in_str[];          /* Input buffer          */
    int     *yr, *mo, *day, *hr, *mn; /* Origin time      */
    double  *sec;
    double  *lat, *lon;        /* Epicenter        */
    double  *depth;            /* h (km)           */
    double  *mag;              /* ML               */
{
    double  latm, lonm;

    *yr = 0;
    *mo = 0;
    *day = 0;
    *hr = 0;
    *mn = 0;
    *sec = 0.0;
    *lat = 0.0;
    *lon = 0.0;
    *depth = 0.0;
    *mag = 0.0;

    *yr = DECODE(in_str, 2, atoi);
    *mo = DECODE((in_str + 2), 2, atoi);
    *day = DECODE((in_str + 4), 2, atoi);
    *hr = DECODE((in_str + 7), 2, atoi);
    *mn = DECODE((in_str + 9), 2, atoi);
    *sec = (double)DECODE((in_str + 12), 5, atof);
    *lat = (double)DECODE((in_str + 18), 2, atof);
    latm = (double)DECODE((in_str + 21), 5, atof);
    *lat = *lat + latm / 60.;
    *lon = (double)DECODE((in_str + 27), 3, atof);
    lonm = (double)DECODE((in_str + 31), 5, atof);
    *lon = -(*lon + lonm / 60.); /* Make E+ */
    *depth = (double)DECODE((in_str + 38), 5, atof);
    *mag = (double)DECODE((in_str + 46), 3, atof);
}
```

```

#
#-----*-----*-----*-----*-----*-----*-----*-----*-----*
#
# Generic Makefile for programs.
# Works for any mix of sun3, sun4, Fortran, and C (I hope!).
#
# "make install" fails the first time unless the executable has
# already been copied to $(BINDIR).
#
# The following is required in the user's .cshrc:
#
# setenv ARCH `/bin/arch`
# if( $ARCH == "sun3") then
#     setenv FLOAT -f68881
# else
#     setenv FLOAT ""
# endif
# setenv DIR_USR /usr/local
# setenv DIR_SRC /usr/src/local

FC      = f77
CC      = cc
FFLAGS  = -u $(FLOAT) -D$(ARCH)
CFLAGS  = $(FLOAT) -D$(ARCH) -g
BINDIR  = $(DIR_USR)

PROG    = sum2cut
SRCARC  = $(PROG).src.a

FSRCS   =
CSRCS   = sum2cut.c
SCRIPT  = seltran

OBJECTS = $(FSRCS:%.f=$(ARCH)/%.o) $(CSRCS:%.c=$(ARCH)/%.o)

LIBS    = -lq -lm
LLIBS   = $(LIBS)

INCLS   =
OTHERS  = Makefile $(SCRIPT).sh $(INCLS)
OTHERBIN=

# P      = enscript -r2 -p-

$(ARCH)/$(PROG): $(OBJECTS)
    $(CC) $(CFLAGS) $(OBJECTS) $(LIBS) -o $@

$(OBJECTS): $(INCLS)

$(ARCH)/%.o: %.c
    $(CC) $(CFLAGS) -c $(@F:.o=.c) -o $@

$(ARCH)/%.o: %.f
    $(FC) $(FFLAGS) -c $(@F:.o=.f) -o $@

install: $(ARCH)/install $(ARCH)/$(PROG) $(SCRIPT).sh
#
# /bin/rm install
#
# ln -s $(ARCH)/install install
# /bin/mv $(BINDIR)/$(PROG) $(BINDIR)/$(PROG).old
# cp $(ARCH)/$(PROG) $(BINDIR)
# chmod 755 $(BINDIR)/$(PROG)
# cp $(SCRIPT).sh $(BINDIR)/$(SCRIPT)
# chmod 755 $(BINDIR)/$(SCRIPT)
# touch install
# touch $(ARCH)/install

# CAUTION: do not run "make clean" on more than one machine at a time;
# the archive commands ("ar") might conflict.
clean:
    /bin/rm $(ARCH)/$(PROG) $(OBJECTS) $(BINDIR)/$(PROG).old
    ar ruv $(SRCARC) $(FSRCS) $(CSRCS) $(OTHERS) $(OTHERBIN)

lint: $(FSRCS) $(CSRCS) $(INCLS)
    /usr/bin/lint -xh -u $(FSRCS) $(CSRCS) $(LLIBS) > lint

print: $(FSRCS) $(CSRCS) $(OTHERS)
#
# pr -l66 $? | $P
# enscript $?
# touch print

printall:
#
# pr -l66 $(FSRCS) $(CSRCS) $(OTHERS) | $P
# enscript $(FSRCS) $(CSRCS) $(OTHERS)
# touch print

printexport:
    pr -t $(FSRCS) $(CSRCS) $(OTHERS)

```

11. Appendix E: Program *dz* Example and Source Code

This program reads listings produced from the USGS/NEIS on-line hypocenter reporting service called *QED*, and produces input appropriate for the *cutpaste* script. The *QED* service can be reached via the Internet by "tel-net neis.cr.usgs.gov", and logging in as user "QED". Select option "q", give your name and institution, select option "d" to specify a date window, and finally select option "g" ("go") to obtain the needed event listing. Use text-selection options on your workstation to copy the listing to a file that is then passed to *dz* (edit out all but date and event lines). An example of such an (edited) listing (accessed 23 December 1993) and the resulting output from *dz* follow:

Example of Input to *dz*

```
DEC 18
032212.9 38.824N 122.757W 5G 1.1 35 NORTHERN CALIFORNIA. ML 3.1
205356.0 63.361N 149.817W 100 4.3 0.8 41 CENTRAL ALASKA
211028.7 62.918N 150.663W 136? 0.8 15 CENTRAL ALASKA
224419.9 20.571S 173.881W 33N 5.6 5.7 1.1 82 TONGA ISLANDS

DEC 19
072144.1* 15.353N 94.277W 33N 4.1 0.7 15 NEAR COAST OF OAXACA, MEXICO
080314.1 63.348N 151.241W 10G 4.1 0.7 21 CENTRAL ALASKA
083855.7 37.636N 118.833W 5G 1.0 41 CALIFORNIA-NEVADA BORDER REGION.
103134.1? 7.29 N 126.69 E 33N 5.3 4.9 1.3 25 MINDANAO, PHILIPPINE ISLANDS
114532.1 25.213N 62.614E 33N 5.2 5.1 1.0 46 SOUTHWESTERN PAKISTAN

DEC 20
003659.8* 51.750N 174.986W 33N 4.3 1.2 12 ANDREANOF ISLANDS, ALEUTIAN IS.
030352.3 42.118N 122.028W 5G 0.7 12 OREGON. ML 2.9 (GS).
083406.8 61.882N 150.570W 60G 0.5 13 SOUTHERN ALASKA
135619.3 6.752S 131.314E 33N 6.0 5.8 1.2 82 TANIMBAR ISLANDS REG.,
155657.7 62.206N 150.473W 10G 0.9 38 CENTRAL ALASKA. ML 3.4 (AEIC).
173659.7 32.333N 115.404W 5G 0.6 14 CALIF.-BAJA CALIF. BORDER
```

Resulting Output from *dz*

```
YYYY/MM/DD hh:mm:ss.ss Lat,N Long,W h,km mb Ms delta azimuth F-E region

**SKIP**
1993/12/18 03:22:12.90 38.824 -122.757 5 0.0 0.0 1.732 320.024 NORTHERN CALIFORNIA
1993/12/18 20:53:56.00 63.361 -149.817 100 4.3 0.0 31.038 335.495 CENTRAL ALASKA
1993/12/18 21:10:28.70 62.918 -150.663 136 0.0 0.0 31.027 334.360 CENTRAL ALASKA

1993/12/18 22:44:19.90 -20.571 -173.881 33 5.6 5.7 76.251 229.926 TONGA ISLANDS
(Delta/back-azm: sft2=075.730/229.638, sft4=076.075/229.826, sft5=076.251/229.926)
```

delta	#	code	time(s)	(min s)	dT/dD	hh:mm:ss.ss
76.25	1	P	705.26	11 45.26	5.6799	22:56:05.16
	2	pP	715.61	11 55.61	5.6945	22:56:15.51
	3	PcP	716.88	11 56.88	4.3197	22:56:16.78
	4	sP	719.70	11 59.70	5.6913	22:56:19.60
	5	PP	876.40	14 36.40	8.4227	22:58:56.30

6	PKiKP	1050.12	17	30.12	1.4975	23:01:50.02
7	pPKiKP	1060.98	17	40.98	1.4968	23:02:00.88
8	sPKiKP	1064.95	17	44.95	1.4969	23:02:04.85
9	SKiKP	1261.14	21	1.14	1.5536	23:05:21.04
10	S	1287.91	21	27.91	10.9685	23:05:47.81
11	pS	1301.09	21	41.09	11.0094	23:06:00.99
12	sS	1305.57	21	45.57	10.9954	23:06:05.47
13	SKS _{ac}	1312.90	21	52.90	7.2666	23:06:12.80
14	SKKS _{ac}	1313.90	21	53.90	7.5423	23:06:13.80
15	ScS	1317.63	21	57.63	8.0658	23:06:17.53
16	SP _n	1323.76	22	3.76	12.8483	23:06:23.66
17	pSKS _{ac}	1327.06	22	7.06	7.2797	23:06:26.96
18	P _n S	1328.49	22	8.49	12.8330	23:06:28.39
19	sSKS _{ac}	1331.23	22	11.23	7.2759	23:06:31.13
20	SS	1582.91	26	22.91	15.1157	23:10:42.81
21	PKKP _{df}	1847.02	30	47.02	-1.6060	23:15:06.92
22	PKKP _{bc}	1859.03	30	59.03	-2.1875	23:15:18.93
23	SKKP _{df}	2058.25	34	18.25	-1.5475	23:18:38.15
24	PKKS _{df}	2062.23	34	22.23	-1.5473	23:18:42.13
25	SKKS _{df}	2273.22	37	53.22	-1.4894	23:22:13.12
26	P'P'd _f	2342.58	39	2.58	-1.8160	23:23:22.48
27	S'S'd _f	3201.94	53	21.94	-1.6197	23:37:41.84
28	S'S'ac	3213.43	53	33.43	-2.2053	23:37:53.33

(est from JB) LR 2147.22 35 47.22 28.1600 23:20:07.12

cutpaste 1 2 3 dddd jjjj sft5 93 352 22 55 05 2201 "T, @TONGA ISLANDS m5.6/5.7 d76 h33"

****SKIP****

1993/12/19 07:21:44.10	15.353	-94.277	33	4.1	0.0	32.595	125.501	NEAR COAST OF OAXACA, MEXICO
1993/12/19 08:03:14.10	63.348	-151.241	10	4.1	0.0	31.506	334.656	CENTRAL ALASKA
1993/12/19 08:38:55.70	37.636	-118.833	5	0.0	0.0	1.982	85.452	CALIFORNIA-NEVADA BORDER REGIO

1993/12/19 10:31:34.10 7.290 126.690 33 5.3 4.9 102.549 289.555 MINDANAO, PHILIPPINE ISLANDS
(Delta/back-azm: sft2=102.303/289.160, sft4=102.464/289.420, sft5=102.549/289.555)

delta	#	code	time(s)	(min s)	dT/dD	hh:mm:ss.ss
102.55	1	Pdiff	832.80	13 52.80	4.4389	10:45:26.90
	2	pPdiff	843.37	14 3.37	4.4389	10:45:37.47
	3	sPdiff	847.41	14 7.41	4.4389	10:45:41.51
	4	PP	1086.07	18 6.07	7.5051	10:49:40.17
	5	PKiKP	1094.00	18 14.00	1.8188	10:49:48.10
	6	pPKiKP	1104.85	18 24.85	1.8182	10:49:58.95
	7	sPKiKP	1108.83	18 28.83	1.8183	10:50:02.93
	8	SKiKP	1306.39	21 46.39	1.8639	10:53:20.49
	9	SKS _{ac}	1469.75	24 29.75	4.7029	10:56:03.85
	10	pSKS _{ac}	1484.32	24 44.32	4.7131	10:56:18.42
	11	sSKS _{ac}	1488.37	24 48.37	4.7103	10:56:22.47
	12	SKKS _{ac}	1507.57	25 7.57	7.1031	10:56:41.67
	13	Sdiff	1534.65	25 34.65	8.3233	10:57:08.75
	14	pSdiff	1548.59	25 48.59	8.3233	10:57:22.69
	15	sSdiff	1552.83	25 52.83	8.3233	10:57:26.93
	16	SP	1629.76	27 9.76	10.5677	10:58:43.86
	17	PS	1634.19	27 14.19	10.5609	10:58:48.29
	18	PKKP _{bc}	1791.62	29 51.62	-2.9638	11:01:25.72

19	PKKPdf	1800.98	30	0.98	-1.8550	11:01:35.08
20	SS	1964.01	32	44.01	13.8199	11:04:18.11
21	SKKPbc	2010.49	33	30.49	-2.7106	11:05:04.59
22	SKKPdf	2013.36	33	33.36	-1.8275	11:05:07.46
23	PKKSbc	2014.49	33	34.49	-2.7101	11:05:08.59
24	PKKSdf	2017.35	33	37.35	-1.8274	11:05:11.45
25	SKKSdf	2229.59	37	9.59	-1.7946	11:08:43.69
26	SKKSac	2231.59	37	11.59	-2.5204	11:08:45.69
27	P'P'df	2293.12	38	13.12	-1.9200	11:09:47.22
28	S'S'ac	3146.04	52	26.04	-2.9314	11:24:00.14
29	S'S'df	3155.66	52	35.66	-1.8597	11:24:09.76

(est from JB) LR 2887.79 48 7.79 28.1600 11:19:41.89

cutpaste 1 2 3 dddd jjjj sft5 93 353 10 44 27 2814 "T, @MINDANAO, PHILIPPINE ISLANDS m5.3/4.9 d1

****SKIP****

1993/12/19	11:45:32.10	25.213	62.614	33	5.2 5.1	117.173	355.990	SOUTHWESTERN PAKISTAN
1993/12/20	00:36:59.80	51.750	-174.986	33	4.3 0.0	39.720	308.708	ANDREANOF ISLANDS, ALEUTIAN IS
1993/12/20	03:03:52.30	42.118	-122.028	5	0.0 0.0	4.644	353.578	OREGON
1993/12/20	08:34:06.80	61.882	-150.570	60	0.0 0.0	30.321	332.870	SOUTHERN ALASKA

1993/12/20 13:56:19.30 -6.752 131.314 33 6.0 5.8 107.855 275.251 TANIMBAR ISLANDS REG
(Delta/back-azm: sft2=107.503/274.835, sft4=107.734/275.109, sft5=107.855/275.251)

delta	#	code	time(s)	(min s)	dT/dD	hh:mm:ss.ss
107.86	1	Pdiff	856.35	14 16.35	4.4389	14:10:35.65
	2	pPdiff	866.92	14 26.92	4.4389	14:10:46.22
	3	sPdiff	870.96	14 30.96	4.4389	14:10:50.26
	4	PKiKP	1103.79	18 23.79	1.8678	14:14:43.09
	5	pPKiKP	1114.63	18 34.63	1.8673	14:14:53.93
	6	sPKiKP	1118.61	18 38.61	1.8674	14:14:57.91
	7	PP	1125.38	18 45.38	7.3138	14:15:04.68
	8	SKiKP	1316.40	21 56.40	1.9091	14:18:15.70
	9	SKSac	1493.58	24 53.58	4.2885	14:21:12.88
	10	pSKSac	1508.21	25 8.21	4.2969	14:21:27.51
	11	sSKSac	1512.25	25 12.25	4.2946	14:21:31.55
	12	SKSdf	1533.19	25 33.19	1.9360	14:21:52.49
	13	SKKSac	1544.88	25 44.88	6.9572	14:22:04.18
	14	pSKSdf	1548.01	25 48.01	1.9360	14:22:07.31
	15	sSKSdf	1552.00	25 52.00	1.9360	14:22:11.30
	16	Sdiff	1578.82	26 18.82	8.3233	14:22:38.12
	17	pSdiff	1592.75	26 32.75	8.3233	14:22:52.05
	18	sSdiff	1596.99	26 36.99	8.3233	14:22:56.29
	19	SP	1684.89	28 4.89	10.2077	14:24:24.19
	20	SP	1686.52	28 6.52	9.2372	14:24:25.82
	21	PS	1689.29	28 9.29	10.2008	14:24:28.59
	22	PS	1690.83	28 10.83	9.2370	14:24:30.13
	23	PKKPbc	1775.42	29 35.42	-3.1450	14:25:54.72
	24	PKKPab	1790.37	29 50.37	-4.4368	14:26:09.67
	25	PKKPdf	1791.07	29 51.07	-1.8790	14:26:10.37
	26	SKKPbc	1995.73	33 15.73	-2.8543	14:29:35.03
	27	PKKSbc	1999.73	33 19.73	-2.8537	14:29:39.03
	28	SKKPdf	2003.58	33 23.58	-1.8574	14:29:42.88
	29	PKKSdf	2007.57	33 27.57	-1.8573	14:29:46.87

30	SS	2036.57	33	56.57	13.5324	14:30:15.87
31	SKKSac	2217.88	36	57.88	-2.6457	14:33:17.18
32	SKKSdf	2219.97	36	59.97	-1.8308	14:33:19.27
33	P'P'df	2282.91	38	2.91	-1.9271	14:34:22.21
34	S'S'ac	3130.07	52	10.07	-3.0873	14:48:29.37
35	S'S'df	3145.73	52	25.73	-1.8824	14:48:45.03

(est from JB) LR 3037.19 50 37.19 28.1600 14:46:56.49

cutpaste 1 2 3 dddd jjjj sft5 93 354 14 09 36 2940 "T, @TANIMBAR ISLANDS REG m6.0/5.8 d108 h33"

****SKIP****

1993/12/20 15:56:57.70	62.206	-150.473	10	0.0	0.0	30.496	333.417	CENTRAL ALASKA
1993/12/20 17:36:59.70	32.333	-115.404	5	0.0	0.0	7.092	135.061	CALIF

Source Code of *dz* Follows:

The source code is available *via* anonymous *ftp* from "andreas.wr.usgs.gov" in directory "~ftp/pub/outgoing/evans/event_extraction". This Open-File Report serves to release them to the public, subject to the limitations cited on the title page.

```

/*
 * DZ -- simple distance, azimuth, and travelttime calculator,
 * input from QED, calls ttimes with distance value, outputs
 * all relevant data. Positive North and East.
 *
 * 27 Sept 93, add output of "epick" "pick" files for each event.
 * Only outputs for channel code "v", i.e., for rotated traces.
 * Requires change in logic so all stations get computed delta/azm
 * and one pick structure is made for each. Also added output of
 * all backazimuths to facilitate correct rotations later.
 *
 * USAGE: dz QED_file_name > output_file
 *
 * BUGS:
 *
 * WARNING: epick "pick" files have times for SFT5 exactly, but
 * estimates values for other sites from ttimes-output slownesses.
 * Interpret with care.
 *
 * Must have the following links in place:
 *   ln -s /we/itch/evans/src/resids/iasp91/iasp91.hed .
 *   ln -s /we/itch/evans/src/resids/iasp91/iasp91.tbl .
 *
 * Works only on Sun3 "evanss"--fragile. Fixable, but need ttimes
 * available and need to change path names appropriately.
 *
 * Creates and destroys files "ttimes.in.junk" and "ttimes.out.junk".
 *
 * Remember to update DATA_YEAR, CENTURY, the definitions of
 * central_lat and central_lon, and the format of "cutpaste" output
 * (the station name) as needed. Not neat, but functional.
 *
 * Also, change (0, MIN_MAG_0), (MID_DELTA, MIN_MAG_1) and
 * (180, MIN_MAG_2) which define a bi-linear distance-magnitude
 * rule determining which epicenters from the input file are
 * processed and which are skipped. (The minimum magnitude so
 * determined is compared to the maximum of mb and Ms.)
 */

#include <stdio.h>
#include <math.h>
#include <string.h>
#include <local/local.h>
#include <local/stdtyp.h>
#include <local/mathconst.h>
#include <local/date_time.h>
#include "pick_struct.h"

#define MAX(a,b) (((a)>(b)) ? (a) : (b)) /* Larger value */

#define MAXBUF 40 /* decode.h buffer size */
#include <local/decode.h>

/* Maximum size of QED event-list */
#define MXQED 2000
#define FE_LEN 31 /* Flinn-Engdahl array length */
#define STR_LEN 256 /* Maximum scratch-string length */

#define DATA_YEAR 1993
#define CENTURY 1900
#define MIN_MAG_0 2.5 /* minimum mb at zero distance */
#define MID_DELTA 5. /* slope-bread distance (deg) */
#define MIN_MAG_1 4.9 /* minimum mb at MID_DELTA */
#define MIN_MAG_2 5.6 /* minimum mb at the antipode */

/* Coordinates of possible station(s) from
   which to calculate delta and azimuth */
#define SFT2NAM "sft2"
#define SFT2LAT 37.153042
#define SFT2LON -121.812232
#define SFT2ELEV 518

#define SFT4NAM "sft4"
#define SFT4LAT 37.38883
#define SFT4LON -121.49450
#define SFT4ELEV 660

#define SFT5NAM "sft5"
#define SFT5LAT 37.505053
#define SFT5LON -121.328514
#define SFT5ELEV 312

#define NSTA 3

typedef struct { /* Station information */
    char code[5]; /* Name */
    double lat; /* Location (+N, degrees) */
}

```

```

double lon;          /* (+E, degrees) */
double elev;         /* meters */
double delta;        /* Distance to event, degrees */
} STA;

/* Surface wave nominal velocity (s/degree) */
#define LR_VEL      28.16

typedef struct {      /* Known-event information */
    TIME    ot;        /* Origin time */
    double  lat;       /* Hypocenter (+N, degrees) */
    double  lon;       /* +E, degrees */
    float   depth;     /* kilometers */
    float   mag;        /* mb */
    float   ms;        /* Ms */
    char    fe[FE_LEN]; /* Flinn-Engdahl region name */
    double  delta;     /* Distance (degrees) */
    double  back;      /* Back azimuth (degrees) */
    int     hit;       /* Number of times triggered on */
} QEDMAP;

#define strneq !strcmp

void    ep_out();      /* Defined below */
int     build_qed();   /* Defined below */
void    ph_sav();      /* Defined below */
bool    read_ev();     /* Defined below */
char    *tm_str();     /* Defined below */
char    *tm_cut();     /* Defined below */

void    datetime();    /* -lq (YMDhms <- epochal) */
void    delaz();       /* -lq (geocen.c) */
FILE    *fopen();      /* Unix C library */
FILE    *efopen();     /* -lq (efopen.c) */
void    error();       /* -lq (error.c) */
bool    isleap();      /* -lq */
void    refpt();       /* -lq (geocen.c) */
void    report();      /* -lq (error.c) */
TIME    timvar();      /* -lq (YMDhms -> epochal) */

#define MAXPHASE      100 /* Maximum number of phases */
int     npcks;         /* Number of phases for event */
TIME    ph_t[MAXPHASE]; /* Phase travel time */
double  ph_s1[MAXPHASE]; /* Phase slowness, s/degree */
double  ph_s2[MAXPHASE]; /* Phase slowness derivative, s/d^2 ??? */
char    ph_n[MAXPHASE][7]; /* Phase name */

main(argc, argv)
    int    argc;
    char   **argv;
{
    int    yr, mo, day, hr, mn;
    double sec;

    int    nqed;        /* Number of events read */

    int    ii;          /* Dummy */
    int    jj;          /* Dummy */
    int    skipped_last = FALSE; /* Output flag */

    float  min_mag;     /* Minimum magnitude to pass */

    char    qed_name[40]; /* Name of "QED" file */
    FILE    *qed_file;
    QEDMAP  qedm[MXQED]; /* QED-event map */

    FILE    *tt_file;   /* ttimes I/O files */
    char    in_str[STR_LEN]; /* Input buffer */
    char    ttout[16];  /* Output buffer for time of day */

    char    sys_str[256]; /* ttimes command buffer */

    bool    first_ph = TRUE; /* Take note of first-arriving phase */
    TIME    phltime;      /* Its travelttime */

    double  central_lat = SFT5LAT; /* Latitude of station */
    double  central_lon = SFT5LON; /* Longitude of station */

    STA     stations[10]; /* Station list */

    /* Build station list the dumb way */

    (void)strcpy(stations[0].code, SFT2NAM, 5);
    stations[0].lat = SFT2LAT;
    stations[0].lon = SFT2LON;
    stations[0].elev = SFT2ELEV;

    (void)strcpy(stations[1].code, SFT4NAM, 5);

```



```

stations[1].lat = SFT4LAT;
stations[1].lon = SFT4LON;
stations[1].elev = SFT4ELEV;

(void)strncpy(stations[2].code, SFT5NAM, 5);
stations[2].lat = SFT5LAT;
stations[2].lon = SFT5LON;
stations[2].elev = SFT5ELEV;

/* Check usage */
if (argc < 2)
    error ("Usage: %s QED_file_name > output_file", argv[0]);

qed_name[0] = '\0';
--argc, argv++;
(void)strcpy(qed_name, *argv); /* "QED" file name */

/* Use middle station as reference point for build_qed */
refpt((central_lat * RAD), (central_lon * RAD));

/* Read in QED list */
if((qed_file = fopen(qed_name, "r")) == NULL)
    error("Error opening \"QED\" file \"%s\".", qed_name);
if ((nqed = build_qed(qedm, qed_file)) > 0) {
    (void)printf("YYYY/MM/DD hh:mm:ss.ss Lat,N Long,W");
    (void)printf(" h,km mb Ms delta azimuth F-E region\n");

    /* Run ttimes for each event, extracting the useful fraction */
    sys_str[0] = '\0';
    (void)strcat(sys_str,
        " /we/geotele/evans/usr/ttimes");
    (void)strcat(sys_str, " < ttimes.in.junk");
    (void)strcat(sys_str, " > ttimes.out.junk");

    for (ii = 0 ; ii < nqed ; ii++) {

        /* Bother only with qualifying magnitudes */

        if (qedm[ii].delta < MID_DELTA) {
            min_mag = MIN_MAG_0 +
                (MIN_MAG_1 - MIN_MAG_0) *
                qedm[ii].delta / MID_DELTA;
        }
        else {
            min_mag = MIN_MAG_1 +
                (MIN_MAG_2 - MIN_MAG_1) *
                (qedm[ii].delta - MID_DELTA) /
                (180.0 - MID_DELTA);
        }

        if (MAX(qedm[ii].mag, qedm[ii].ms) >= min_mag) {

            if (skipped_last)
                (void)printf("\n");
            skipped_last = FALSE;

            /* Create input to ttimes, and run ttimes */

            tt_file = fopen("ttimes.in.junk", "w");
            (void)fprintf(tt_file,
                "all\n%.3f\n%.3lf\n-1\n-1\n",
                qedm[ii].depth, qedm[ii].delta);
            (void)fclose(tt_file);

            if ((system(sys_str) & 0377) == 127)
                error("ttimes call failed!");

            /* Output relevent event information */

            datetime(qedm[ii].ot,
                &yr, &mo, &day, &hr, &mn, &sec,
                GREGORIAN);
            (void)printf("\n%4i/%02i/%02i %02i:%02i:%05.2f ",
                yr, mo, day, hr, mn, sec);
            (void)printf("%7.3lf %8.3lf %3.0f ",
                qedm[ii].lat, qedm[ii].lon,
                qedm[ii].depth);
            (void)printf("%3.1f %3.1f %7.3lf %7.3lf ",
                qedm[ii].mag, qedm[ii].ms,
                qedm[ii].delta,
                qedm[ii].back);
            (void)printf("%s\n",
                qedm[ii].fe);

            /* Calculate and output all deltas
               and backazimuths */
            (void)printf("(Delta/back-azm: ");

```

```

for (jj = 0 ; jj < NSTA ; jj++) {
    double delta, az0, az1;

    refpt((stations[jj].lat * RAD),
          (stations[jj].lon * RAD));
    delaz(qedm[ii].lat * RAD,
          qedm[ii].lon * RAD,
          &delta, &az0, &az1);
    az0 *= DEG;
    delta *= DEG;
    stations[jj].delta = delta;

    (void)printf("%s=%07.3lf/%07.3lf",
                 stations[jj].code,
                 delta, az0);
    if (jj == NSTA - 1)
        (void)printf("\n");
    else
        (void)printf(", ");
}
(void)printf("\n");

/* copy through the useful parts of ttimes
   output */
npcks = 0;      /* Start epick phase list */
tt_file = fopen("ttimes.out.junk", "r");
while(fgets(in_str, sizeof(in_str), tt_file) !=
      NULL)
    if (strneq("Source depth (km):",
              in_str, 18))
        break;
while(fgets(in_str, sizeof(in_str), tt_file) !=
      NULL) {
    if (strneq(
        "Enter delta: Source depth (km):",
        in_str, 32))
        break;
    else {
        if (strneq("Enter delta: ",
                  in_str, 14)) {
            in_str[53 + 14] = '\0';
            (void)printf(
                "%s hh:mm:ss.ss\n",
                in_str + 14);
        }
        else {
            TIME tt = 0;
            tt = (double)DECODE(
                (in_str + 22), 9, atof);
            if (tt > 0.0) {
                if (first_ph) {
                    phltime = tt;
                    first_ph = FALSE;
                }
                in_str[53] = '\0';
                (void)printf("%s %s\n",
                             in_str, tm_str(
                                 qedm[ii].ot, tt,
                                 ttout));

                /* Save phase for
                   epick "pick" file */
                ph_sav(in_str, tt);
            }
            else
                (void)printf("%s", in_str);
        }
    }
}

/* Protect DECODE from leftovers */
for (jj = 0 ; jj < STR_LEN ; jj++)
    in_str[jj] = ' ';
}

/* Estimate surface waves */
sec = qedm[ii].delta * LR_VEL;
mn = sec / 60;
(void)printf("(est from JB) LR");
(void)printf(" %10.2f%4i%7.2f%11.4f %s\n\n",
             sec, mn, (sec - (mn * 60)), LR_VEL,
             tm_str(qedm[ii].ot, sec, ttout));

(void)fclose(tt_file);
first_ph = TRUE;
if ((system("/bin/rm ttimes.out.junk") &
    0377) == 127)

```

```

        report("rm call failed...");

/* Output for "cutpaste" script
   (from (first phase - PAD_FRONT)
   to (LR + PAD_REAR)) */

#define PAD_FRONT      60.0    /* s */
#define PAD_REAR      700.0    /* s */

(void)printf("cutpaste    1 2 3    ");
(void)printf("dddd jjjj sft5    ");
(void)printf("%s    %4d    \nT, @%s ",
             tm_cut((TIME) (qedm[ii].ot + phltime -
                          PAD_FRONT + 0.5), ttout),
             (int) (sec - phltime +
                    PAD_FRONT + PAD_REAR),
             qedm[ii].fe);
(void)printf("m%3.1f/%3.1f d%0.1f h%0.0f\n",
             qedm[ii].mag, qedm[ii].ms,
             qedm[ii].delta, qedm[ii].depth);

/* Output phase list to epick "pick" file */
ep_out(qedm[ii].ot, ttout,
       qedm[ii].delta, stations);
}

/* Report skipped events FYI */
else {
    datetime(qedm[ii].ot,
            &yr, &mo, &day, &hr, &mn, &sec,
            GREGORIAN);

    if (!skipped_last)
        (void)printf("\n**SKIP**\n");
    skipped_last = TRUE;

    (void)printf("%4i/%02i/%02i %02i:%02i:%05.2f ",
                yr, mo, day, hr, mn, sec);
    (void)printf("%7.3lf %8.3lf %3.0f ",
                qedm[ii].lat, qedm[ii].lon,
                qedm[ii].depth);
    (void)printf("%3.1f %3.1f %7.3lf %7.3lf ",
                qedm[ii].mag, qedm[ii].ms,
                qedm[ii].delta,
                qedm[ii].back);
    (void)printf("%s\n",
                qedm[ii].fe);
}

}

if ((system("/bin/rm ttimes.in.junk") & 0377) == 127)
    report("rm call failed...");
}

(void)fclose(qed_file);
}

/*
 * Given two TIME variables, add, and return time string "hh:mm:ss.ss".
 */

char *
tm_str(t1, t2, str)
    TIME    t1;
    TIME    t2;
    char    *str;
{
    int      yr, mo, day, hr, mn;
    double   sec;

    datetime((t1 + t2), &yr, &mo, &day, &hr, &mn, &sec, GREGORIAN);
    (void)sprintf(str, "%02i:%02i:%05.2f", hr, mn, sec);

    return (str);
}

/*
 * Given one TIME variable, returns time string "yy jjj hh mm ss".
 */

char *
tm_cut(tt, str)
    TIME    tt;
    char    *str;
{
    int      yr, mo, day, hr, mn, jday;
    double   sec;

```

```

    datetime(tt, &yr, &mo, &day, &hr, &mn, &sec, GREGORIAN);
    jday = yrdays(mo, day, isleap(yr, GREGORIAN));
    (void)sprintf(str, "%02i %03i %02i %02i %02i",
        yr - CENTURY, jday, hr, mn, (int)sec);

    return (str);
}

/*
 * Read an event line from QEDs defaulting numbers to zero and
 * strings to null if not present. Returns TRUE if it seems to
 * be an event line, otherwise FALSE.
 */

bool
read_ev(in_str, hr, mn, sec, lat, lath, lon, lonh, depth, mag, ms, fe)
    char in_str[];          /* Input buffer */
    int *hr, *mn;
    double *sec;
    double *lat, *lon;
    float *depth, *mag, *ms;
    char *lath, *lonh;
    char fe[];              /* Flinn-Engdahl buffer */
{
    int ii;                 /* Dummy */
    bool rd;

    rd = FALSE;

    *hr = 0;
    *mn = 0;
    *sec = 0.0;
    *lat = 0;
    *lath = '\0';
    *lon = 0;
    *lonh = '\0';
    *depth = 0;
    *mag = 0;
    *ms = 0;
    for (ii = 0; ii < FE_LEN; ii++)
        fe[ii] = '\0';

    *hr = DECODE(in_str, 2, atoi);
    *mn = DECODE((in_str + 2), 2, atoi);
    *sec = (double)DECODE((in_str + 4), 4, atof);
    *lat = (double)DECODE((in_str + 10), 6, atof);
    *lath = in_str[16];
    *lon = (double)DECODE((in_str + 18), 7, atof);
    *lonh = in_str[25];
    *depth = (float)DECODE((in_str + 27), 3, atof);
    *mag = (float)DECODE((in_str + 32), 3, atof);
    *ms = (float)DECODE((in_str + 36), 3, atof);

    (void)strcpy(fe, (in_str + 48));
    fe[FE_LEN - 1] = '\0';

    /* Strip trailing comments (avoiding 1-character abbreviations) */
    for (ii = 2; ii < FE_LEN; ii++) {
        if ((fe[ii] == '.' && fe[ii - 2] != ' ') || fe[ii] == '\n') {
            fe[ii] = '\0';
            break;
        }
    }

    /* Kludge, since there seems no other guarantee in QED listings */
    if ((*lath == 'N' || *lath == 'n' || *lath == 'S' || *lath == 's') &&
        (*lonh == 'E' || *lonh == 'e' || *lonh == 'W' || *lonh == 'w'))
        rd = TRUE;

    return (rd);
}

/*
 * Read "QED" file and build data base of events.
 */

int
build_qed(qedm, qed_file)
    QEDMAP qedm[];          /* Known-event map */
    FILE *qed_file;
{
    int nqed;
    char in_str[STR_LEN];    /* Input buffer */
    int yr = DATA_YEAR, mo = 0, day, hr, mn;
    double sec;
    double lat, lon;
    float depth, mag, ms;
    char lath, lonh;

```

```

char    fe[FE_LEN];          /* Flinn-Engdahl buffer          */
double  delta, az0, az1;
int      ii;                  /* Dummy                      */

/* refpt called from here in other versions.  In "dz", called from main. */

nqed = 0;
while (fgets(in_str, sizeof(in_str), qed_file) != NULL) {

    ii = intmo(in_str);        /* Always present above events */
    if (ii != 0) {
        mo = ii;
        if (sscanf(in_str + 3, "%d", &day) != 1)
            error("Could not read day in QEDs");
    }

    if (read_ev(in_str, &hr, &mn, &sec,
                &lat, &lath, &lon, &lonh, &depth, &mag, &ms, fe)) {

        /* Clear input string to avoid problems
         * with subsequent event and non-event lines */
        for (ii = 0 ; ii < STR_LEN ; ii++)
            in_str[ii] = '\0';

        if (lath != 'N')
            lat = -lat;

        if (lonh != 'E')
            lon = -lon;

        /* Calculate distance to event */
        delaz(lat * RAD, lon * RAD, &delta, &az0, &az1);
        delta *= DEG;
        az0 *= DEG;

        if (mo == 0)
            error("Programming error.");

        if (nqed >= MXQED)
            error("Too many QED entries.");

        /* Save event information */
        qedm[nqed].ot =
            timvar(yr, mo, day, hr, mn, sec, GREGORIAN);

        qedm[nqed].lat = lat;
        qedm[nqed].lon = lon;

        qedm[nqed].depth = depth;
        qedm[nqed].mag = mag;
        qedm[nqed].ms = ms;

        (void)strcpy(qedm[nqed].fe, fe);

        qedm[nqed].delta = delta;
        qedm[nqed].back = az0;
        qedm[nqed].hit = 0;

        nqed++;
    }
}
return (nqed);
}

/*
 * Add picks for one phase to the epick "pick" file
 */

void
ep_out(ot, str, delta, stations)
    TIME    ot;                /* Event origin time          */
    char    str[];             /* "cuspaste" string passed in */
    double  delta;             /* "center" (SFT5) delta, deg  */
    STA     stations[];        /* Station list                */
{
    FILE    *ep_file;          /* Output stream              */
    PMAIN    pck;              /* An epick "pick"            */
    int      iph, ist;
    double   ddelta;

    /* Open file (construct name from "cuspaste" string) */
    str[2] = str[3];
    str[3] = str[4];
    str[4] = str[5];
    str[5] = '.';
    str[6] = str[7];

```

```

str[7] = str[8];
str[8] = str[10];
str[9] = str[11];
str[10] = str[13];
str[11] = str[14];
str[12] = '.';
str[13] = 'e';
str[14] = 'p';
str[15] = '\0';

ep_file = fopen(str, "w");

/* Create and write out epick "pick" structures */
for (iph = 0 ; iph < npcks ; iph++) {
    for (ist = 0 ; ist < NSTA ; ist++) {

        (void)strcpy(pck.stcode, stations[ist].code);
        (void)strcpy(pck.chan, "v");

        (void)strcpy(pck.phcode, ph_n[iph]);
        ddelta = stations[ist].delta - delta;
        pck.t = ot + ph_t[iph] + ddelta * ph_s1[iph];
        (ph_s1[iph] + ph_s2[iph] * ddelta / 2); */
        pck.setime = 999.999;

        pck.onset = ' ';
        pck.pol = ' ';
        pck.qual = '1';          /* Meaning "iaspi91"      */

        pck.amp = 0.0;
        pck.freq = 0.0;

        if (fwrite((char *)&pck, sizeof(pck), 1, ep_file) != 1)
            report("Can't write picks");
    }
}

/* Clean up */
(void)fclose(ep_file);
}

/*
 * Save phase traveltimes and names for epick "pick" file.
 */

void
ph_sav(in_str, tt)
char    *in_str;
TIME    tt;
{
    int    jj;

    if (npcks == MAXPHASE)
        report("Too many phase names--ignored\n%s", in_str);

    else {
        ph_t[npcks] = tt;
        ph_s1[npcks] = (double)DECODE((in_str + 46), 10, atof);
        ph_s2[npcks] = (double)DECODE((in_str + 65), 10, atof);

        (void)strncpy(ph_n[npcks], in_str + 14, 6);
        ph_n[npcks][6] = '\0';
        for (jj = 5 ; jj >= 0 ; jj--)
            if (ph_n[npcks][jj] == ' ')
                ph_n[npcks][jj] = '\0';

        npcks++;
    }
}

```

```
/* Character month -> integer month */

#include <string.h>

#define strneq !strcmp
#define streq !strcmp

int
intmo(chmo)
    char    *chmo;
{
    if (strneq(chmo, "Jan", 3) || strneq(chmo, "JAN", 3))
        return(1);
    else if (strneq(chmo, "Feb", 3) || strneq(chmo, "FEB", 3))
        return(2);
    else if (strneq(chmo, "Mar", 3) || strneq(chmo, "MAR", 3))
        return(3);
    else if (strneq(chmo, "Apr", 3) || strneq(chmo, "APR", 3))
        return(4);
    else if (strneq(chmo, "May", 3) || strneq(chmo, "MAY", 3))
        return(5);
    else if (strneq(chmo, "Jun", 3) || strneq(chmo, "JUN", 3))
        return(6);
    else if (strneq(chmo, "Jul", 3) || strneq(chmo, "JUL", 3))
        return(7);
    else if (strneq(chmo, "Aug", 3) || strneq(chmo, "AUG", 3))
        return(8);
    else if (strneq(chmo, "Sep", 3) || strneq(chmo, "SEP", 3))
        return(9);
    else if (strneq(chmo, "Oct", 3) || strneq(chmo, "OCT", 3))
        return(10);
    else if (strneq(chmo, "Nov", 3) || strneq(chmo, "NOV", 3))
        return(11);
    else if (strneq(chmo, "Dec", 3) || strneq(chmo, "DEC", 3))
        return(12);
    else
        return(0);
}
```

```
/* STCSIZE should be abandoned in favor of using CODESIZE from ahead.h */

/*
 * STCSIZE and CHANSIZE define the maximum number of PRINTABLE characters
 * in the respective strings. These strings, therefore, MAY NOT BE NULL
 * TERMINATED. Use 'strcmp(..., STCSIZE)' and '... "%s", STCSIZE,
 * stcode ...' and such.
 */

#define STCSIZE      4          /* Length of station code      */
#define CHANSIZE     6          /* Length of channel code     */
#define PHCSIZE      8          /* Length of phase code       */
#define NULL_TIME    1.0e10

/* Indispensable part of a PICK */
typedef struct {
    char    stcode[STCSIZE]; /* Station code                */
    char    chan[CHANSIZE];  /* Channel code                */
    char    phcode[PHCSIZE]; /* Phase code                  */
    TIME    t;              /* Arrival time               */
    float    setime;         /* Standard error of arrival time */
    char    onset;          /* 'e' or 'i'                 */
    char    pol;            /* Polarity: 'u' or 'd'       */
    char    qual;           /* Pick quality                */
    float    amp;           /* Amplitude                  */
                                /* Note: convention (e.g. peak-to-peak */
                                /* vs. zero-to-peak) is not enforced by */
                                /* program (but is important).        */
    float    freq;          /* Frequency                   */
} PMAIN;

/* Time from clock or radio time channel */
typedef struct {
    TIME    t;
} CODE_T;

/* Related things measured from a trace */
typedef struct {
    PMAIN    pm;            /* Required                    */
    CODE_T    c_t;         /* Optional                    */
} PICK;
```



```

#
#-----*-----*-----*-----*-----*-----*-----*-----*-----*
#
# Generic Makefile for programs.
# Works for any mix of sun3, sun4, Fortran, and C (I hope!).
#
# "make install" fails the first time unless the executable has
# already been copied to $(BINDIR).
#
# The following is required in the user's .cshrc:
#
# setenv ARCH `bin/arch`
# if( $ARCH == "sun3") then
#     setenv FLOAT -f68881
# else
#     setenv FLOAT ""
# endif
# setenv DIR_USR /usr/local
# setenv DIR_SRC /usr/src/local

FC      = f77
CC      = cc
FFLAGS  = -u $(FLOAT) -D$(ARCH)
CFLAGS  = $(FLOAT) -D$(ARCH) -g
BINDIR  = $(DIR_USR)

PROG    = dz
SRCARC  = $(PROG).src.a

FSRCS   =
CSRCS   = dz.c \
          intmo.c
OBJECTS = $(FSRCS:%.f=$(ARCH)/%.o) $(CSRCS:%.c=$(ARCH)/%.o)

LIBS    = -lq -lm
LLIBS   = $(LIBS)

INCLS   =
OTHERS  = Makefile $(INCLS)
OTHERBIN=

# P      = enscript -r2 -p-

$(ARCH)/$(PROG): $(OBJECTS)
    $(CC) $(CFLAGS) $(OBJECTS) $(LIBS) -o $@

$(OBJECTS): $(INCLS)

$(ARCH)/%.o: %.c
    $(CC) $(CFLAGS) -c $(@F:.o=.c) -o $@

$(ARCH)/%.o: %.f
    $(FC) $(FFLAGS) -c $(@F:.o=.f) -o $@

install: $(ARCH)/install $(ARCH)/$(PROG)
#
# /bin/rm install
#
# ln -s $(ARCH)/install install
# /bin/mv $(BINDIR)/$(PROG) $(BINDIR)/$(PROG).old
# cp $(ARCH)/$(PROG) $(BINDIR)
# chmod 755 $(BINDIR)/$(PROG)
# touch install
# touch $(ARCH)/install

# CAUTION: do not run "make clean" on more than one machine at a time;
# the archive commands ("ar") might conflict.
clean:
    /bin/rm $(ARCH)/$(PROG) $(OBJECTS) $(BINDIR)/$(PROG).old
    ar ruv $(SRCARC) $(FSRCS) $(CSRCS) $(OTHERS) $(OTHERBIN)

lint: $(FSRCS) $(CSRCS) $(INCLS)
    /usr/bin/lint -xh -u $(FSRCS) $(CSRCS) $(LLIBS) > lint

print: $(FSRCS) $(CSRCS) $(OTHERS)
#
# pr -l66 $? | $P
# enscript $?
# touch print

printall:
#
# pr -l66 $(FSRCS) $(CSRCS) $(OTHERS) | $P
# enscript $(FSRCS) $(CSRCS) $(OTHERS)
# touch print

printexport:
    pr -t $(FSRCS) $(CSRCS) $(OTHERS)

```

12. Appendix F: Manual Page and Source Code for *ahwwvb*

The algorithm used in the *ahwwvb* time-code reader is an amalgam of robust processes coded in C and FORTRAN—several running-median filters, an edge-finding algorithm, and an L1 fit to the second ticks. The latter, and the BCD deciphering code that follows, are taken from Carl Johnson's earlier efforts, while the signal preprocessors are creations of the first author of this Report. The signal processors were designed from general principals (Evans, 1992) and optimized by tests on real data, mostly of 100 sps. It also works well at 250 sps, but is only lightly tested at other sample rates. It is the most reliable automated WWVB reader known to us.

Both the source code and the manual page are available *via* anonymous *ftp* from "andreas.wr.usgs.gov" in directory "~ftp/pub/outgoing/evans/wwvb". This Open-File Report serves to release them to the public, subject to the limitations cited on the title page.

Source code for the running-median routine (from the first author's *librmf*), and relevant routines from Bruce Julian's *libq* are listed in Appendix H. The I/O library *libah* is available from Lamont-Doherty ("ftp lamont.ldeo.columbia.edu" in compressed file "/pub/ah.tar.Z").

SYNOPSIS

```
ahwwvb -m min_rate ah_file [ -vb code channel ] [ -v ] [ -la | -l list_file ] [ -y year ] [ -r | -s ] [ -d
delay ] [ -nc ]
```

DESCRIPTION

Reads a WWVB radio time-code trace in (*xdr*) AH format, infers sampling rate and the NTIS/UTC time of the first sample. Accordingly, changes the WWVB trace starting time and digitizing interval, as well as those of any other traces indicated. Affects all traces listed in *list_file* (in the same format as *epick*'s *list_file*), or else changes *all* traces in *ah_file* (*-la*, the default). If *-l* is used, copies through, unchanged, all other traces encountered. Performs simple relatedness checks on traces to be changed, *but only with the -la option*. Simply asks whether they all have the same length, original sampling interval, and original starting time.

ahwwvb is intended primarily for correcting start time and digitizing interval of data from "five-day recorders" and other instruments that record WWVB radio time code in parallel with seismic data, but which may give inaccurate start time or digitizing interval due to inaccurate internal clocks or analog-instrumentation foibles. Hence, *ahwwvb* assumes that all traces to be changed are *exactly the same* time interval and that *all* traces from that parallel record are targeted for the change. Since AH has no formal way of associating a group of traces, this game is somewhat dangerous. *You may get bizarre results if you change too few or too many traces*. Logs the changes it makes in the AH header.

ahwwvb does significant signal massaging to extract robustly the most accurate time. Uses a combination of several running-median filters and an edge-finding algorithm to interpret the WWVB trace into a binary time series. This binary series is decoded by Carl Johnson's routine, the operation of which is partly mysterious. The translation to binary includes a number of empirically set parameters for determining what a sharp corner really is and how extreme the signal massaging may be. These may need adjustment in some cases (see file *ahwwvb.h*). Also biases the result to account for a filtered *versus* unfiltered WWVB trace. Uses an L1 fit (in C. Johnson's code) to calculate fractional seconds of the start time, and the sampling interval.

It is often the case that part of the starting time can be inferred, but not all of it. Fractional seconds and sample interval are found most often, then units and tens of seconds, and lastly (as one group) the Julian day, hour, and minute. Any one of these not found by interpreting WWVB is simply set to respective values taken from the original AH header, with appropriate warnings sent to *stdout*. WWVB does not tell one the year, so that is taken from the original AH header or overridden by the *-y* option.

One of the sad discoveries made in writing *ahwwvb* is that current five-day playback procedures are very poor. Many files have varying digitizing rates, witnessed by varying intervals between WWVB "ticks", and some files have drastic fluctuations in rate. Based on my small test sample, these all appear to be increases in sample interval (WWVB appearing compressed) lasting on the order of 10 s and often occurring several times per minute. Presumably, the tape-speed compensation is not working properly and the recorder slows episodically (this mechanical dragging is known to be common near the terminal end of tapes, due to imperfections in the recorder design and, to a lesser degree, maintenance). Even this veteran observer failed to notice many of these fluctuations until looking at the tick intervals rather closely, so I have added some diagnostics to *ahwwvb*, to wit a statistical evaluation of tick intervals. This problem often is accompanied by a computed digitizing rate different from that requested. The work-around (other than *fixing* the tape-speed compensation!) is to check the time of a WWVB tick mark near each your picks, using *epick* to repick WWVB where necessary. Data digitized in real time from the networks are, as one expects, rock constant in sampling rate.

ahwwvb reports the value of variable *ires* if *-v* flag is set. *ires* means:

- 0 = total failure to decode WWVB,
- 1 = sample interval and fractional "sec" decoded successfully,
- 2 = units of "sec" decoded too,
- 3 = all of "sec" decoded,
- (4 is not used)
- 5 = WWVB was fully decoded (Julian day, hour, and minute too)

ARGUMENTS

-m *min_rate* is the lowest sampling rate (samples per second) that this trace could possibly be. This figure is used to design a long running-median filter to clip off noise spikes shorter than 60% (-r) or 80% (-s) of the shortest valid signal pulse (0.2 s, in the case of WWVB). You should give the highest reasonable number but not overestimate, or valid data could be clipped and unreadable as well. You may disable filtering by giving any *min_rate* < 15 sps. I know of no good reason, aside from bad judgement in sampling rates, to override the filtering. If filtering is done, *ahwwvb* prefilters with a series of shorter RMF windows to improve edge fidelity. In any case, *ahwwvb* tells you what it did or did not do.

ah_file The (*xdr*) AH file containing the WWVB trace and possibly other traces.

OPTIONS

- vb** *code* and *channel* identify the WWVB trace in *ah_file*. Defaults are "WWVB" and either "TIME", "time", or "T", respectively.
- v** Cause verbose output that provides a trace of the processing done. The default is to print out only the summary message and any error messages. (Users with scripts designed to digest the output of early versions of *ahwwvb* should invoke this option.)
- la** (Default) All traces in *ah_file* are changed.
- l** *list_file* must contain one or more AH file names and trace sequence-numbers; only these traces are changed. The WWVB trace must be listed here (a check is made). *list_file* is in the same format as for *epick*.
- y** *year* is the year of the first sample in the file. It overrides the year found in the original AH header.
- d** *delay* is the total lag of the recorded WWVB signal behind UTC. It is typically 18±5 ms radio lag + the flight time. *delay* is given in ms (default 0). The start time is set earlier than the recorded WWVB signal by this amount.
- r**
- s** Is the WWVB trace "rounded" (-r, the default) or "sharp" (-s)? That is, does the trace appear to have passed through a low-pass filter, as is the case for the five-day recorder (the discriminators apply low pass filters to all traces). WWVB digitized by CUSP, for example, is more nearly raw, with apparently sharp transitions. However, WWVB received on a TrueTime® radio and digitized at 1000 sps on a PASSCAL recorder shows significantly rounded transitions, with about 5 ms from the start of the transition to the steepest slope.

The issue is where to infer that the original WWVB signal transitioned. For -r, it is assumed to be one sample interval before the steepest first difference; for -s, it is taken to be at the same sample interval as the steepest first difference. This distinction makes only a one-sample-interval difference. In all cases, the true transition time is assumed to be midway in the identified sample interval. It is preferable to use -s with -d to compensate explicitly when you know the time lag caused by filtering ("-d (*time_of_steepest_slope* - *time_of_actual_transition* + *any_other_lags*)").

- nc** By default, *ahwwvb* updates the traces themselves, as described. This action can be suppressed by the -nc ("no change") flag, in which case, the result is still reported but nothing is done with it. The -nc option is useful for tests and to those who record clock corrections elsewhere (and apply them to results rather than to the original data).

SEE ALSO

epick—X11 *ah* picking routine derived and extended from Lamont's *sunpick*. *tcpick* invokes *epick* in a mode allowing one to manually read (and optionally correct to) a recorded time code like WWVB.

ahclk—called by *ahwwvb* to make the actual changes to AH headers.

Other *ah* filters.

BUGS

list_file must be sorted with all the traces for any particular file together and in ascending numerical order within each file. *mkahlist* does that for you. Someday, *ahclk* could be made smarter about this, but it hasn't been yet.

ahwwvb should be smart enough to add the WWVB trace ID to any *list_file*, but it isn't.

Uses inflagrevous FORTRAN spaghetti code of C. Johnson, so it is largely a mystery how the interpreter works (or doesn't). Watch out for errors (particularly 1-, 10-, and 60-s errors?).

ahwwvb cannot be used as a UNIX filter.

Either *ahwwvb* or *ahclk* should check relatedness for the *-l* option too.

Would be more accurate with a rawer WWVB. TrueTime® receivers try make the trace pretty by thresholding the signal to a running mean of the signal, thereby damaging the information contained (adding non-constant small delays to the transitions).

The *-r* transition bias is utterly empirical, set from years of experience almost exclusively with 100 sps five-day recorder data. It should at least be user-controllable or, better yet, knowledgeable of the filtering.

Should offer a way of holding the digitizing rate fixed, for the case of digital recorders.

ahwwvb is new and tested on only ten examples—keep your eyes open!

The filtering and L1 fit are noticeably slow at high sample rates. 2-minute 1000-sps records decode in about 7 minutes on a Sun 3/60.

ahwwvb has been used at 100, 200, and 1000 sps. It probably will work out to 2000 sps, but there are some poorly understood magic numbers in C. Johnson's code, and these may cause it to fail at some high rate. Of course, it will certainly fail below 5 sps because WWVB has meaningful pulses as short as 0.2 s.

AUTHOR

John R. Evans, USGS, Menlo Park, CA. Original written March 16, 1992. *-nc* flag added June 17, 1992. Made to work for high sampling rates (at least to 1000 sps) June 23, 1992.

```

/*
 * AH filter AHWWVB (AH WWVB reader):
 *   Read WWVB and change AH headers accordingly.  See "ahwwvb.man".
 *
 * dbxtool -I . -I /we/itch/evans/src/lib/rmf /we/itch/evans/usr.sun3/ahwwvb &
 * stop in median
 * when at "/we/itch/evans/src/lib/rmf/cmedian.c":143 { print iout,arrout[iout] ; }
 * when ( ( iout % 100 ) == 0 ) { print iout,arrout[iout] ; }
 * stop if { iout == 15000 }
 * run -m 990 06.29.0192.4.ah -vb 0192 4 -nc
 */

#include <stdio.h>
#include <math.h>
#include <rpc/rpc.h>
#include <local/ahhead.h>
#include <local/stdtyp.h>
#include <local/date_time.h>

#define MIN(a,b)      ((a) < (b)) ? (a) : (b)      /* Smaller value */
#define MAX(a,b)      ((a) > (b)) ? (a) : (b)      /* Larger value */

#include "ahwwvb.h"

#define strneq !strncmp
#define streq !strcmp

void      error();          /* -lq (error.c)          */
void      report();         /* -lq (error.c)         */
FILE      *efopen();        /* -lq (efopen.c)        */
char      *emalloc();       /* -lq                   */
char      *erealloc();      /* -lq                   */
void      mnday();          /* -lq                   */
bool      isleap();         /* -lq                   */
TIME      timvar();         /* -lq                   */
void      datetime();       /* -lq                   */

/*      xdr_getthead()      -lah          */
/*      mkdataspace()      -lah          */
/*      xdr_getdata()      -lah          */

int      read_wwvb();       /* Defined below         */
TIME      ah2epoch();       /* Defined below         */
int      seq_ok();         /* Defined below         */
void      ch_files();       /* Defined below         */
void      decode_WWVB();    /* Defined below         */
void      tick_size();     /* Defined below         */
static int icomp();        /* Defined below         */
int      is_wwvb();        /* Defined below         */
void      to_bin();        /* Defined below         */
static int fcomp();        /* Defined below         */
bool      is_max();        /* Defined below         */
bool      is_min();        /* Defined below         */
void      wwvb_();         /* wwvb.f                */

char      *progname;

main(argc, argv)
int      argc;
char      **argv;
{
    FILE      *ah_file;
    char      ah_name[NM_LEN]; /* Name of AH file with WWVB trace */
    XDR      xdr_in;

    int      n_traces;        /* Total traces in ah_file */
    int      itrace;

    int      wwvb_seq;        /* Position of WWVB trace */
    ahhed      wwvb_head;     /* Space for WWVB AH header */
    float      *wwvb_trace;   /* Pointer to WWVB AH trace */

    FILE      *list_file;     /* listfile a la "epick" */
    char      list_name[NM_LEN];
    int      all_traces;      /* Over-ride listfile and change
                                all traces */
    int      change_ah;       /* Actually change ah traces? */
    int      yr, mo, day, hr, mn;
    double    sec;

    bool      verbose = FALSE; /* Write out gory details? */

    FILE      *temp_file;     /* listfile a la "epick" */
    char      temp_name[NM_LEN]; /* List file name actually
                                passed to AHCLK */

    float      min_rate;      /* Minimum digitizing rate possible */

```

```

int          RMF_win;          /* Resulting maximal RMF window length */

char         wwvb_code[CODESIZE]; /* WWVB trace code          */
char         wwvb_chan[CHANSIZE]; /* WWVB trace channel      */

int          year = 0;         /* Year of trace; overrides ahheader */

TIME         new_st;           /* New starting time        */
double       new_int;          /* New sampling interval    */
TIME         old_st;           /* Original starting time   */
double       old_int;          /* Original sampling interval */

char         sys_str[STR_LEN]; /* Scratch string           */
int          sysRet;

bool         rounded;          /* Is the trace "rounded" (filtered)? */
TIME         lagg;             /* Set clock back by this ammount */

/* Initialize */
new_st = NULL;                 /* No-change flag          */
new_int = -1.;                 /* No-change flag          */

min_rate = -1.0;               /* Flag no-value-given    */
RMF_win = -1;                  /* Flag no-value-given    */
ah_name[0] = '\0';             /* Flag no-value-given    */

all_traces = TRUE;             /* Default: change all traces */
change_ah = TRUE;

list_name[0] = '\0';
list_file = (FILE *) NULL;

(void)strcpy(wwvb_code, "WWVB");
(void)strcpy(wwvb_chan, DEFAULT_CHAN); /* == "TIME", "time", or "T" */

rounded = TRUE;
lagg = 0.0;

/* Check usage */
progname = argv[0];
if (argc < 4)
    error("Usage: %s -m min_rate ah_file [-vb code chan] [-v ] [-la | -l listfile] [-y year] [-r | -s] [-d delay] [
-nc]",
        argv[0]);

/* Interpret command line arguments */
for (--argc, argv++; argc; --argc, argv++) {

    /* Minimum sampling rate implies RMF window length */
    if (strcmp(*argv, "-m", 2)) {
        --argc, argv++;
        if (sscanf(*argv, "%f", &min_rate) != 1)
            error("Unreadable minimum digitization rate.");
    }

    /* listfile name, or do all traces */
    else if (strcmp(*argv, "-l", 2)) {
        if (strcmp(*argv, "-la", 3)) {
            all_traces = TRUE;
        }
        else {
            all_traces = FALSE;
            --argc, argv++;
            (void)strcpy(list_name, *argv);
        }
    }

    /* Verbose output? */
    else if (strcmp(*argv, "-v", 2) && !strcmp(*argv, "-vb", 3)) {
        printf("AHWWVB verbose output:\n");
        verbose = TRUE;
    }

    /* WWVB code and channel names */
    else if (strcmp(*argv, "-vb", 3)) {
        --argc, argv++;
        if (argc < 2)
            error("Bad WWVB code or channel.");
        (void)strcpy(wwvb_code, *argv);
        --argc, argv++;
        (void)strcpy(wwvb_chan, *argv);
    }

    /* new sampling interval */
    else if (strcmp(*argv, "-y", 2)) {
        --argc, argv++;
        if (sscanf(*argv, "%d", &year) != 1)
            error("Bad year in -y option.");
    }
}

```

```

    }

    /* "Rounded" or "sharp" WWVB trace? */
    else if (strneq(*argv, "-r", 2))
        rounded = TRUE;
    else if (strneq(*argv, "-s", 2))
        rounded = FALSE;

    /* Radio + flight time lag */
    else if (strneq(*argv, "-d", 2)) {
        --argc; argv++;
        if ((sscanf(*argv, "%lf", &lagg) != 1))
            error("Bad delay in -d option.");
        if (lagg < 0.0)
            error("NEGATIVE delay given--non-causal!");
    }

    /* Do not change traces */
    else if (strneq(*argv, "-nc", 3))
        change_ah = FALSE;

    /* WWVB-containing AH-file name */
    else
        (void)strcpy(ah_name, *argv);
}

/* Check for required arguments */
if (ah_name[0] == '\0')
    error("No WWVB-containing AH file given (\\"ah_file\\").");
if (min_rate < 0.0)
    error("No valid minimum sampling rate (\\"min_rate\\") given.");

/* Modify RMF window length */
if (rounded)
    RMF_win = 1 + 2 *
        ((int)(0.2 * min_rate * RMF_FUDGE_r) - 2);
else
    RMF_win = 1 + 2 *
        ((int)(0.2 * min_rate * RMF_FUDGE_s) - 2);

if (RMF_win < 3) {
    RMF_win = 0;
    if (verbose) {
        printf("Sampling interval too low;");
        printf(" no RMF filtering done.\n");
    }
}
else if (verbose)
    printf("Longest RMF window is of %d points.\n", RMF_win);

/* Inform user */
if (verbose)
    if (rounded)
        printf("Compensating for low-pass filtered WWVB.\n");
    else
        printf("Assuming unfiltered WWVB.\n");

if (verbose && (lagg > 0.0))
    printf("Correcting for %f ms flight time + radio lag.\n", lagg);

/* Get WWVB trace and header */
ah_file = fopen(ah_name, "r");
xdrstdio_create(&xdr_in, ah_file, XDR_DECODE);

n_traces = read_wwvb(&xdr_in, wwvb_code, wwvb_chan,
    &wwvb_head, &wwvb_trace, &wwvb_seq, ah_name,
    all_traces, &old_st, &old_int);

xdr_destroy(&xdr_in);
(void)fclose(ah_file);

/* Interpret WWVB */
(void)decode_WWVB(rounded, year, min_rate, RMF_win,
    wwvb_head, wwvb_trace, &new_st, &new_int, old_st, old_int,
    verbose);
new_st += (lagg / 1000.0);          /* Remove any delay */

/* By default, change ah traces accordingly. */
if (change_ah) {
    /* (-la) Create temporary list_file,
       a complete list of traces in ah_file */

    if (all_traces) {
        (void)sprintf(temp_name, "%s.XXXXXX", ah_name);
        (void)mktemp(temp_name);

        temp_file = fopen(temp_name, "w");
    }
}

```



```

        for (itrace = 1 ; itrace <= n_traces ; itrace++)
            fprintf(temp_file, "%s %d\n", ah_name, itrace);

        (void)fclose(temp_file);
    }

    /* (~1) check that WWVB trace is in list_file
       and check sequence in listfile */

    else {
        if (list_name[0] == '\0')
            error("Requires listfile name.");

        list_file = fopen(list_name, "r");

        rewind(list_file);
        if (!seq_ok(list_file, ah_name, wwvb_seq))
            error("listfile out of sequence or WWVB trace not listed there.");

        (void)fclose(list_file);

        (void)strcpy(temp_name, list_name);
    }

    /* Spawn AHCLK to make the necessary changes in AH headers */
    ch_files(temp_name, TRUE, new_st, TRUE, new_int, verbose);

    /* Clean up and exit */
    if (all_traces) {
        (void)sprintf(sys_str, "/bin/rm -f %s", temp_name);
        if ((sysret = (system(sys_str) >> 8) & 0377)) == 127)
            report("Could not delete temporary list file.");
    }

#ifdef DEBUG
    printf("/bin/rm returns %d.\n", sysret);
#endif

    }

    /* In all cases, report the result. */
    (void)datetime(new_st, &yr, &mo, &day, &hr, &mn, &sec, GREGORIAN);

    printf("\n-----ahwwvb-----\n");
    printf("File \"%s\", code \"%s\", channel \"%s\":\n",
        ah_name, wwvb_code, wwvb_chan);
    printf("Start time = %04d/%02d/%02d %02d:%02d:%07.4lf UTC\n",
        yr, mo, day, hr, mn, sec);
    printf("Sample interval = %.10lf s (%lf sps)\n",
        new_int, 1. / new_int);
    printf("-----\n\n");

    /* Clean up and quit */
    free((char *)wwvb_trace);
    exit(0);
}

/*
 * Check sequence in listfile and presence of WWVB.
 * If either is awry, return FALSE.
 */

typedef struct {
    char        fname_in[NM_LEN];
    int         rec_num;
    int         ugid;
} TLIST;

int
seq_ok(list_file, ah_name, wwvb_seq)
    FILE        *list_file;
    char        *ah_name;
    int         wwvb_seq;
{
    char        in_str[STR_LEN];          /* One line from list file */

    int         i;
    int         same_nm;
    int         seq;
    int         rv;
    int         vb;

    int         nlines;
    TLIST        *file_list[MX_LST];     /* known AH-file names */

    rv = TRUE;

```

```

vb = FALSE;
nlines = 0;
file_list[0] = (TLIST *)emalloc(sizeof(TLIST));

while (fgets(in_str, sizeof(in_str), list_file) != NULL) {
    if (sscanf(in_str, "%s%d%d",
               file_list[nlines] -> fname_in,
               &(file_list[nlines] -> rec_num),
               &(file_list[nlines] -> ugid))
        >= 2) {

        if (strcmp(file_list[nlines] -> fname_in, ah_name)
            && (file_list[nlines] -> rec_num == wwvb_seq))
            vb = TRUE;

        same_nm = TRUE;
        seq = file_list[nlines] -> rec_num;
        for (i = nlines; i >= 0; --i) {
            if (strcmp(file_list[nlines] -> fname_in,
                       file_list[i] -> fname_in)) {
                if (!same_nm)
                    rv = FALSE;
                if (file_list[i] -> rec_num >= seq
                    && i < nlines)
                    rv = FALSE;
            } else
                seq = file_list[i] -> rec_num;
        }

        else {
            same_nm = FALSE;
        }

        nlines++;
        if (nlines >= MX_LST)
            error("listfile too long.");
        file_list[nlines] = (TLIST *)emalloc(sizeof(TLIST));
    }
    else
        error("Bad listfile format.");
}

for (i = 0; i <= nlines; i++)
    free((char *)file_list[i]);

return (rv && vb);
}

/*
 * Call AHCLK to change the AH file headers.
 */

void
ch_files(temp_name, start, new_st, interval, new_int, verbose)
char *temp_name; /* List file name actually passed to AHCLK */
int start; /* Change start time if TRUE */
TIME new_st; /* Recalculated start time */
int interval; /* Change sampling interval if TRUE */
double new_int; /* Recalculated sampling interval */
bool verbose; /* Write out gory details? */
{
    int yr, mo, day, hr, mn;
    double sec;
    char sys_str[STR_LEN];
    char add_str[STR_LEN];
    int sysret;

    (void)sprintf(sys_str, "ahclk -l %s", temp_name);
    if (start) {
        (void)datetime(new_st, &yr, &mo, &day, &hr, &mn, &sec, GREGORIAN);
        (void)sprintf(add_str, "-s %04d %02d %02d %02d %02d %lf",
                      yr, mo, day, hr, mn, sec);
        (void)strcat(sys_str, add_str);
    }
    if (interval) {
        (void)sprintf(add_str, "-i %.10lf", new_int);
        (void)strcat(sys_str, add_str);
    }

    /* Report change */
    if (verbose) {
        printf("\nChanging start time and interval ...\n", sys_str);
        printf("\t%s\n\t", sys_str);
    }

    if ((sysret = (system(sys_str) >> 8) & 0377)) == 127)
        report("Trouble changing AH files.  CONDITION OF FILES UNKNOWN!");
}

```

```

    else if (verbose)
        printf("... successful.\n\n", sys_str);

#ifdef DEBUG
    printf("AHCLK returns %d.\n", sysret);
#endif
}

/*
 * Read all trace and header pairs. Return the WWVB trace and header,
 * its sequence (position) within the AH file, and the total number of
 * traces in the AH file. Allocates space for wwvb_trace, which must
 * be released elsewhere. If "-la" option is in force, make relatedness
 * check.
 */

int
read_wwvb(xdr_in, wwvb_code, wwvb_chan, wwvb_head, wwvb_trace,
          wwvb_seq, ah_name, all_traces, old_st, old_int)
XDR      *xdr_in;
char      *wwvb_code;      /* WWVB trace code */
char      *wwvb_chan;      /* WWVB trace channel */
ahhed     *wwvb_head;      /* Space for WWVB AH header */
float     **wwvb_trace;    /* Pointer to WWVB AH trace */
int       *wwvb_seq;       /* Position of WWVB trace */
char      *ah_name;        /* Name of AH file with WWVB trace */
int       all_traces;      /* TRUE if "-la" */
TIME      *old_st;         /* Return original starting time */
double    *old_int;        /* and sampling interval */
{
    ahhed     temp_head;    /* Space for arbitrary AH header */
    float     *temp_trace;  /* Pointer to WWVB AH trace */
    int       n_traces = 0; /* Total traces in ah_file */

    bool      first = TRUE; /* Flag first trace */
    bool      GOT_IT = FALSE; /* Flag whether WWVB found at all */

    long      t_length;     /* Trace length */
    TIME      t_start;      /* Original trace start time */
    TIME      t_temp;       /* Original trace start time */
    float      t_int;       /* Original trace sampling interval */

    while (TRUE) {
        /* Read header */
        if (xdr_gethead(&temp_head, xdr_in) != 1)
            break;

        /* Relatedness check */
        if (all_traces) {
            if (first) {
                first = FALSE;
                t_length = temp_head.record.ndata;
                t_start = ah2epoch(temp_head.record.abstime);
                t_int = temp_head.record.delta;
            }
            else {
                t_temp = ah2epoch(temp_head.record.abstime);

                if (temp_head.record.ndata != t_length
                    || t_temp != t_start
                    || temp_head.record.delta != t_int)
                    error("Traces not all related!");
            }
        }

        /* Successful */
        n_traces++;

        /* Allocate trace storage */
        temp_trace = (float *)mkdatspace(&temp_head);

        /* Read trace */
        if (xdr_getdata(&temp_head, (char *)temp_trace, xdr_in) == -1)
            error("Error reading data record.");

        /* Save WWVB when found */
        if (is_wwvb(temp_head, wwvb_code, wwvb_chan)) {
            *wwvb_head = temp_head;
            *wwvb_trace = temp_trace;
            *wwvb_seq = n_traces;

            /* Pass back originals */
            *old_st = ah2epoch(temp_head.record.abstime);
            *old_int = temp_head.record.delta;
        }
    }
}

```

```

        GOT_IT = TRUE;
    }

    else
        free((char *)temp_trace);
}

if (GOT_IT)
    return(n_traces);
else
    error("No WWVB trace found in %d trace(s) in \"%s\".",
        n_traces, ah_name);
}

TIME
ah2epoch(ah_t)
    struct ah_time ah_t;          /* Given an AH-header time ... */
                                  /* ... return epochal time      */
{
    return(timvar((int)ah_t.yr, (int)ah_t.mo, (int)ah_t.day,
        (int)ah_t.hr, (int)ah_t.mn, (double)ah_t.sec,
        GREGORIAN));
}

/*
 * Determine whether this trace is a WWVB trace.
 */

int
is_wwvb(temp_head, wwvb_code, wwvb_chan)
    ahhed temp_head; /* Space for arbitrary AH header */
    char *wwvb_code; /* WWVB trace code */
    char *wwvb_chan; /* WWVB trace channel if ==
        DEFAULT_CHAN, recognizes both
        "TIME" and "T" (note caps) */
{
    if (streq(temp_head.station.code, wwvb_code)) {
        if (streq(wwvb_chan, DEFAULT_CHAN)) {
            if (streq(temp_head.station.chan, "TIME")
                || streq(temp_head.station.chan, "time")
                || streq(temp_head.station.chan, "T"))
                return (TRUE);
            else
                return (FALSE);
        }
        else {
            if (streq(temp_head.station.chan, wwvb_chan))
                return (TRUE);
            else
                return (FALSE);
        }
    }
    else
        return (FALSE);
}

/*
 * Message and decode WWVB trace, the latter by calling C. Johnson's
 * routines. Report the signal processing and any fudges needed. Compile
 * and report statistics on lengths of WWVB "ticks" to reveal inconstant
 * digitizing rates.
 */

void
decode_WWVB(rounded, year, min_rate, RMF_win, wwvb_head, wwvb_trace,
    new_st, new_int, old_st, old_int, verbose)
    bool rounded; /* Is the trace "rounded" (filtered)? */
    int year; /* Year; overrides ahheader if != 0 */
    ahhed wwvb_head; /* WWVB AH header */
    float *wwvb_trace; /* Pointer to WWVB AH trace */
    TIME *new_st; /* New starting time */
    double *new_int; /* New sampling interval */
    float min_rate; /* Minimum digitizing rate possible */
    int RMF_win; /* Resulting maximal RMF window length */
    TIME old_st; /* Original starting time */
    double old_int; /* Original sampling interval */
    bool verbose; /* Write out gory details? */
{
    int ns; /* Number of samples in time code */
    int strt; /* Starting point in time code */
    int *ks; /* Record to decode */
    float step; /* Sampling rate */
    int ires; /* Decoding "score" (synch. level) */

```

```

/* Time of first sample as returned by WWVB routine of C. Johnson */
long      kda;          /* ("Julian") day of year      */
long      khr;          /* Hour of day                */
long      kmn;          /* Minute of hour             */
float     sec;          /* Seconds past minute        */

double     threshold;   /* Threshold (mean of trace) for
                        reducing float to binary (int) */
int        ii, jj;      /* Loop dummies               */
int        RMF_half;    /* Half-width RMF window length */

/* Variables for epochal time translation */
TIME       tpt;
int        tpyr, tpmo, tpd, tphr, tpmn;
double     tpsec;

#ifdef DEBUG
    for (ii = 0 ; ii < wwvb_head.record.ndata ; ii++)
        printf("%f\n", wwvb_trace[ii]);
#endif

/* Despise with long-window running-median filter (maximal
   filter sequence would be 3, 5, 7, 9, ..., RMF_win,
   which is slow but not prohibitive.) */

if (RMF_win >= 3) {

    /* First kill single-point spikes, which are common
       and can draw edges toward them by one sample */
    if (RMF_win > 3) {
        if (verbose)
            printf("Running RMF prefilter of 3 points ...\n");
        (void)median(wwvb_trace, wwvb_trace,
                     (int)wwvb_head.record.ndata, 3, FALSE);
    }

    /* And kill two-point spikes for the same reason */
    if (RMF_win > 5) {
        if (verbose)
            printf("Running RMF prefilter of 5 points ...\n");
        (void)median(wwvb_trace, wwvb_trace,
                     (int)wwvb_head.record.ndata, 5, FALSE);
    }

    /* And do a half window for really long final windows */
    RMF_half = 1 + 2 * (RMF_win / 4);
    if (RMF_half > 9) {
        if (verbose)
            printf("Running RMF prefilter of %d points ...\n",
                   RMF_half);
        (void)median(wwvb_trace, wwvb_trace,
                     (int)wwvb_head.record.ndata, RMF_half, FALSE);
    }

    /* Run the near-maximal RMF filter */
    if (verbose)
        printf("Running main RMF filter of %d points ...\n",
               RMF_win);
    (void)median(wwvb_trace, wwvb_trace,
                 (int)wwvb_head.record.ndata, RMF_win, FALSE);

#ifdef DEBUG
    for (ii = 0 ; ii < wwvb_head.record.ndata ; ii++)
        printf("%f\n", wwvb_trace[ii]);
#endif

    /* Trim unfiltered ends */
    ns = wwvb_head.record.ndata - RMF_win + 1;
    strt = (RMF_win - 1) / 2;
}

else {
    /* Use entire (unfiltered) trace */
    ns = wwvb_head.record.ndata;
    strt = 0;
}

/* Interpret the trace into binary */

ks = (int *)emalloc(sizeof(int) * ns);
(void)to_bin(rounded, min_rate, wwvb_head.record.ndata,
             strt, ns, wwvb_trace, ks);

#ifdef DEBUG
    for (ii = 0 ; ii < strt ; ii++)
        printf("-1\n");
    for (jj = 0 ; jj < ns ; jj++)
        printf("%d\n", ks[jj]);

```

```

    for (ii = 0 ; ii < strt ; ii++)
        printf("-1\n");
#endif DEBUG

/* Decode WWVB with C. Johnson's FORTRAN spaghetti */
if (verbose)
    printf("Interpreting time series ...\n");
wwvb_(&ns, &ks, &step, &kda, &khr, &kmn, &sec, &ires);
if (&ires <= 0)
    if (&kda == -1)
        error("ELSKREW apparantly in infinite loop. Giving up.");
    else
        error("WWVB was completely unreadable (&ires=%d).",
            &ires);

/* Since &ires > 0, new sampling interval OK. Verify and report it. */
if ((&min_rate > 0.0) && (1. / &step) < &min_rate) {
    fprintf(stderr, "New sampling interval is lower");
    fprintf(stderr, " than -m argument claimed.\n");
    error("Filtering may have failed; files NOT changed.");
}
else
    *new_int = &step;

if (verbose && (*new_int != old_int))
    printf("Sampling rate changed from %.3f to %.3f sps.\n",
        1. / old_int, 1. / (*new_int));

/* Compile and report statistics on the duration of WWVB "ticks" */
tick_size(&ns, &ks, &step, verbose);

/* If partly decoded, fill in any unknowns with old values */
/* &ires: */
/* 0 = total failure to decode */
/* 1 = fractional "sec" OK */
/* 2 = units of "sec" OK too */
/* 3 = all of "sec" OK */
/* 5 = fully decoded (Jday, hour, and minute OK too) */

(void)datetime(old_st,
    &tpyr, &tpmo, &tpdy, &tphr, &tpmn, &tpsec, GREGORIAN);

if (&ires < 5) {
    printf("WWVB was only partly readable (&ires=%d):\n", &ires);

    printf("\tDefaulting to original month (%d), day (%d),",
        &tpmo, &tpdy);
    printf(" hour (%d), and minute (%d).\n", &tphr, &tpmn);

    /* If part of seconds unknown, default them to old values */
    if (&ires < 3) {
        double tens, units, fracs;

        if (&sec < 0.0 || &tpsec < 0.0)
            error("Programming bug 1.");

        if (&ires == 1) {
            units = aint(&tpsec);
            fracs = &sec - aint((double)&sec);
            &tpsec = units + fracs;

            printf("\tUsing WWVB fractional seconds");
            printf(" (%.3f) with original units", fracs);
            printf(" of seconds (%.0f s).\n", units);
        }

        else if (&ires == 2) {
            tens = 10. * aint(&tpsec / 10.);
            fracs = (&sec / 10.);
            fracs = 10. * (fracs - aint(fracs));
            &tpsec = tens + fracs;

            printf("\tUsing WWVB units and fractional");
            printf(" seconds (%.3f) with original", fracs);
            printf(" tens of seconds (%.0f s).\n", tens);
        }

        else
            error("Programming bug 2.");
    }

    else if (&ires == 3)
        &tpsec = &sec;

    else
        error("Programming bug 3.");
}

```

```

/* Decide what year to use. If 0 (the default),
   use the original trace starting year */
if (year > 0) {
    if (verbose)
        if (year != tpyr)
            printf("Changing year to %d.\n", year);
        else
            printf("Keeping year at %d.\n", year);
    tpyr = year;
}
else if (verbose)
    printf("Defaulting year to %d.\n", tpyr);

/* If fully decoded, use WWVB date and time */
if (ires == 5) {
    /* Translate J-day to month and day */
    mnday(kda, isleap(tpyr, GREGORIAN), &tpmo, &tpdy);

    tphr = khr;
    tpmn = kmn;
    tpsec = sec;
}

/* Set start time back to beginning of wwvb_trace */
tpsec -= (float)strt * step;

/* Take second ticks to be half way between the 0 and 1
   samples of the binary trace */
tpsec -= 0.5 * step;

/* Resolve and store new starting time via epochal time. */
*new_st = timvar(tpyr, tpmo, tpdy, tphr, tpmn, tpsec, GREGORIAN);

/* Clean up */
free((char *)ks);
}

/*
 * Compile and report statistics on WWVB "tick" lengths.
 */

void
tick_size(ns, ks, step, verbose)
int ns; /* Number of samples in time code */
int *ks; /* Record to decode */
float step; /* Sampling rate */
bool verbose; /* Write out gory details? */
{
    int width[MAXTRANS]; /* Tick widths */
    int n_width; /* How many there are */
    int ii; /* Loop dummy */
    bool first; /* Flag first transition */
    int last; /* Previous tick location */
    float span; /* Range of intervals */

    /* Gather tick widths */
    n_width = 0;
    first = TRUE;
    for (ii = 0; ii < ns - 1; ii++) {
        if ((ks[ii + 1] - ks[ii]) == 1) { /* Up tick */
            if (first) {
                last = ii;
                first = FALSE;
            }
            else {
                width[n_width] = ii - last;
                last = ii;
                n_width++;
            }
        }
    }

    /* Sort widths */
    (void)qsort((char *)width, n_width, sizeof(int), icomp);

    /* Report various ranges and deviations */

    span = (step * width[nint(0.95 * (double)(n_width - 1))])
        - (step * width[nint(0.05 * (double)(n_width - 1))]);

    if (verbose || (span > SPAN_WARN * step)) {
        printf("\nSecond intervals range from %.3f to %.3f s.\n",
            step * width[0], step * width[n_width - 1]);

        if (span > SPAN_WARN * step)
            printf("WARNING: ");
        printf("5th and 95th percentiles differ by %.3f s.\n\n", span);
    }
}

```

```

    }
}

/*
 * Comparison rule for qsort
 */

static int
icomp(a, b)
    int      *a, *b;
{
    if (*a > *b)
        return (1);
    else if (*a < *b)
        return (-1);
    else
        return (0);
}

/*
 * Interpret WWVB code into binary by finding transition points.
 *
 * If "rounded==TRUE", the transition is taken to happen 1 sample before
 * the interval with the largest slope. This empiricle artifice accounts
 * for low-pass filtering of WWVB by the playback discriminator, and is
 * about right near 100 sps. If "rounded==FALSE", the interval with the
 * largest slope is taken to be the interval with the transition. This
 * assumption is appropriate for CUSP data, and probably much other.
 *
 * If the difference peak has a flat top, binary output transitions at
 * the earliest part of peak.
 *
 * This routine CHANGES the input trace (wwvb_trace).
 */

void
to_bin(rounded, min_rate, npts, strt, ns, wwvb_trace, ks)
    bool      rounded;          /* Is the trace "rounded" (filtered)? */
    float     min_rate;         /* Minimum digitizing rate possible */
    long      npts;             /* Number of samples in input code */
    int       ns;               /* Number of samples in output code */
    int       strt;             /* Starting point of output within
                                the input array */
    float     *wwvb_trace;      /* Pointer to WWVB AH trace */
    int       *ks;               /* Record to decode */
{
    float     *mins, *maxs;     /* Arrays of mins and maxs */
    int       nmins, nmaxs;     /* Number of mins and maxs */
    float     min_th, max_th;   /* Thresholds for mins and maxs */

    int       scan_win;         /* Window length for "global" extrema */
    int       ii, jj, kk;       /* Loop dummies */
    bool      first;             /* Flag first transition */
    int       new_val, old_val; /* Used in binary translator */

    int       bias;             /* Put transitions of the binary
                                trace this many sample intervals
                                from the largest-first-difference
                                interval */

    /* Initialize */
    mins = (float *)emalloc(sizeof(float));
    nmins = 0;
    maxs = (float *)emalloc(sizeof(float));
    nmaxs = 0;

    if (rounded)
        bias = -1;
    else
        bias = 0;

    /* Take first difference of input
     (points contain the following-interval first difference) */
    for (ii = 0; ii < npts - 1; ii++)
        wwvb_trace[ii] = wwvb_trace[ii + 1] - wwvb_trace[ii];
    wwvb_trace[ii] = wwvb_trace[ii - 1]; /* kludge */

    /* To determine appropriate thresholds in the first difference,
     first collect all local mins and max's */
    for (ii = strt + 1, jj = 1; jj < ns - 1; ii++, jj++) {
        if (wwvb_trace[ii] > wwvb_trace[ii - 1]
            && wwvb_trace[ii] > wwvb_trace[ii + 1]) {
            maxs[nmaxs] = wwvb_trace[ii];
            nmaxs++;
            maxs = (float *)erealloc((char *)maxs,

```



```

        sizeof(float) * (nmaxs + 1));
    }

    if (wwvb_trace[ii] < wwvb_trace[ii - 1]
        && wwvb_trace[ii] < wwvb_trace[ii + 1]) {

        mins[nmins] = wwvb_trace[ii];
        nmins++;
        mins = (float *)erealloc((char *)mins,
            sizeof(float) * (nmins + 1));
    }
}

if (nmaxs <= 0)
    error("No local maxima found in first difference.");
if (nmins <= 0)
    error("No local minima found in first difference.");

/* Use TH_FUDGE * the TH_PER'th percentile as the threshold */

(void)qsort((char *)maxs, nmaxs, sizeof(float), fcomp);
max_th = TH_FUDGE * maxs[(int)((float)nmaxs * TH_PER / 100.)];

(void)qsort((char *)mins, nmins, sizeof(float), fcomp);
min_th = TH_FUDGE * mins[(int)((float)nmins * (1.0 - TH_PER / 100.))];

/* Find transition points and translate to binary */

/* Find the "global" peaks by scanning for the min/max a window */
scan_win = MAX(1, 0.1 * min_rate); /* May fail at low sampling rates */

/* Value prior to first transition not known at beginning */
first = TRUE;
for (ii = strt, jj = 0 ; jj < ns - (1 + bias) ; ii++, jj++) {
    if (is_max(wwvb_trace, ii, npts, scan_win, max_th)) {
        new_val = 1;
        old_val = 0;
        ks[jj + 1 + bias] = new_val;

        if (first) {
            for (kk = 0 ; kk < jj + 1 + bias ; kk++)
                ks[kk] = old_val;
            first = FALSE;
        }
    }

    else if (is_min(wwvb_trace, ii, npts, scan_win, min_th)) {
        new_val = 0;
        old_val = 1;
        ks[jj + 1 + bias] = new_val;

        if (first) {
            for (kk = 0 ; kk < jj + 1 + bias ; kk++)
                ks[kk] = old_val;
            first = FALSE;
        }
    }

    else if (!first)
        ks[jj + 1 + bias] = new_val;
}

/*
 * Comparison rule for qsort
 */

static int
fcomp(a, b)
    float *a, *b;
{
    if (*a > *b)
        return (1);
    else if (*a < *b)
        return (-1);
    else
        return (0);
}

/*
 * Find "global" maximum.
 */

bool
is_max(wwvb_trace, ii, npts, scan_win, max_th)
    float *wwvb_trace; /* Pointer to WWVB AH trace */
    int ii; /* Index to wwvb_trace */

```

```

long      npts;          /* Number of samples in input code */
int       scan_win;      /* Window length for "global" extreema */
float     max_th;        /* Threshold */

{
    int     left, right;  /* Window indicies */
    int     jj;          /* Loop dummy */

    /* If not a local maximum (or flat), it cannot be a global maximum */
    if (wwvb_trace[ii] < wwvb_trace[MAX(0, ii - 1)])
        if (wwvb_trace[ii] < wwvb_trace[MIN(npts, ii + 1)])
            return (FALSE);

    /* Is it the largest point in window? */
    left = MAX(0, ii - (int)(scan_win / 2));
    right = MIN(npts, ii + (int)(scan_win / 2));
    for (jj = left; jj < right; jj++)
        if (wwvb_trace[jj] > wwvb_trace[ii])
            return (FALSE);

    /* If the point at ii IS a local maximum (or flat),
       IS (one of) the largest point(s) in the window, and
       IS greater than the threshold,
       it is a "global maximum" */
    if (wwvb_trace[ii] >= max_th)
        return (TRUE);
    else
        return (FALSE);
}

/*
 * Find "global" minimum.
 */

bool
is_min(wwvb_trace, ii, npts, scan_win, min_th)
float      *wwvb_trace; /* Pointer to WWVB AH trace */
int        ii;          /* Index to wwvb_trace */
long       npts;        /* Number of samples in input code */
int        scan_win;    /* Window length for "global" extreema */
float      min_th;      /* Threshold */

{
    int     left, right;  /* Window indicies */
    int     jj;          /* Loop dummy */

    /* If not a local minimum (or flat), it cannot be a global minimum */
    if (wwvb_trace[ii] > wwvb_trace[MAX(0, ii - 1)])
        if (wwvb_trace[ii] > wwvb_trace[MIN(npts, ii + 1)])
            return (FALSE);

    /* Is it the smallest point in window? */
    left = MAX(0, ii - (int)(scan_win / 2));
    right = MIN(npts, ii + (int)(scan_win / 2));
    for (jj = left; jj < right; jj++)
        if (wwvb_trace[jj] < wwvb_trace[ii])
            return (FALSE);

    /* If the point at ii IS a local minimum (or flat),
       IS (one of) the smallest point(s) in the window, and
       IS less than the threshold,
       it is a "global minimum" */
    if (wwvb_trace[ii] <= min_th)
        return (TRUE);
    else
        return (FALSE);
}

```

```
/*
 * AH filter AHWWVB (AH WWVB reader) include file.
 */

#define NM_LEN 128      /* Maximum file-name length */
#define STR_LEN 256     /* Maximum scratch-string length */
#define MX_LST 1024     /* Maximum line count of listfile */

/* Recognize WWVB channel names "TIME", "time", or "T" */

#define DEFAULT_CHAN    "DFult"

/* Reduce RMF window by this factor
   to account for slow transitions in the reciever */

#define RMF_FUDGE_r     0.60
#define RMF_FUDGE_s     0.80

/* Use TH_FUDGE * the TH_PER'th percentile as the threshold
   determining what a "sharp corner" is (for interpreting trace
   into binary form). */

#define TH_FUDGE        0.5
#define TH_PER          98.

/* How many WWVB "ticks" to compile statistics on */
#define MAXTRANS        2000

/* A warning is issued if WWVB "ticks" (95th - 5th percentile)
   vary by more than this many sample intervals */
#define SPAN_WARN       5.0
```

```

/*      structure for data file header  --      witte, 11 June 85      */

#define AHHEADSIZE 1024
#define CODESIZE 6
#define CHANSIZE 6
#define STYPESIZE 8
#define COMSIZE 80
#define TYPEMIN 1
#define TYPEMAX 6
#define LOGSIZE 202
#define LOGENT 10
#define NEXTRAS 21
#define NOCALPTS 30

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

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

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

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

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

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

struct station_info {
    char        code[CODESIZE]; /* station code          */
    char        chan[CHANSIZE]; /* lpz,spn, etc.         */
    char        stype[STYPESIZE]; /* wwssn,hglp,etc.      */
    float       slat;      /* station latitude      */
    float       slon;      /* " longitude           */
    float       elev;      /* " elevation           */
    float       DS;        /* gain                  */
    float       AO;        /* normalization        */
    struct calib cal[NOCALPTS]; /* calibration info      */
};

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

struct record_info {
    short       type;      /* data type (int,float,...) */
    long        ndata;     /* number of samples        */
    float       delta;     /* sampling interval        */
    float       maxamp;    /* maximum amplitude of record */
    struct ah_time abstime; /* start time of record section */
    float       rmin;      /* minimum value of abscissa */
    char        rcomment[COMSIZE]; /* comment line          */
    char        log[LOGSIZE]; /* log of data manipulations */
};

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

```

```
#define FLOAT 1
#define COMPLEX 2
#define VECTOR 3
#define TENSOR 4
#define DOUBLE 6
#define INTEGER 7
```

```

c
c-----*-----*-----*-----*-----*-----*-----*-----*-----*-----*
c
c-----elskew : ll solution to n equations in m unknowns
c
c      subroutine elkew (m,n,at,d,gu,gd,small,x)
c
c      Note : transpose form
c
c      find x(i) to minimize:
c
c      
$$esum = \sum_{k=1}^n (skewnorm(k, \sum_{i=1}^m (d(k)-at(i,k)*x(i)) ))$$

c
c      skewnorm(k,er) = er * ( gu(k), if (er.gt.+small) )
c      ( 0.0 , if error is small )
c      ( gd(k), if (er.lt.-small) )
c
c-----Note: This routine has been modified to accept the transpose of
c      the 'a' matrix. This was done because it is more convenient
c      to set up a packed transpose when one does not know
c      the number of equations a priori.
c      -Carl
c
c      real at(m,n),x(m),d(n),gu(n),gd(n)
c      real w(2000),f(2000),b(4,4),col(4),row(4)
c      real sc(4),g(4),gp(4),gm(4)
c      real hit,small,wt,oldk,gr,tk,t,esum
c
c      integer k(2000)
c      integer i,j,k,l,m,n,kick,loop,ml,mh,new
c
c-----Also change value in format statement 160
c      integer*4 looped,MXLOOP
c      parameter (MXLOOP=100000)
c
c      if (m.lt.5.and.n.lt.501) goto 5
c      return
c
c 5      looped = 0
c
c      do 30 j=1,m
c          x(j)=0.
c          do 10 i=1,m
c              b(i,j)=0.
c              sc(j)=0.
c              do 20 i=1,n
c                  sc(j)=sc(j)+abs(at(j,i))
c              continue
c              b(j,j)=n/sc(j)
c          continue
c      continue
c
c      do 40 i=1,n
c          f(i)=d(i)
c          k(i)=i
c      continue
c      loop=0
c 50      loop=loop+1
c          kick=1+mod(loop-1,m)
c          if (loop.le.m) goto 190
c
c-----find best equation to -kick- out of basis
c      do 110 i=1,m
c          gp(i)=0.
c          gm(i)=0.
c          g(i)=0.
c 110      continue
c
c      do 135 l=1,n
c          if (abs(f(l)).lt.small) goto 120
c          if (f(l).gt.small) hit=gu(l)
c          if (f(l).lt.small) hit=gd(l)
c          do 115 j=1,m
c              g(j)=g(j)-at(j,l)*hit
c          continue
c          goto 135
c
c 120      do 130 i=1,m
c          wt=0.
c          do 125 j=1,m
c              wt=wt+at(j,l)*b(j,i)
c          continue
c 125      if (wt.lt.0.) gp(i)=gp(i)-gu(l)*wt
c          if (wt.gt.0.) gp(i)=gp(i)-gd(l)*wt
c          if (wt.gt.0.) gm(i)=gm(i)-gu(l)*wt
c          if (wt.lt.0.) gm(i)=gm(i)-gd(l)*wt
c
c

```

```

130     continue
135     continue
c
      oldk=0.
      do 150 i=1,m
        gr=0.
        do 140 j=1,m
          gr=gr+g(j)*b(j,i)
140      continue
          if ((gr+gp(i))*(gr+gm(i)).lt.0.) goto 150
          tk=min(abs(gr+gp(i)),abs(gr+gm(i)))
          if (tk.gt.oldk) kick=1
          if (tk.gt.oldk) oldk=tk
150     continue
c
      if (oldk.eq.0.) return
c
c-----Give up if seem to be in infinite loop.
c      Message this with zero vector (not adequate in general!)
c
      looped = looped + 1
      if (looped.gt.MXLOOP) then
        do 160 j=1,m
          x(j) = 0.0
160     continue
        return
      endif
c
190     continue
c
c-----find scalar t where x=x0+(col of b)*t
      do 60 i=1,n
        w(i)=0.
        do 60 j=1,m
          w(i)=w(i)+at(j,i)*b(j,kick)
60     continue
      call skewer(2000,n,w,f,gu,gd,small,k,t,ml,mh)
c
c-----pick out new basis
      new=k(ml)
      do 70 l=ml,mh
        if (abs(w(new)).lt.abs(w(k(l)))) new=k(l)
70     continue
      t=f(new)/w(new)
      esum=0.
      do 80 i=1,n
        f(i)=f(i)-w(i)*t
        if (f(i).gt. 0.) esum=esum+gu(i)*f(i)
        if (f(i).lt. 0.) esum=esum+gd(i)*f(i)
80     continue
c
c-----update x and basis matrix as described(transposed) in hadley p.
      do 90 j=1,m
        col(j)=b(j,kick)
        x(j)=x(j)+col(j)*t
        row(j)=0.
        do 85 i=1,m
          row(j)=row(j)-at(i,new)*b(i,j)/w(new)
85     continue
90     continue
      row(kick)=1./w(new)-1.
      do 100 i=1,m
        do 95 j=1,m
          b(i,j)=b(i,j)+col(i)*row(j)
95     continue
100    continue
      goto 50
c
      end
c
c-----*****
c
      subroutine skewer (nd,n,w,f,gu,gd,small,k,t,ml,mh)
c
c      solve rank 1 overdetermined equations with skew norm
c
c      inputs- n,w,f,u,d,small,k.    outputs- k,t,ml,mh.
c
c      find t to minimize
c
c      n
c      is = sum skewnorm(k,f(k)-w(k)*t)
c          k=1
c
c      where:
c
c          ( gu(k)*(er-small) if er.gt.+small      gu.gt.0
c      skewnorm(k,er) = ( gd(k)*(er+small) if er.lt.-small  gd.lt.0
c                      ( 0.          ) if abs(er).le.small.ge.0.

```

```

c
c   ggu,ggd,w,and f are referenced indirectly as w(k(i)),i=1,n etc.
c   minima will be at equations k(ml),k(ml+1),...k(mh)
c
c   real w(nd),f(nd),gu(nd),gd(nd)
c   real g(2001)
c   real gn,gp,small,er,t,gnt,gplx,gmix,grad,gpt
c
c   integer k(nd)
c   integer low,large,ml,mh,itry,n,kix,i,l,k,mlt,mht,nd,ixg
c
c   low=1
c   large=n
c   ml=n
c   mh=1
c   gn=0.
c   gp=0.
c
c   do 50 itry=1,n
c     kix=low+mod((large-low)/3+itry,large-low+1)
c     l=k(kix)
c     if (abs(w(l)).eq.0.) goto 50
c     t=f(l)/w(l)
c     f(l)=w(l)*t
c     do 10 i=low,large
c       l=k(i)
c       er=f(l)-w(l)*t
c       g(l)=0.
c       if (er.gt.small) g(l)=-w(l)*gu(l)
c       if (er.lt.-small) g(l)=-w(l)*gd(l)
10    continue
c     call split(low,large,k,g,mlt,mht)
c     gnt=gn
c     do 20 i=low,mlt
c       ixg=k(i)
c       gnt=gnt+g(ixg)
20    continue
c     gpt=gp
c     do 30 i=mht,large
c       ixg=k(i)
c       gpt=gpt+g(ixg)
30    continue
c     gplx=0.
c     gmix=0.
c     do 40 i=mlt,mht
c       l=k(i)
c       if (w(l).lt.0.) gplx=gplx-w(l)*gu(l)
c       if (w(l).gt.0.) gplx=gplx-w(l)*gd(l)
c       if (w(l).gt.0.) gmix=gmix-w(l)*gu(l)
c       if (w(l).lt.0.) gmix=gmix-w(l)*gd(l)
40    continue
c     grad=gnt+gpt
c     if ((grad+gplx)*(grad+gmix).lt.0.) goto 60
c     if (grad.ge.0.) low=mht+1
c     if (grad.le.0.) large=mlt-1
c     if (low.gt.large) goto 60
c     if (grad.ge.0.) gn=gnt+gmix
c     if (grad.le.0.) gp=gpt+gplx
c     if ((grad+gplx).eq.0.) ml=mlt
c     if ((grad+gmix).eq.0.) mh=mht
50    continue
c
c   60 ml=min0(ml,mlt)
c     mh=max0(mh,mht)
c
c   return
c   end
c
c-----
c
c   subroutine split (low,large,k,g,mlpass,mhpass)
c
c   given g(k(i)),i=low,large
c   then rearrange k(i),i=low,large and find ml,mh so that
c   (g(k(i)),i=low,(ml-1)) .lt. 0 and
c   (g(k(i)),i=ml,mh)=0. and
c   (g(k(i)),i=(mh+1),large) .gt.0.
c
c   real g(5000)
c
c   integer k(large)
c   integer ml,mh,ixg,keep,k,ii,i,ixgml,ixgmh,
c   * mlpass,mhpass,low,large
c
c   ml=low
c   mh=large
10  ml=ml-1
20  ml=ml+1

```



```
      ixg=k(ml)
      if (g(ixg)) 20,30,30
30    mh=mh+1
40    mh=mh-1
      ixg=k(mh)
      if (g(ixg)) 50,50,40
50    keep=k(mh)
      k(mh)=k(ml)
      k(ml)=keep
      ixgml=k(ml)
      ixgmh=k(mh)
      if (g(ixgml).ne.g(ixgmh)) goto 10
c
      do 60 i=ml,mh
        ii=i
        ixg=k(i)
        if (g(ixg).ne.0.0) goto 70
60    continue
c
      mlpass=ml
      mhpass=mh
      return
c
70    keep=k(mh)
      k(mh)=k(ii)
      k(ii)=keep
      goto 30
c
      end
```

```

c
c-----*-----*-----*-----*-----*-----*-----*-----*-----*-----*
c
c-----wwvb : wwvb decoder (determines absolute time of first sample)
c
c in          ** **
c usage : call wwvb(minrat, ns, ks, step, kda, khr, kmn, sec, ires)
c out          ** **** ** ** ** **
c
c      ns : number of sample in time code
c      ks : record to decode (standard integers,
c           already massaged into unspikey binary)
c      step : sampling rate
c      kda : ("Julian") day of year      (ires = 5)
c           (-1 indicates infinite loop exit from elskew)
c      khr : hour of day                  (ires = 5)
c      kmn : minute of hour               (ires = 5)
c      sec : seconds past minute          (ires = 3, 5)
c           seconds past 10 sec mark      (ires = 2)
c           seconds past 1 sec mark       (ires = 1)
c
c      ires : synchronization level obtained
c           0 = total failure to decode
c           1 = 1 second synch. (fractional "sec" now OK)
c           2 = 10 second synch. (units of "sec" OK too)
c           3 = minute synch. (all of "sec" now OK)
c           5 = fully decoded (day, hour, and minute OK too)
c
c      blame : carl johnson (9 apr 83)
c
c      Modified for hard typing and external signal processing by
c      J. R. Evans, USGS, Menlo Park, March, 1992.
c      Also demystified the use of "t0" and "td" in "rdecod"
c      common block--they had been assumed to form a
c      sequential pair in memory due to their places in
c      the common block (dangerous and obscure programming).
c      These values now are buffered through "tt".
c
c      subroutine wwvb (ns, ks, step, kda, khr, kmn, sec, ires)
c
c      real s,slast,wt,rate,sdel,step,small,xj10,xj60,
c      * secs,tt(2)
c      real sum,sex
c
c      integer ks(1)
c      integer i,i1,i2,i3,i4,i5,j,j1,j2,js,ns,is,ismax,isin,nsyn,
c      * ires,krate,i10,maxi10,mknt,iat,itick,ntick,n,
c      * kode,j10,n10,j60,imn,imn10,jmin,ibase,ihr,ihr10,
c      * ida,idal0,idal100
c      integer kbias,ksis
c
c      integer*4 kda
c      integer*4 khr
c      integer*4 kmn
c      real sec
c
c      include 'rdecod.inc'
c
c      ismax = ns - 1
c
c      kda = 0
c
c      c-----signal conditioning
c      c----- (replaced by robust processing in the calling routine)
c      c*
c      c*-----bias calculation
c      c*      sum = 0.0
c      c*      do 120 is=1, ns
c      c*          sum = sum + ks(is)
c      c*120 continue
c      c*      kbias = sum / ns
c      c*
c      c*-----convert to binary representation
c      c*      do 130 is=1, ns
c      c*          ksis = 0
c      c*          if (ks(is) .gt. kbias) ksis = 1
c      c*          ks(is) = ksis
c      c*130 continue
c      c*
c      c*-----first order deglitching
c      c*      do 150 is = 2, ismax
c      c*          if (ks(is) .ne. ks(is-1) .and. ks(is) .ne. ks(is+1))
c      c*              *      ks(is) = ks(is+1)
c      c*150 continue
c
c      c-----Calculate approximate digitization interval
c
c      do 220 isyn = 1, MAXSYN

```

```

        ksyn(isyn) = 0
220 continue
c
c Rising-edge detector
isyn = 0
do 230 is = 1, ismax
    if (isyn .lt. MAXSYN) isyn = isyn + 1
    if (ks(is) .eq. 1) goto 230
    if (ks(is+1) .eq. 0) goto 230
    ksyn(isyn) = ksyn(isyn) + 1
    isyn = 0
230 continue
c
c Calculate approximate sampling period ("step")
kper = 0
mknt = 0
do 250 isyn = 1, MAXSYN
    if (ksyn(isyn) .le. mknt) goto 250
    mknt = ksyn(isyn)
    krate = isyn
250 continue
c
rate = 0.0
wt = 0.0
do 260 i=1, 3
    j = krate + i - 2
    rate = rate + j * ksyn(j)
    wt = wt + ksyn(j)
260 continue
rate = rate / wt
step = 1.0 / rate
c write(*, *) 'rate, step:', rate, step
c
ires = 0
if (krate .le. 1 .or. krate .ge. MAXSYN) return
c
c-----Calculate time code phase
slast = -10.0
c
do 315 i=1, 10
    k10(i) = 0
315 continue
nsyn = 0
c
do 320 is = 1, ismax
    if (ks(is) .eq. 1) goto 320
    if (ks(is+1) .eq. 0) goto 320
    s = is * step
    sdel = s - slast
    if (sdel .lt. 1.0 - 2.0 * step) goto 320
    slast = s
    if (sdel .gt. 1.0 + 2.0 * step) goto 320
    js = s
    s = s - js
c-----Initial crude phase estimate
i10 = 10.0 * s
i10 = i10 + 1
if (i10 .gt. 10) i10 = 1
k10(i10) = k10(i10) + 1
c
nsyn = nsyn + 1
ksyn(nsyn) = is
c type *, nsyn, is, s
if (nsyn .ge. MAXSYN) goto 340
320 continue
if (nsyn .lt. 10) goto 680
c
c-----First crude phase (fractional seconds) estimate
340 max10 = 0
do 345 i=1, 10
    if (k10(i) .lt. max10) goto 345
    max10 = k10(i)
    t0 = (i-1) * 0.10
345 continue
c type *, 'crude t0:', t0
c
c-----Setup normal equations (to find start-time fractional
c seconds and sample interval): t = t0 + step * is
c
do 350 i=1, nsyn
    gu(i) = 1.0
    gd(i) = -1.0
    iat = 2 * (i - 1)
    at(iat + 1) = 1.0
    at(iat + 2) = ksyn(i)
c xx = step * ksyn(i) - t0
t(i) = anint(step * ksyn(i) - t0)
c type *, i, ksyn(i), t(i), xx

```

```

350 continue
c
c-----Solve for final "t0" and "step"
small = step * 0.01
call elskew (2, nsyn, at, t, gu, gd, small, tt)
c
c-----If exited apparant infinite loop, then bug out, indicating garbage
if (tt(1).eq.0.0 .and. tt(2).eq.0.0) then
    ires = 0
    kda = -1
    return
endif

t0 = tt(1)
td = tt(2)
if (t0 .lt. 0.0) t0 = t0 + 1.0
if (t0 .ge. 1.0) t0 = t0 - 1.0
step = td
sec = t0 + td
c
c-----Valid sampling interval ("step") and
fractional seconds of start time ("sec")
ires = 1

c    write(*, *) 'sync. 1 :', sec
c
c    do 380 i=1, nsyn
c        ti = t0 + step * ksyn(i)
c        type *, ti
c 380 continue
c
c-----Translate time code elements (0-fubar, 1-zero, 2-one, 3-mark)
ntick = ismax * step
if (ntick .gt. MAXSYN) ntick = MAXSYN
c
rate = 1.0 / step
krate = rate
do 490 itick = 1, ntick
    i1 = anint((itick - t0) / step) + 0.5
    if (i1 .lt. 1) goto 470
    i2 = i1 + 0.2 * rate
    i3 = i1 + 0.5 * rate
    i4 = i1 + 0.8 * rate
    i5 = i1 + rate
c
c-----Test first segment
n = i2 - i1
j = 0
do 425 i=1, n
    j = j + ks(i1 + i)
425 continue
if (j .lt. 0.7 * n) goto 470 ! fubar
kode = 1
c
c-----Test second segment
n = i3 - i2
j = 0
do 435 i=1, n
    j = j + ks(i2 + i)
435 continue
if (j .lt. 0.3 * n) goto 450 ! tick = zero
if (j .lt. 0.7 * n) goto 470 ! fubar
kode = 2
c
c-----Test third part
n = i4 - i3
j = 0
do 445 i=1, n
    j = j + ks(i3 + i)
445 continue
if (j .lt. 0.3 * n) goto 460 ! tick = one
if (j .lt. 0.7 * n) goto 470 ! fubar
kode = 3
goto 460
c
c-----Test third segment low
450 n = i4 - i3
j = 0
do 455 i=1, n
    j = j + ks(i3 + i)
455 continue
if (j .gt. 0.3 * n) goto 470 ! fubar
c
c-----Test fourth segment low
460 n = i5 - i4
j = 0
do 465 i=1, n
    j = j + ks(i4 + i)

```

```

465      continue
      if (j .gt. 0.3 * n) goto 470 ! fubar
      goto 480
c
c-----Fubar
470      kode = 0
c
480      ksyn(itick) = kode
      j1 = i1 + 1
      j2 = i5
c      write(*, 5480) itick, kode, (ks(j), j=j1, j2)
5480      format(' ', 2i5, ' ', 6i1)
c
490      continue
c
c      write(*, 5490) (ksyn(i), i=1, ntick)
5490      format(' ', 60i1)
c
c-----10-second synchronization
      i10 = 0
c
      do 520 i=1, 10
      k10(i) = 0
520      continue
c
      do 530 i=1, ntick
      i10 = i10 + 1
      if (i10 .gt. 10) i10 = 1
      if (ksyn(i) .eq. 3) k10(i10) = k10(i10) + 1
530      continue
c
      j10 = 0
      n10 = 0
      do 540 i=1, 10
      if (k10(i) .lt. n10) goto 540
      j10 = i
      n10 = k10(j10)
540      continue
c
      if (j10 .lt. 1) return
      xj10 = (j10 - t0) / step
      secs = 9.0 - step * (xj10 - 1.0)
c
c      if (secs .ge. 10.0) secs = secs - 10.0
c      if (secs .lt. 0.0) secs = secs + 10.0
c
      if (secs .ge. 10.0) then
      secs = secs - 10.0
      j10 = j10 + 10
      endif
c
      if (secs .lt. 0.0) then
      secs = secs + 10.0
      j10 = j10 - 10
      endif
c
c-----Units of seconds also now valid.
      sec = secs
      ires = 2
c
c      write(*, *) 'sync. 10 :', sec
c
c-----Minute synchronization
      j60 = j10 - 10
c
550      j60 = j60 + 10
      if (j60 .gt. ntick) return
      if (ksyn(j60 + 1) .ne. 3) goto 550
c
c-----Minute mark found
      xj60 = (j60 - t0) / step
      secs = 59.0 - step * (xj60 - 1.0)
      if (secs .ge. 60.0) secs = secs - 60.0
      if (secs .lt. 0.0) secs = secs + 60.0
c
c-----Tens of seconds also now valid
      sec = secs
      ires = 3
c
c      write(*, *) 'sync. 60 :', sec
c
c      do 580 i=1, 300
c      sex = sec + (i-1) * step
c      type *, i, ks(i), sex
c 580      continue
c
c-----Translate BCD time-of-day code
c-----Minutes

```

```

      jmin = -1
620 jmin = jmin + 1
      ibase = j60 + 60 * (jmin - 1)
      if (ibase + 2 .lt. 1) goto 620
      if (ibase + 8 .gt. ntick) goto 680
c
      call binary (3, ksyn(ibase+2), imn10)
      if (imn10 .lt. 0) goto 620
      if (imn10 .gt. 5) goto 620
c
      call binary (4, ksyn(ibase+6), imn)
      if (imn .lt. 0) goto 620
      if (imn .gt. 9) goto 620
      kmn = 10 * imn10 + imn - jmin
      if (kmn .gt. 59) goto 620
      if (kmn .lt. 0) kmn = kmn + 60
c
c-----Hours
      jmin = -1
640 jmin = jmin + 1
      ibase = j60 + 60 * (jmin - 1)
      if (ibase + 13 .lt. 1) goto 640
      if (ibase + 19 .gt. ntick) goto 680
c
      call binary (2, ksyn(ibase+13), ihr10)
      if (ihr10 .lt. 0) goto 640
      if (ihr10 .gt. 2) goto 640
c
      call binary (4, ksyn(ibase+16), ihr)
      if (ihr .lt. 0) goto 640
      if (ihr .gt. 9) goto 640
c
      khr = 10 * ihr10 + ihr
      if (khr .gt. 23) goto 640
      if (kmn + jmin .gt. 59) khr = khr - 1
      if (khr .eq. -1) khr = 23
c
c-----Day of year
      jmin = -1
660 jmin = jmin + 1
      ibase = j60 + 60 * (jmin - 1)
      if (ibase + 23 .lt. 1) goto 660
      if (ibase + 34 .gt. ntick) goto 680
c
      call binary (2, ksyn(ibase+23), idal00)
      if (idal00 .lt. 0) goto 660
      if (idal00 .gt. 3) goto 660
c
      call binary (4, ksyn(ibase+26), idal0)
      if (idal0 .lt. 0) goto 660
      if (idal0 .gt. 9) goto 660
c
      call binary (4, ksyn(ibase+31), ida)
      if (ida .lt. 0) goto 660
      if (ida .gt. 9) goto 660
c
      kda = 100 * idal00 + 10 * idal0 + ida
      if (ida .gt. 366) goto 660
      if (kmn + jmin .gt. 59 .and. khr .eq. 23) kda = kda - 1
c
c-----Translation successfull (day, hour, and minute also now valid)
      ires = 5
      return
c
c-----Translation failure, return with seconds past minute
680 return
      end
c
c-----
c-----
c-----binary : translate binary time code fields
      subroutine binary (n, kd, ires)
c
      integer kd(1)
      integer i,n,ires,ival
c
      ival = 0
      ires = -1
      do 10 i=1, n
         if (kd(i) .lt. 1) return
         if (kd(i) .gt. 2) return
         ival = ival + ival + kd(i) - 1
      10 continue
      ires = ival
c
      return
      end

```

```
c
c-----*-----*-----*-----*-----*-----*-----*-----*-----*-----*
c
c-----rdecod.inc : variables needed for decoding
c
      real*4 sec
      integer*4 ksyn
      real*4 gu
      real*4 gd
      real*4 at
      real*4 t
      integer*4 k10
      real*4 t0
      real*4 td
      integer*4 MAXSYN
      parameter (MAXSYN = 2500)

c      ksyn(MAXSYN)      binary blips
c      gu(MAXSYN)        upper constraint
c      gd(MAXSYN)        lower constraint
c      at(2*MAXSYN)      coef. norm. eqs.
c      t(MAXSYN)         right hand side
c      k10(100)
c      t0, td            calculated by ELSKEW called by IRIE and WWVB
c                        CAUTION: Due to overly clever programming, it
c                        was essential that t0, td remained a sequential
c                        pair in memory. This has been demystified in
c                        WWVB.

      common /rdecod/ ksyn(MAXSYN), gu(MAXSYN), gd(MAXSYN),
*      at(2*MAXSYN), t(MAXSYN), k10(100), t0, td
```

```

#
#-----*-----*-----*-----*-----*-----*-----*-----*-----*
#
# Generic Makefile for programs.
# Works for any mix of sun3, sun4, Fortran, and C (I hope!).
#
# "make install" fails the first time unless the executable has
# already been copied to $(BINDIR).
#
# The following is required in the user's .cshrc:
#
# setenv ARCH `/bin/arch`
# if( $ARCH == "sun3") then
#     setenv FLOAT -f68881
# else
#     setenv FLOAT ""
# endif
# setenv DIR_USR /usr/local
# setenv DIR_SRC /usr/src/local

FC      = f77
CC      = cc
FFLAGS  = -u $(FLOAT) -D$(ARCH) -Bstatic
CFLAGS  = $(FLOAT) -D$(ARCH) -Bstatic
BINDIR  = $(DIR_USR)

PROG    = ahwwvb
SRCARC  = $(PROG).src.a

FSRCS   = elskew.f \
         wwvb.f
CSRCS   = ahwwvb.c
OBJECTS = $(FSRCS:%.f=$(ARCH)/%.o) $(CSRCS:%.c=$(ARCH)/%.o)

LIBS    = $(DIR_USR)/librmf.a -lq -lm -lah
LLIBS   = $(LIBS)

INCLS   = rdecod.inc ahhead.h
OTHERS  = Makefile ahwwvb.man announce announce.rev \
         announce.rev2 $(INCLS)
OTHERBIN=

# P      = enscript -r2 -p-

$(ARCH)/$(PROG): $(OBJECTS)
    $(CC) $(CFLAGS) $(OBJECTS) $(LIBS) -o $@

$(OBJECTS): $(INCLS)

$(ARCH)/%.o: %.c
    $(CC) $(CFLAGS) -c $(@F:.o=.c) -o $@

$(ARCH)/%.o: %.f
    $(FC) $(FFLAGS) -c $(@F:.o=.f) -o $@

install: $(ARCH)/install $(ARCH)/$(PROG)
#
# /bin/rm install
#
# ln -s $(ARCH)/install install
# /bin/mv $(BINDIR)/$(PROG) $(BINDIR)/$(PROG).old
# cp $(ARCH)/$(PROG) $(BINDIR)
# chmod 755 $(BINDIR)/$(PROG)
# touch install
# touch $(ARCH)/install

# CAUTION: do not run "make clean" on more than one machine at a time;
# the archive commands ("ar") might conflict.
clean:
    /bin/rm $(ARCH)/$(PROG) $(OBJECTS) $(BINDIR)/$(PROG).old
    ar ruv $(SRCARC) $(FSRCS) $(CSRCS) $(OTHERS) $(OTHERBIN)

lint: $(FSRCS) $(CSRCS) $(INCLS)
    /usr/bin/lint -xh -u $(FSRCS) $(CSRCS) $(LLIBS) > lint

print: $(FSRCS) $(CSRCS) $(OTHERS)
#
# pr -l66 $? | $P
# enscript $?
# touch print

printall:
#
# pr -l66 $(FSRCS) $(CSRCS) $(OTHERS) | $P
# enscript $(FSRCS) $(CSRCS) $(OTHERS)
# touch print

printexport:
    pr -t $(FSRCS) $(CSRCS) $(OTHERS)

```


announce

Fri Oct 8 09:57:50 1993

- 73 -

To: dawson@iyers croker@iyers moses@andreas michael@andreas julian@andreas oneill@andreas AWalter lindh@gsvax0 iyer@gsvax0
Cc: evans@evanss
Subject: AHWWVB

My new WWVB interpreter for xdr-AH files is now ready. It has been tested at 100 sps on eight five-day-recorder examples from Arizona and one CUSP example from Pasadena. It should work well, but could require a little tweeking for your own cases. It probably has the most robust signal massaging of any WWVB reader--I spent the better part of a week looking at time codes and making that processing smarter. It uses a series of running-median filters and an edge finder. After massaging, it uses Carl Johnson's inflagrevous FORTRAN spaghetti to interpret the time code. Carl's routine determines digitizing interval and fractional seconds by an L1 fit, so the results are pretty good, though variations of 0.02 s along the trace are routine with five-day recorders.

The result is good enough to have shown just how bad current five-day recorder digitizing is--tape speed, and therefore digitizing rate, vary by as much as 40% in the worst cases (2 of the 8 tested!) producing a visible telescoping of the WWVB signal. AHWWVB tries to warn you about these problems by reporting any apparent fluctuations in WWVB second-intervals in the massaged trace.

Source code and documentation are in:
iyers:/we/itch/evans/src/proc/ahwwvb

Sun3 executable is:
evanss:/we/itch/evans/usr.sun3/ahwwvb

Sun4 executable is:
iyers:/we/itch/evans/usr.sun4/ahwwvb

AHWWVB spawns an AHCLK job to make the actual file changes, so you need that routine too. AHCLK also is used by EPICK to change file start time and digitizing rate, and is a sometimes-useful filter by itself.

Source code and documentation are in:
iyers:/we/itch/evans/src/proc/ahclk

Sun3 executable is:
evanss:/we/itch/evans/usr.sun3/ahclk

Sun4 executable is:
iyers:/we/itch/evans/usr.sun4/ahclk

To compile, you also need my RMF library (which also contains a rather good general despkicker):

Source code is in:
iyers:/we/itch/evans/src/lib/rmf

Sun3 binary is:
evanss:/we/itch/evans/usr.sun3/librmf.a

Sun4 binary is:
iyers:/we/itch/evans/usr.sun4/librmf.a

I'll put copies of the manual pages for AHWWVB in your boxes.

Good luck!
John Evans

announce.rev

Fri Oct 8 09:57:49 1993

- 74 -

To: dawson@iyers croker@iyers moses@andreas michael@andreas julian@andreas oneill@andreas AWalter lindh@gsvax0 iyer@gsvax0
Cc: evans@evanss
Subject: AHWWVB

A "-nc" (no change) option has been added to AHWWVB to suppress changes to the ah traces themselves. The default is to change the trace data.

AHWWVB now also prints (stdout) a summary of the results, as follows:

```
-----ahwwvb-----  
File "tbnh1320.xdr", code "LTB6", channel "spT":  
Start time = 1990/11/08 13:20:01.3316 UTC  
Sample interval = 0.0101375813 s (98.642859 sps)  
-----
```

Some folks prefer to collect a list of clock corrections rather than applying them to the traces. Hence the ability to get this message without changing the trace data.

Please let me know if you discover any bugs.

--John Evans

To: dawson@iwers croker@iwers michael@andreas julian@andreas oneill@andreas AWalter
Cc: evans@evanss
Subject: AHWWVB

More diddling with AHWWVB (hopefully this will be all for awhile):

There is now a "-v" flag to make it verbose (formerly the only choice). If missing, one only receives warnings and the final answer.

With -v, it will act as before but with addition of two lines of output ("AHWWVB verbose output:" and "Interpreting time series ..."), the former at the beginning and the latter when it finishes filtering and starts trying to interpret the cleaned up trace. There are also a number of minor punctuation changes for the sake of consistency. Two typical examples are appended here. Script writers take note.

Bruce ran into another bug for high sample rates. The L1 fitter sometimes went into an infinite loop. This should now be fixed and a trap installed just in case.

Please let me know if you discover any other bugs.

--John Evans

```
iwers{evans}:l11==> ahwwvb -m 80 tbnhl320.xdr -vb LTB6 spT -v
AHWWVB verbose output:
Longest RMF window is of 15 points.
Compensating for low-pass filtered WWVB.
Running RMF prefilter of 3 points ...
Running RMF prefilter of 5 points ...
Running main RMF filter of 15 points ...
Interpreting time series ...
```

Second intervals range from 0.993 to 1.004 s.
5th and 95th percentiles differ by 0.010 s.

Defaulting year to 1990.

```
Changing start time and interval ...
ahclk -l tbnhl320.xdr.a28305 -s 1990 11 08 13 20 1.331641 -i 0.0101375813
... successful.
```

```
-----ahwwvb-----
File "tbnhl320.xdr", code "LTB6", channel "spT":
Start time = 1990/11/08 13:20:01.3316 UTC
Sample interval = 0.0101375813 s (98.642859 sps)
-----
```

```
iwers{evans}:l12==> ahwwvb -m 990 06.29.0192.4.ah -vb 0192 4 -nc
```

```
-----ahwwvb-----
File "06.29.0192.4.ah", code "0192", channel "4":
Start time = 1991/04/10 14:06:29.2087 UTC
Sample interval = 0.0009999890 s (1000.011012 sps)
-----
```

```
iwers{evans}:l13==>
```

BUG REPORT for AHWWVB.

The lags given with option "-d" were applied with reversed sign in versions prior to 23 Oct 92. If you have source code, change one line of ahwwvb.c as follows:

```
from:  new_st -= lagg / 1000.0;          /* Remove any delay */
to:    new_st += (lagg / 1000.0);        /* Remove any delay */
```

Otherwise, get binaries from:

```
Sun4    iyers:/we/itch/evans/usr.sun4
Sun3    evanss:/we/itch/evans/usr.sun3
```

Sorry about this, folks.

--John

13. Appendix G: Manual Page and Source Code for *ahclk*

The program *ahclk* is a C program called by *ahwwwb* to actually make the changes to start time and sample rate in an *xdr-ah* seismogram's header. Program *ahclk* is a creation of the first author. Both the source code and the manual page are available via anonymous *ftp* from "andreas.wr.usgs.gov" in directory "~ftp/pub/outgoing/evans/wwwb". This Open-File Report serves to release them to the public, subject to the limitations cited on the title page.

SYNOPSIS

ahclk -l *list_file* [-s *year mo dy hr mn sec*] [-i *new_interval*]

DESCRIPTION

Changes starting time and/or digitizing interval in the header of one or more traces listed in *list_file* (same format as *list_file* used by *epick*). Copies through, unchanged, all traces encountered that are NOT listed in *list_file*.

ahclk is intended primarily for correcting start time and digitizing interval of data from "five-day recorders" and other instruments that record a time code in parallel with seismic data, but which may give inaccurate start time or digitizing interval due to inaccurate internal clocks or analog-instrumentation foibles. Hence, it assumes that all traces in *list_file* are *exactly the same* time interval as all the other traces listed. You will get bizarre results if this is not so.

ahclk can be used to correct for clock drift and similar errors that apply individually or to a subset of traces in the AH file by listing just the appropriate trace(s) in *list_file*.

ARGUMENTS

-l *list_file* must contain one or more AH file names and trace sequence-numbers.

OPTIONS

-s *year mo dy hr mn sec* is the time of the first sample in the file (e.g. "-s 1990 10 20 11 04 20.071"). Replaces "ahead->record->abstime" in the *ah* header. These six values *must* all be positive.

-i *new_interval* is the new sampling interval (seconds). Replaces "ahead->record->delta" in the *ah* header. *new_interval* must be positive.

SEE ALSO

epick—X11 *ah* picking routine derived and extended from Lamont's *sunpick*. *epick* calls *ahclk* to invoke changes in trace timing derived from "alternate time" picks (i.e., of time codes recorded in parallel with seismograms). *ahclk* may be used as a stand-alone filter in the normal way as well.

Other *ah* filters.

BUGS

list_file must be sorted with all the traces for any particular file together and in ascending numerical order within each file. *mkahlist* does that for you. Someday, *ahclk* could be made smarter about this, but it hasn't been yet. *ahclk* does verify this ordering, however.

Should get smart enough to check relatedness of traces being changed.

AUTHOR

John R. Evans, USGS, Menlo Park, CA. Original written September 25, 1990.

```

/*
 * AH filter AHCLK (AH Clock change):
 */

#include <stdio.h>
#include <rpc/rpc.h>
#include <local/ahhead.h>

#define NM_LEN 128      /* Maximum file-name length      */
#define STR_LEN 256     /* Maximum scratch-string length    */
#define MX_LST 256     /* Maximum line count of listfile   */

#define strneq !strcmp
#define streq  !strcmp

    struct ah_time NULL_AH_TIME; /* NULL starting time      */
    void error(); /* -lq (error.c)           */
    void report(); /* -lq (error.c)          */
    FILE *efopen(); /* -lq (efopen.c)        */

char *progname;

main(argc, argv)
    int argc;
    char **argv;
{
    FILE *list_file; /* listfile a la "epick" */
    char list_name[NM_LEN];
    void filter(); /* Defined below */
    struct ah_time new_st; /* New starting time */
    double new_int; /* New sampling interval */
    void mod_files(); /* Defined below */
    int seq_ok(); /* Defined below */

    /* Initialize */
    (void)bzero((char *)(&NULL_AH_TIME), sizeof(NULL_AH_TIME));

    new_st = NULL_AH_TIME; /* No-change flag */
    new_int = -1.; /* No-change flag */

    list_name[0] = '\0';
    list_file = (FILE *) NULL;

    /* Check usage */
    if (argc < 5)
        error("Usage: %s -l listfile [-s year mo dy hr mn sec] [-i new_interval]",
            argv[0]);

    progname = argv[0];

    /* Interpret command line arguments */
    for (--argc, argv++; argc; --argc, argv++) {
        /* listfile name */
        if (strneq(*argv, "-l", 2)) {
            --argc, argv++;
            (void)strcpy(list_name, *argv);
        }

        /* new start time */
        else if (strneq(*argv, "-s", 2)) {
            --argc, argv++;
            if (argc < 6)
                report ("Bad start time");
            new_st.yr = atoi(*argv); --argc, argv++;
            new_st.mo = atoi(*argv); --argc, argv++;
            new_st.day = atoi(*argv); --argc, argv++;
            new_st.hr = atoi(*argv); --argc, argv++;
            new_st.mn = atoi(*argv); --argc, argv++;
            if (sscanf(*argv, "%f", &(new_st.sec)) != 1)
                report ("Bad start time");
        }

        /* new sampling interval */
        else if (strneq(*argv, "-i", 2)) {
            --argc, argv++;
            if ((sscanf(*argv, "%lf", &new_int) != 1) ||
                (new_int <= 0.))
                report ("Bad sampling interval");
        }
        else
            error("Illegal option \"%s\"", *argv);
    }

    /* Open seismogram listfile */
    if(list_name[0] == '\0')
        error("Need listfile name");
    list_file = fopen(list_name, "r");

```

```

/* Check sequence in listfile */
rewind (list_file);
if (!seq_ok(list_file))
    error("listfile out of sequence");

/* Process the named files */
rewind (list_file);
(void)mod_files(list_file, new_st, new_int);

/* Exit */
(void)fclose(list_file);
exit(0);
}

/*
 * Modify all files in list_file; copy all others unchanged.
 */

void
mod_files(list_file, new_st, new_int)
FILE *list_file; /* listfile a la "epick" */
struct ah_time new_st; /* New starting time */
double new_int; /* New sampling interval */
{
    void close_out(); /* Defined below */
    int copy_head(); /* Defined below */
    ahhed tmp_head; /* Space to return ah header */
    void copy_trace(); /* Defined below */

    char fname_in[NM_LEN]; /* Input AH-file name */
    FILE *ah_fp_in;
    XDR xdr_in;

    char fname_out[NM_LEN]; /* Temporary output file name */
    FILE *ah_fp_out;
    XDR xdr_out;

    int ugid;
    int rec_num;
    int cur_rec;
    char cur_file[NM_LEN];
    char in_str[STR_LEN]; /* One line from list file */

    /* Read list file, processing AH files as it goes */
    ah_fp_in = NULL;
    ah_fp_out = NULL;
    cur_file[0] = '\0';
    cur_rec = 0;

    while (fgets(in_str, sizeof(in_str), list_file) != NULL) {
        if (sscanf(in_str, "%s%d%d", fname_in, &rec_num, &ugid) < 2)
            error("Bad listfile format.");

        /* Open AH file if not already open */
        if (!streql(cur_file, fname_in)) {

            /* If input file is already open, copy through
             any remaining traces, then close the output
             file and overwrite fname_in with it */
            if (ah_fp_in != NULL)
                close_out(fname_in, ah_fp_in, &xdr_in,
                    fname_out, ah_fp_out, &xdr_out);

            /* Open AH file, attach to XDR stream */

            (void)sprintf(fname_out, "%s.XXXXXX", fname_in);
            (void)mktmp(fname_out);

            ah_fp_in = fopen(fname_in, "r");
            ah_fp_out = fopen(fname_out, "w");
            xdrstdio_create(&xdr_in, ah_fp_in, XDR_DECODE);
            xdrstdio_create(&xdr_out, ah_fp_out, XDR_ENCODE);

            cur_rec = 1;
            (void)strcpy(cur_file, fname_in);
        }

        /* Copy through any unaffected traces prior to target trace */
        while (cur_rec < rec_num) {
            (void)copy_head(&tmp_head, &xdr_in, &xdr_out,
                NULL_AH_TIME, -1.);
            (void)copy_trace(&xdr_in, &xdr_out, tmp_head);
            cur_rec++;
        }

        /* Modify and copy affected traces */
        if (cur_rec == rec_num) {

```



```

        (void)copy_head(&tmp_head, &xdr_in, &xdr_out,
                        new_st, new_int);
        (void)copy_trace(&xdr_in, &xdr_out, tmp_head);
        cur_rec++;
    }
    else
        error("This should never happen--see Evans");
}

/* Close out last file pair */
close_out(fname_in, ah_fp_in, &xdr_in, fname_out, ah_fp_out, &xdr_out);
}

/*
 * Close out a file pair, overwriting input file with output file.
 */

void
close_out(fname_in, ah_fp_in, xdr_in, fname_out, ah_fp_out, xdr_out)
    char    fname_in[]; /* Input AH-file name */
    FILE    *ah_fp_in;
    XDR      *xdr_in;

    char    fname_out[]; /* Temporary output file name */
    FILE    *ah_fp_out;
    XDR      *xdr_out;
{
    int      copy_head(); /* Defined below */
    ahhed    tmp_head; /* Space to return ah header */
    char      sys_str[STR_LEN]; /* "system" call argument */

    /* Copy through any unaffected traces remaining in file */
    while (copy_head(&tmp_head, xdr_in, xdr_out, NULL_AH_TIME, -1.))
        (void)copy_trace(xdr_in, xdr_out, tmp_head);

    /* Close files */
    xdr_destroy(xdr_in);
    xdr_destroy(xdr_out);
    (void)fclose(ah_fp_in);
    (void)fclose(ah_fp_out);

    /* Overwrite input file with output file */
    (void)sprintf(sys_str, "/bin/mv -f %s %s",
                  fname_out, fname_in);
    if ((system(sys_str) & 0377) == 127)
        report("File overwrite (mv) failed!");
}

/*
 * Copy through an AH trace, changing start time and sampling interval
 * as requested. (new_st == NULL_AH_TIME) prevents change of starting
 * time. (new_int <= 0.) prevents change of sampling interval.
 * Hence "copy_head(hd, xdr_in, xdr_out, NULL_AH_TIME, -1.)" copies
 * header unchanged.
 *
 * Returns copied header in "hd". Returns TRUE after successful copy,
 * or FALSE if header could not be read properly (e.g., at end-of-file).
 */

int
copy_head(hd, xdr_in, xdr_out, new_st, new_int)
    ahhed    *hd; /* Space to return ah header */
    XDR      *xdr_in;
    XDR      *xdr_out;
    struct ah_time new_st; /* New starting time */
    double    new_int; /* New sampling interval */
{
    void      log_ahclk(); /* Defined below */

    /* Read header */
    if (xdr_gethead(hd, xdr_in) != 1)
        return (FALSE);

    /* Log any changes that will be made */
    (void)log_ahclk(hd, new_st, new_int);

    /* Change start time */
    if (!strcmp((char *)&new_st, (char *)&NULL_AH_TIME),
        sizeof(new_st))
        hd->record.abstime = new_st;

    /* Change sampling interval */
    if (new_int > 0.)
        hd->record.delta = new_int;

    /* Write header */
    if (xdr_puthead(hd, xdr_out) != 1)
        error("Error writing header.");
}

```

```

    return (TRUE);
}

/*
 * Log any changes made by ahclk.
 */

void
log_ahclk(hd, new_st, new_int)
    ahhed      *hd;
    struct ah_time new_st;      /* New starting time          */
    double      new_int;        /* New sampling interval    */
{
    char        log_str[STR_LEN]; /* Log message written to ahead */
    void        log_app();        /* Defined below            */

    if (!strneq((char *)(&new_st), (char *)(&NULL_AH_TIME),
                sizeof(new_st))) {
        (void)sprintf(log_str,
            "%s: abstime was %d %d %d %d %d %f;", progname,
            hd->record.abstime.yr, hd->record.abstime.mo,
            hd->record.abstime.day, hd->record.abstime.hr,
            hd->record.abstime.mn, hd->record.abstime.sec);
        (void)log_app(hd, log_str);
    }

    if (new_int > 0.) {
        (void)sprintf(log_str,
            "%s: delta was %f;", progname, hd->record.delta);
        (void)log_app(hd, log_str);
    }
}

/*
 * Append log_str to log message in ahead "hd".
 */

#define MIN(a,b) (((a)<(b)) ? (a) : (b)) /* Smaller value */

void
log_app(hd, log_str)
    ahhed      *hd;
    char        log_str[];
{
    int         strl;
    void        ch_app();        /* Defined below            */

    /* Find end of non-blank part of log */
    for (strl = MIN(LOGSIZE - 1, strlen(hd->record.log)) ;
        (strl >= 0) && (hd->record.log[strl] == ' '); --strl);

    /* If find "null" at start of string, overwrite it */
    if (strl == 4 && strneq(hd->record.log, "null", 4))
        strl = 0;

    /* Append new log message */
    if (strl != 0)
        (void)ch_app(&strl, hd, " ");
    (void)ch_app(&strl, hd, log_str);
}

/*
 * Append characters to log until full.
 */

void
ch_app(strl, hd, log_str)
    int         *strl;
    ahhed      *hd;
    char        log_str[];
{
    int         i;
    int         l_len;

    i = 0;
    l_len = strlen(log_str);
    while((*strl < (LOGSIZE - 1)) && (i < l_len))
        hd->record.log[(*strl)++] = log_str[i++];
    hd->record.log[*strl] = '\0';
}

/*
 * Copy through an AH trace, unchanged.
 */

void
copy_trace(xdr_in, xdr_out, hd)

```

```

XDR          *xdr_in;
XDR          *xdr_out;
ahhed        hd;

{
    float      *darray;

    /* Allocate trace storage */
    darray = (float *)mkdatspace(&hd);

    /* Read trace */
    if (xdr_getdata(&hd, (char *)darray, xdr_in) == -1)
        error("Error reading data record");

    /* Write trace */
    if (xdr_putdata(&hd, (char *)darray, xdr_out) == -1)
        error("Error reading data record");

    /* Deallocate trace storage */
    free ((char *)darray);
}

/*
 * Check sequence in listfile
 */

typedef struct {
    char      fname_in[NM_LEN];
    int       rec_num;
    int       ugid;
} TLIST;

int
seq_ok(list_file)
    FILE      *list_file;
{
    char      in_str[STR_LEN];          /* One line from list file */

    int       i;
    int       same_nm;
    int       seq;
    int       rv;

    int       nlines;
    TLIST      *file_list[MX_LST];      /* known AH-file names */

    rv = TRUE;
    nlines = 0;
    file_list[0] = (TLIST *)emalloc(sizeof(TLIST));

    while (fgets(in_str, sizeof(in_str), list_file) != NULL) {
        if (sscanf(in_str, "%s%d%d",
            file_list[nlines]->fname_in,
            &(file_list[nlines]->rec_num),
            &(file_list[nlines]->ugid))
            >= 2) {

            same_nm = TRUE;
            seq = file_list[nlines]->rec_num;
            for(i = nlines; i >= 0; --i) {
                if (strcmp(file_list[nlines]->fname_in,
                    file_list[i]->fname_in)) {
                    if (!same_nm)
                        rv = FALSE;
                    if (file_list[i]->rec_num >= seq
                        && i < nlines)
                        rv = FALSE;
                    else
                        seq = file_list[i]->rec_num;
                }
            }
            else {
                same_nm = FALSE;
            }
        }

        nlines++;
        if (nlines >= MX_LST)
            error("listfile too long");
        file_list[nlines] = (TLIST *)emalloc(sizeof(TLIST));
    }
    else
        error("Bad listfile format.");
}

for (i = 0; i <= nlines; i++)
    free((char *)file_list[i]);
return rv;
}

```

```

/*      structure for data file header  --      witte, 11 June 85      */

#define AHHEADSIZE 1024
#define CODESIZE 6
#define CHANSIZE 6
#define STYPESIZE 8
#define COMSIZE 80
#define TYPEMIN 1
#define TYPEMAX 6
#define LOGSIZE 202
#define LOGENT 10
#define NEXTRAS 21
#define NOCALPTS 30

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

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

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

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

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

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

struct station_info {
    char      code[CODESIZE]; /* station code      */
    char      chan[CHANSIZE]; /* lpz,spn, etc.     */
    char      stype[STYPESIZE]; /* wwssn,hqlp,etc.  */
    float     slat;        /* station latitude  */
    float     slon;        /* " longitude      */
    float     elev;        /* " elevation      */
    float     DS;          /* gain             */
    float     AO;          /* normalization    */
    struct calib cal[NOCALPTS]; /* calibration info */
};

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

struct record_info {
    short     type;        /* data type (int,float,...) */
    long      ndata;       /* number of samples      */
    float     delta;       /* sampling interval      */
    float     maxamp;      /* maximum amplitude of record */
    struct ah_time abstime; /* start time of record section */
    float     rmin;        /* minimum value of abscissa */
    char      rcomment[COMSIZE]; /* comment line */
    char      log[LOGSIZE]; /* log of data manipulations */
};

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

```

```
#define FLOAT 1
#define COMPLEX 2
#define VECTOR 3
#define TENSOR 4
#define DOUBLE 6
#define INTEGER 7
```

```
#
#-----*
#
# Make file for AHCLK.

CFLAGS= $(FLOAT) -g -D$(ARCH)

SRCS=  ahclk.c

OBJS=  $(SRCS:%.c=$(ARCH)/%.o)

LIBS=  -lah -lq

BIN = $(DIR_USR)

prog:  $(ARCH)/ahclk

$(ARCH)/ahclk: $(OBJS)
        cc -o $@ $(CFLAGS) $(OBJS) $(LIBS)

$(ARCH)/%.o: %.c
        cc $(CFLAGS) -c $(F:.o=.c) -o $@

install: $(ARCH)/ahclk
        cp $(ARCH)/ahclk $(BIN)

lint: $(SRCS)
        /usr/bin/lint -xh -u $(SRCS) $(LIBS) > lint

clean:
        -@rm $(ARCH)/*.o $(ARCH)/ahclk
        ci -l -sExp -f \
            -m"USGS Computer Program ahclk; Version 1.0; John R. Evans" \
            $(SRCS) Makefile test_ah.in test_ah.out
```

In order to use the makefile in this directory, the following MUST be in your .cshrc file

#start here

```
setenv ARCH `/bin/arch`
if( $ARCH == "sun3") then
    setenv FLOAT -f68881
else
    setenv FLOAT ""
endif
```

#end here

14. Appendix H: Manual Pages and Source Code for Portions of *librmf* and *libq*

The library *librmf* contains C and FORTRAN programs for running-median filters and automated despikers based upon them. The routine *median* from this library is used by *ahwwwvb* to filter the WWVB trace, and is listed here. Several routines from Bruce Julian's library *libq* are used by *ahwwwvb* and *ahclk*, and are listed here. The source code for these routines is available *via* anonymous *ftp* from "andreas.wr.usgs.gov" in directory "~ftp/pub/outgoing/evans/library_stuff". This Open-File Report serves to release them to the public, subject to the limitations cited on the title page.

A descriptive writeup for the median filter is in file "median.writeup" of that *ftp* directory. Manual pages for the date and time routines of *libq* are included, and listed here. The routines *efopen*, *emalloc*, and *erealloc* act just like their standard C counterparts, *fopen*, *malloc*, and *realloc*, but performing error reporting. Manual pages are unavailable for the other routines.

NAME

jdn, date, yrday, mnday, isleap, ckdate – calendar routines

SYNTAX

```
#include <local/date_time.h>
#include <local/qerr.h>

long jdn(yr, mo, da, caltyp)
    int yr, mo, da, caltyp;

void date(n, pyr, pmo, pda, caltyp)
    long n;
    int *pyr, *pmo, *pda, caltyp;

int yrday(mo, da, leap)
    int mo, da, leap;

void mnday(d, leap, pmo, pda)
    int d, leap, *pmo, *pda;

int isleap(yr, caltyp)
    int yr, caltyp;

int ckdate(yr, mo, da, caltyp)
    int yr, mo, da, caltyp;
```

DESCRIPTION

These routines perform computations about the calendar.

Jdn returns the Julian Day Number of the day beginning at noon of the calendar date specified by the arguments *yr*, *mo*, and *da*. **Date** performs the inverse computation, decoding the Julian Day Number *n* and storing the results indirectly, through its pointer arguments *pyr*, *pmo*, and *pda*. In both routines, the argument *caltyp* specifies which calendar is to be used: **GREGORIAN** (modern) or **JULIAN** (old-style). These calendar-type codes are defined in the include file **date_time.h**.

Yrday returns the day of the year corresponding to day number *da* of month number *mo*. **Mnday** performs the inverse computation, determining the month and day of the month corresponding to the *n*th day of the year, and storing them through the pointer arguments *pmo* and *pda*. In both functions the boolean argument *leap* specifies whether the year in question is a leap year.

Isleap returns a boolean value telling whether year *yr* is a leap year in the calendar of type *caltyp*. It is suitable for use as the argument *leap* to routines **yrday** and **mnday**.

Ckdate returns the value 0 if the date specified by arguments *yr*, *mo*, and *da* is legal in the calendar of type *caltyp*, and otherwise one of the values **EBADDAY**, **EBADMONTH**, or **EBADYEAR**, defined in the file **qerr.h**.

NOTES

Do not confuse the Julian Day Number, which counts days since 1 January -4713 of the old-style calendar, with the day of the year, which is sometimes incorrectly called the “Julian day”.

SEE ALSO

date_time(3Q)

A list of the dates when various countries adopted the Gregorian calendar is given in the “Explanatory Supplement to The Astronomical Ephemeris and The American Ephemeris and Nautical Almanac”, published periodically by Her Majesty’s Stationary Office.

EXAMPLES

The following fragment of code...

```
#include <local/date_time.h>
```

AUTHOR

Bruce R. Julian, USGS Menlo Park, Calif.

NAME**timvar, datetime** – time routines**SYNTAX****#include <local/date_time.h>****TIME timvar(yr, mo, da, hr, mn, sec, caltyp)****int yr, mo, da, hr, mn;****double sec;****int caltyp;****datetime(t, pyr, pmo, pda, phr, pmn, psec, caltyp)****TIME t;****int *pyr, *pmo, *pda, *phr, *pmn;****double *psec;****int caltyp;****DESCRIPTION**

These routines convert conventional dates and times-of-day to **TIME** (double precision) variables, a form that is computationally efficient and has enormous dynamic range (2.9×10^{-39} seconds to 10^6 years) and resolution (one part in 10^{17}). This form is more than adequate for all purposes, ranging from geologic to picosecond scales. Using ordinary language-defined arithmetic operators, **TIME** variables may be subtracted from one another to produce time intervals, time intervals may be added and subtracted from each other and multiplied or divided by dimensionless values, and time intervals may be added to or subtracted from times to produce new times.

Timvar converts a calendar date and time of day into a **TIME** variable. **Datetime** performs the inverse transformation, decoding the **TIME** variable *t* and storing the results indirectly, through its pointer arguments. *Caltyp* specifies which calendar is to be used: **GREGORIAN** (modern) or **JULIAN** (old-style).

The C-language include file **date_time.h** defines the type **TIME**, the calendar-type codes **GREGORIAN** and **JULIAN**, and the constant time intervals **YEAR**, **LEAPYEAR**, **WEEK**, **DAY**, **HOURL**, **MINUTE**, **SECOND**, **MILLISEC**, **MICROSEC**, and **NANOSEC**, as well as a few things useful with the **calendar(3Q)** routines.

BUGS

No allowance is made for “leap seconds”, which are periodically inserted to keep the earth and the clock synchronized for the convenience of navigators.

SEE ALSO**calendar(3Q)**

A list of the dates when various countries adopted the Gregorian calendar is given in the “Explanatory Supplement to The Astronomical Ephemeris and The American Ephemeris and Nautical Almanac”, published periodically by Her Majesty’s Stationary Office.

EXAMPLES

The following fragment of code finds the number of fortnights between two epochs:

#include <local/date_time.h>**double fort;****fort = (timvar(1983, 5, 31, 14, 50, 12.34, GREGORIAN) -****timvar(1950, 1, 1, 0, 0, 0.0, GREGORIAN))/(14*DAY);****AUTHOR**

Bruce R. Julian, USGS Menlo Park, Calif.

```

#ifndef lint
static char rcsid[]="$Header: calendar.c,v 1.1 90/11/25 12:45:10 julian Exp S";
#endif
/*
 * Calendar computations (old and new-style)
 * Bruce R. Julian, USGS Menlo Park, CA
 */
#include <local/local.h>
#include <local/date_time.h>
#include <local/qerr.h>

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

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

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

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

/* Check calendar date for legality */
int
ckdate(y, m, d, cal)
    int y, m, d; /* Year, month, date */
    int cal; /* Calendar type */
{
    int *et = eom[isleap(y, cal)];

    if (y == 0) return EBADYEAR;
    if (m < 1 || m > 12) return EBADMONTH;
    if (d < 1 || d > et[m]-et[m-1]) return EBADDAY;
    return 0;
}

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

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

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

/* Compute Julian Day Number from calendar date */
long
jdn(yr, mo, day, cal)
    int yr, mo, day; /* Year, month, date */
    int cal; /* Calendar type */
{
    if (yr < 0)
        yr++;

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

```

```

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

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

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

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

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

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

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

```

```
#ifndef lint
static char rcsid[]="$Header: efopen.c,v 1.2 88/10/17 11:15:44 julian Exp $";
#endif
/*
 * Open file, die if can't
 * Bruce R. Julian, USGS Menlo Park, CA
 */
#include <stdio.h>

void          error();

FILE*
efopen(file, mode)
    char      *file, *mode;
{
    FILE      *fp;

    if ((fp = fopen(file, mode)) == NULL)
        error("Can't open file \"%s\", mode %s", file, mode);
    return fp;
}
```

```
#ifndef lint
static char rcsid[]="$Header: emalloc.c,v 1.2 88/10/17 11:15:48 julian Exp $";
#endif
/*
 * Allocate memory, die if can't
 * Bruce R. Julian, USGS Menlo Park, CA
 */
#define NULL 0
#define u_int unsigned

char *calloc();           /* Unix C library */
char *malloc();           /* Unix C library */
void error();             /* libq.a */
char *realloc();          /* Unix C library */

char*
ecalloc(n, s)
    u_int n, s;
{
    char *p;

    if ((p = calloc(n, s)) == NULL)
        error("Can't allocate %u X %u bytes");
    return p;
}

char*
emalloc(s)
    u_int s;
{
    char *p;

    if ((p = malloc(s)) == NULL)
        error("Can't allocate %u bytes");
    return p;
}

char*
erealloc(p, s)
    char *p;
    u_int s;
{
    if ((p = realloc(p, s)) == NULL)
        error("Can't reallocate %u bytes");
    return p;
}
```

```
#ifndef lint
static char rcsid[]="$Header: eopen.c,v 1.2 88/10/17 11:15:50 julian Exp $";
#endif
/*
 * Open file, die if can't
 */

void    error();

int
eopen(file, mode)
    register char    *file;
    register int     mode;
{
    register int     f;

    if ((f = open(file, mode)) < 0)
        error("Can't open file \"%s\", mode %d", file, mode);
    return f;
}
```



```

#ifndef lint
static char rcsid[]="$Header: error.c,v 1.4 90/04/08 12:32:22 julian Exp $";
#endif
/*
 * Report error, optionally die
 * Bruce R. Julian, USGS Menlo Park, Calif. 14 Dec 1983
 */
#include <stdio.h>
#include <local/stdtyp.h>
#include <varargs.h>

extern int sys_nerr;
extern int errbase, nerr;
extern char *sys_errlist[], *q_errlist[];
extern int errno;
extern char *progname;

void exit();
char *sprintf();

/* Return error message */
char
*errmsg(ier)
    int ier;
{
    static char s[12];

    if (ier == 0)
        return (NULL);
    if (ier > 0 && ier < sys_nerr)
        return (sys_errlist[ier]);
    ier -= errbase;
    if (ier > 0 && ier < nerr)
        return (q_errlist[ier]);
    else
        return (sprintf(s, "Error %d", errbase+ier));
}

/* Print error message */
/*VARARGS*/
static void
msg(args)
    va_list args;
{
    char *fmt; /* Format string */
    char *p;

    if (progname)
        (void)fprintf(stderr, "%s: ", progname);
    fmt = va_arg(args, char*);
    _doprnt(fmt, args, stderr);
    if ((p = errmsg(errno)) != NULL)
        (void)fprintf(stderr, " (%s)", p);
    (void)fprintf(stderr, "\n");
}

/* Print error message but don't die */
/*VARARGS*/
void
report(va_alist)
    va_dcl
{
    va_list args;

    va_start(args);
    msg(args);
    va_end(args);
}

/* Print error message and die */
/*VARARGS*/
void
error(va_alist)
    va_dcl
{
    va_list args;

    va_start(args);
    msg(args);
    va_end(args);
    exit(1);
}

```

```

/*
*-----*-----*-----*-----*-----*-----*-----*-----*-----*
C-callable version of routine "median", an odd-length running
median filter for floating-point data.

Arrays are zero-offset, standard C arrays:
    arrin[0..arrlen-1]
    arrout[0..arrlen-1]

This is a recoding, rather than an interfacing routine to the
FORTRAN version because the latter proved impractical due to I/O
system interference between C and FORTRAN (and because these
difficulties are likely to rise anew each time a new compiler is
introduced even if I do circumvent them now).

FORTRAN version's "ln" here is called "arrlen".
FORTRAN version's "m" here is called "winlen".
FORTRAN version's "n" here is called "hww".
*/

#include <stdio.h>
#define TRUE 1
#define FALSE 0

void
median(arrin, arrout, arrlen, winlen, iprint)

    float arrin[], arrout[];
    int arrlen, winlen, iprint;
{
    float *sort, sorttemp;
    int *subs, substemp;
    int hww, iold, inew, jj, iout, i, key;
    int inok, ijunk;
    void mederrf();

    if(iprint) {
        fprintf(stdout, "\n Starting MEDIAN. Window length=%3i.", winlen);
        fprintf(stdout, " Array length=%10i.\n", arrlen);
    }

    hww = (winlen - 1) / 2; /* "Half-window width", which in a zero-offset
                           array also points to middle-point of window */

    /* Test for errors */
    inok = TRUE;

    if (winlen < 3) {
        fprintf(stderr, " Window length too small. Winlen=%5i\n", winlen);
        mederrf(&inok);
    }

    if (winlen > arrlen) {
        fprintf(stderr, " Window length greater than trace length.\n");
        mederrf(&inok);
    }

    /* Allocate window storage */
    sort = (float *)malloc(winlen * sizeof(float));
    subs = (int *)malloc(winlen * sizeof(int));

    if (((2 * hww) + 1) != winlen) {
        fprintf(stderr, " Window must be odd length. Winlen=%5i\n", winlen);
        mederrf(&inok);
    }

    /* Proceed if input parameters are OK */
    if (inok) {

        /* "Copy-on" ends of arrin where RMF is undefined */
        for(jj = 0 ; jj < hww ; jj++) {
            arrout[jj] = arrin[jj];
        }
        for(jj = arrlen - hww ; jj < arrlen ; jj++) {
            arrout[jj] = arrin[jj];
        }

        /* Fill sorting array for first time and bubble sort */
        for(i = 0 ; i < winlen ; i++) {
            subs[i] = i;
            sort[i] = arrin[i];
        }

        key = TRUE;
        while(key) { /* Ascending-order bubble sort */
            key = FALSE;
            for(i = 1 ; i < winlen ; i++) {
                if (sort[i - 1] > sort[i]) {

```

```

        key = TRUE;
        sorttemp = sort[i - 1];
        sort[i - 1] = sort[i];
        sort[i] = sorttemp;
        substemp = subs[i - 1];
        subs[i - 1] = subs[i];
        subs[i] = substemp;
    }
}

/* Put result (middle point of "sort") into "arout" */
arout[hww] = sort[hww];

/* Thereafter, drop in only the new point from "arrin" to save sorting time */
for(iout = hww + 1; iout < arrlen - hww; iout++) {
    iold = iout - (hww + 1); /* points to oldest element still in sort window */
    inew = iout + hww;       /* points to element to be added to sort window */

    for(i = 0; subs[i] != iold; i++); /* find old point in sort array */

    sort[i] = arrin[inew]; /* replace old point in sort array */
    subs[i] = inew;

    while(i < winlen - 1) { /* move point up list */
        if(sort[i] > sort[i + 1]) {
            sorttemp = sort[i];
            sort[i] = sort[i + 1];
            sort[i + 1] = sorttemp;
            substemp = subs[i];
            subs[i] = subs[i + 1];
            subs[i + 1] = substemp;
            i++;
        }
        else {
            break;
        }
    }

    while(i > 0) { /* move point down list */
        if(sort[i - 1] > sort[i]) {
            sorttemp = sort[i - 1];
            sort[i - 1] = sort[i];
            sort[i] = sorttemp;
            substemp = subs[i - 1];
            subs[i - 1] = subs[i];
            subs[i] = substemp;
            i--;
        }
        else {
            break;
        }
    }

    /* Put result (middle point of "sort") into "arout" */
    arout[iout] = sort[hww];
}

if(iprint) {
    fprintf(stdout, " Return from MEDIAN.\n\n");
}

/* Clean up */
free((char *)sort);
free((char *)subs);
}

/* Error message and flag for immediate exit */

void
mederrf(inok)

    int *inok;

{
    fprintf(stderr, " -----Error in subroutine MEDIAN-----\n");
    fprintf(stderr, "          (Time series returned unfiltered.)\n");
    *inok = FALSE;
}

```

```
#ifndef lint
static char rcsid[]="$Header: time.c,v 1.2 88/10/17 13:56:48 julian Exp $";
#endif
/*
 * Computations based on date and time of day
 * Bruce R. Julian, USGS Menlo Park, CA
 */
#include <math.h>
#include <local/local.h>
#include <local/date_time.h>

/* Convert date and time to TIME variable */
TIME
timvar(yr, mo, da, hr, mn, sec, caltyp)
    int    yr, mo, da, hr, mn;
    double sec;
    int    caltyp;
{
    return (double)DAY*(jdn(yr, mo, da, caltyp)-BASEJDN) +
        HOUR*hr + MINUTE*mn + SECOND*sec;
}

/* Convert TIME variable to date and time */
void
datetime(t, pyr, pmo, pda, phr, pmn, psec, caltyp)
    TIME    t;
    int     *pyr, *pmo, *pda, *phr, *pmn;
    double  *psec;
    int     caltyp;
{
    long    d;

    d = floor(t/DAY);
    date(d+BASEJDN, pyr, pmo, pda, caltyp);
    t -= d*DAY;
    *phr = t/HOUR;
    t -= (*phr)*HOUR;
    *pmn = t/MINUTE;
    *psec = t - (*pmn)*MINUTE;
}
```

```
#define LAMLAM 0
#define LAMMU 1
#define MUMU 2
#define LAMRHO 3
#define MURHO 4
#define RHORHO 5

typedef struct {
    float    vp, vs;           /* Average wave speeds          */
    float    rho;              /* Average density             */
    float    s2[6];            /* Zero-lag covariances       */
    float    a[6][3];          /* Correlation distances       */
    double    (*bfunct)();      /* 3-D Fourier transform function */
} SCATMDL;

double    bexp(), bgauss();
void      born_pp(), born_ps(), born_sp(), born_ss();
```

```

/*          C M D _ O P T . H
 * Include file for command-line argument processing
 * Bruce R. Julian, USGS Menlo Park, Calif.
 */
#ifndef CMDOPT_H
#define CMDOPT_H
#include <local/stdtyp.h>

#ifndef DIM
#define DIM(x) (sizeof(x)/sizeof((x)[0]))      /* Dimension of array */
#endif DIM

struct command {
    int  (*c_func)();      /* Function to be called for option */
    char *c_name;          /* Name of option in argument list */
    char *c_args;          /* Args (printed for documentation) */
    char *c_comment;       /* Printed out for documentation */
};

char      *aarg();
struct command *findname();
int      isnarg();
double    narg();
char      *peekarg();
void      prt_help();
void      prt_doc();

int      eargc;           /* Copy of argc */
char     **eargv;         /* Copy of argv */
int      iarg;            /* Argument index */
int      ncmd;            /* No. of elements in cmd[] */
char     *progname;       /* For error messages */

#define PROCESS_OPTS { \
    progname = argv[0]; \
    eargc    = argc; \
    eargv    = argv; \
    ncmd     = DIM(cmd); \
    for (iarg=1; iarg<eargc && eargv[iarg][0]!='-'; iarg++) \
        XEQCMD(eargv[iarg]); \
}

#define XEQCMD(s)      (*findname(s)->c_func)()
#endif CMDOPT_H

```

```
/*
 *      c u i n t . h
 *      Cubic, bicubic, and tricubic interpolation
 */
#ifndef CUINT_H
#define CUINT_H

typedef struct {          /* Two-dimensional result          */
    float  f,             /* Value              */
          fx, fy,         /* First derivatives  */
          fxx, fxy, fyy;  /* Second derivatives */
} PT2;

typedef struct {          /* Three-dimensional result          */
    float  f,             /* Value              */
          fx, fy, fz,     /* First derivatives  */
          fxx, fxy, fyy,  /* Second derivatives */
          fxz, fyz, fzz;
} PT3;

void  cucof();
void  cuint();
void  bcucof();
void  bcuint();
void  tcucof();
void  tcuint();
#endif CUINT_H
```

```
/*      D A T E _ T I M E . H
 * Include file for calendar and time routines
 * Bruce R. Julian, USGS Menlo Park, Calif.
 */
#ifndef DATETIME_H
#define DATETIME_H

#define BASEJDN      2440588          /* 1 January 1970      */

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

/* Time intervals */
#define SECOND       1
#define MINUTE       (60*SECOND)
#define HOUR         (60*MINUTE)
#define DAY          (24*HOUR)
#define WEEK         (7*DAY)
#define YEAR         (365*DAY)
#define LEAPYEAR     (366*DAY)
#define MILLISEC     (1e-3*SECOND)
#define MICROSEC     (1e-6*SECOND)
#define NANOSEC      (1e-9*SECOND)

/* Data types */
typedef double TIME;

/* External functions */
int    ckdate();
void   date();
bool   isleap();
long   jdn();
void   mnday();
TIME   timvar();
int     yrday();
#endif DATETIME_H
```



```
/*
 * /usr/include/local/dec_float.h
 */

typedef long DEC_FLOAT;
typedef struct dec_dble {
    long    h;
    long    l;
} DEC_DOUBLE;

double  cvdecf();
```

```
#ifndef DECODE_H
#define MAXBUF
#define MAXBUF 20
#endif MAXBUF
char dcbuf[MAXBUF];

char *strncpy();

/* ASCII-to-number conversion (for Fortran-like formatted reading) */
/* char *s is beginning of field */
/* int j is field length */
/* f is conversion function */
#define DECODE(s, j, f) (dcbuf[j]='\0', f(strncpy(dcbuf, s, j)))
#define HASDECPT index(dcbuf, '.')
#endif DECODE_H
```

```
/*          E A R T H C O N S T . H
 * Bruce R. Julian, USGS Menlo Park, Calif., 15 Dec 1983
 */
#ifndef EARTHCONST_H
#define EARTHCONST_H

#define REQUAT      6378.163      /* Equatorial radius      */
#define RPOLAR      6356.177      /* Polar radius           */
#define REARTH      6371          /* Equal-volume sphere    */
#define FLATTENING  (1/298.24)    /* (REQUAT-RPOLAR)/REQUAT */
#define KM          (5280*12*0.0000254) /* Kilometers per mile */
#define MILE        (1.0/KM)      /* Miles per kilometer    */
#endif
```

```
/*          F 7 7 T Y P E S . H
 * C data types used by f77 compiler (installation-dependent)
 * Version for Integrated Solutions (4.2BSD) Unix
 * Bruce R. Julian, USGS Menlo Park, Calif.  2 Jan 1986
 */
#define F77INT      int          /* Default integer data type */
#define F77LOG      int          /* Default logical data type */
#define F77SLARG    long         /* String-length argument type */
```

```

/*          L O C A L . H
 * Bruce R. Julian, USGS Menlo Park Calif., Dec 15 1983
 */
#ifndef LOCAL_H
#define LOCAL_H

#include "stdtyp.h"

#ifndef NULL
#define NULL 0
#endif

#define PRIVATE static          /* Better name for it */
#define FOREVER for(;;)        /* Infinite loop */
#define NOT !
#define YES 1                   /* Truth value */
#define NO 0                    /* Truth value */
#ifndef TRUE
#define TRUE 1                  /* Truth value */
#define FALSE 0                /* Truth value */
#endif
#define TRUE
#define DIM(x) (sizeof(x)/sizeof((x)[0])) /* Dimension of array */
#ifndef MAX
#define MAX(a,b) (((a)>(b)) ? (a) : (b)) /* Larger value */
#endif
#define MIN
#define MIN(a,b) (((a)<(b)) ? (a) : (b)) /* Smaller value */
#define ABS(a) ((a) > 0 ? (a) : -(a)) /* Absolute value */
#define FAIL -1                 /* Function return */
#define FAILED == -1
#define SUCCEED 0               /* Function return */
#define SUCCEEDED != -1
#define STDIN 0                 /* read() write() unit */
#define STDOUT 1                /* read() write() unit */
#define STDERR 2                /* read() write() unit */
#define READ 0                  /* Mode for open() */
#define WRITE 1                 /* Mode for open() */
#define UPDATE 2                /* Mode for open() */
#define HEAD 0                  /* Whence for lseek() */
#define HERE 1                  /* Whence for lseek() */
#define TAIL 2                  /* Whence for lseek() */
#define reg register
#define repeat do
#define until(c) while(!(c))
#define streq !strcmp           /* String comparison */
#define strneq !strncmp         /* String comparison */

#define ALLOC(x) (struct x *) malloc(sizeof(struct x))

#define TURN_ON(x, f)  (x |= (f))
#define TURN_OFF(x, f) (x &= ~(f))
#define IS_ON(x, f)    (x & (f))
#define IS_OFF(x, f)   (!(x & (f)))

#endif LOCAL_H

```

```
/*          MATHCONST.H
*/
#ifndef MATHCONST_H
#define MATHCONST_H

/* PI and its relatives */
#define PI      3.14159265358979324
#define PIINV   0.318309886183790672      /* 1./PI      */
#define HALFPI  1.57079632679489662
#define TWOPI   6.28318530717958648
#define FOURPI  12.5663706143591730
#define SQRTPI  1.77245385090551603
#define PI32    5.56832799683170785      /* PI**1.5    */

/* Irrational square roots */
#define SQRT2    1.41421356237309505
#define SQRT3    1.73205080756887729
#define SQRTHALF 0.707106781186547524

/* Logarithms */
#define LN2      0.693147180559945310
#define LN10     2.30258509299404568
#define LOG2     0.301029995663981195
#define LOGE     0.434294481903251828

/* Degrees per radian and vice-versa */
#define DEG      57.2957795130823209
#define RAD      0.0174532925199432957

/* One degree in radians, a la Mathematica */
#define Degree   0.0174532925199432957

/* Floating-point resolution (machine-dependent) */
#define EPS      5.96046448e-08
#define DEPS     1.38777878e-17

/* Euler's constant */
#define E_GAMMA  0.5772156649015328606

/* Array subscripts */
#define X 0
#define Y 1
#define Z 2
#define XX 0
#define XY 1
#define YX 1
#define YY 2
#define XZ 3
#define ZX 3
#define YZ 4
#define ZY 4
#define ZZ 5
#endif
```

```

/*
 *
 */
m o d e l 3 d . h

#ifdef MODEL3D_H
#define MODEL3D_H
#endif
#define NULL
#define NULL 0
#endif NULL

typedef struct {
    struct lim {
        float lat0, lon0, rot;
        int nx, ny, nz;
    } limits;
    float *xgrid; /* [0..nx-1] */
    float *ygrid; /* [0..ny-1] */
    float *zgrid; /* [0..nz-1] */
    float ***s; /* [0..nx-1][0..ny-1][0..nz-1] */
    float ***sx; /* [0..nx-1][0..ny-1][0..nz-1] */
    float ***sy; /* [0..nx-1][0..ny-1][0..nz-1] */
    float ***sz; /* [0..nx-1][0..ny-1][0..nz-1] */
    float ***sxy; /* [0..nx-1][0..ny-1][0..nz-1] */
    float ***sxz; /* [0..nx-1][0..ny-1][0..nz-1] */
    float ***syz; /* [0..nx-1][0..ny-1][0..nz-1] */
    float ***sxyz; /* [0..nx-1][0..ny-1][0..nz-1] */
} MODEL3D;

typedef struct { /* A point within the earth */
    float s, /* Wave slowness */
    sx, sy, sz, /* Spatial derivatives */
    sxx, syy, szz, /* Second derivatives */
    sxy, sxz, syz;
} EPOINT;

void free_md1();
void sder();
MODEL3D *rdmdl();
#endif MODEL3D_H

```

```
/*          Q E R R . H
 * Bruce R. Julian, USGS Menlo Park Calif., 15 Dec. 1983
 */
#ifndef QERR_H
#define QERR_H

#define ERRBASE      1000
#define EBADDAY      (ERRBASE + 1) /* Illegal day          */
#define EBADMONTH    (ERRBASE + 2) /* Illegal month      */
#define EBADYEAR     (ERRBASE + 3) /* Illegal year       */
#define EBADVALUE    (ERRBASE + 4) /* Illegal value      */
#define EFEWDATA     (ERRBASE + 5) /* Not enough data    */
#define ENOCONVRG    (ERRBASE + 6) /* Convergence Failure */
#define EUNDERFLOW   (ERRBASE + 7) /* Underflow          */
#define ENOSTATION   (ERRBASE + 8) /* No such station    */
#define ESIZE        (ERRBASE + 9) /* Inappropriate structure size */
#define ESINGULAR    (ERRBASE + 10) /* Singular matrix    */
#define ETOODEEP     (ERRBASE + 11) /* Focal depth too great */
#define EBADPHASE    (ERRBASE + 12) /* No such phase      */
#define ENOFEAS      (ERRBASE + 13) /* Constraints are inconsistent */
#define EINFIN       (ERRBASE + 14) /* Solution is unbounded */
#define ENOTFOUND    (ERRBASE + 15) /* Can't find desired record */
#define ENOCUROR     (ERRBASE + 16) /* No current origin  */
#define ENOCURAM     (ERRBASE + 17) /* No current average magnitude */
#define ECANTBRACKET (ERRBASE + 18) /* Root not bracketed */
#endif QERR_H
```



```

/*          Q U A K E . H
 * Bruce R. Julian, USGS Menlo Park Calif., Dec 15 1983
 */
#ifndef QUAKE_H
#define QUAKE_H

#include <local/stdtyp.h>
#include <local/date_time.h>

#ifndef NULL
#define NULL 0
#endif

#define repeat do
#define until(c) while(!(c))

#define FOREVER for(;;)           /* Infinite loop      */
#define NOT !                      /* Truth value       */
#define YES 1                      /* Truth value       */
#define NO 0                      /* Truth value       */
#define TRUE 1                    /* Truth value       */
#define FALSE 0                  /* Truth value       */
#define DIM(x) (sizeof(x)/sizeof((x)[0])) /* Dimension of array */
#ifndef MAX
#define MAX(a,b) ((a) > (b) ? (a) : (b)) /* Larger value      */
#endif MAX
#ifndef MIN
#define MIN(a,b) ((a) < (b) ? (a) : (b)) /* Smaller value     */
#endif MIN
#define ABS(a) ((a) > 0 ? (a) : -(a)) /* Absolute value    */
#define FAIL -1                   /* Function return   */
#define SUCCEED 0                 /* Function return   */
#define STDIN 0                   /* read() write() unit */
#define STDOUT 1                  /* read() write() unit */
#define STDERR 2                  /* read() write() unit */
#define READ 0                    /* Mode for open()   */
#define WRITE 1                   /* Mode for open()   */
#define UPDATE 2                  /* Mode for open()   */

/*----- E R R O R   C O D E S -----*/
#define ERRBASE 1000
#define EBADDAY (ERRBASE + 1) /* Illegal calendar day */
#define EBADMONT (ERRBASE + 2) /* Illegal calendar month */
#define EBADYEAR (ERRBASE + 3) /* Illegal calendar year */
#define EBADVALUE (ERRBASE + 4) /* Illegal value */
#define EFEWDATA (ERRBASE + 5) /* Not enough data */
#define ENOCONVRG (ERRBASE + 6) /* Failed to converge */
#define EUNDERFLOW (ERRBASE + 7) /* Arithmetic underflow */
#define ENOSTATION (ERRBASE + 8) /* Unknown seismograph station */
#define ESIZE (ERRBASE + 9)
#define ESINGULAR (ERRBASE + 10) /* Matrix is singular */
#define ETOODEEP (ERRBASE + 11) /* Focal depth too great */
#define EBADPHASE (ERRBASE + 12) /* Unknown seismic phase code */

/*----- E V E N T   O R I G I N   S T R U C T U R E -----*/
typedef struct {
    TIME o_time; /* Origin time */
    float o_lat; /* Epicentral latitude (Rad) */
    float o_lon; /* Epicentral longitude (Rad) */
    float o_depth; /* Focal depth */
} ORIGIN;

/*----- S T A T I O N   S T R U C T U R E -----*/
#define MXSTCOD 4
typedef struct {
    char s_code[MXSTCOD]; /* Station code */
    float gdc[3]; /* Geocentric direction cosines */
    float s_elev; /* Station elevation (m) */
} STATION;

/*----- R E A D I N G   S T R U C T U R E -----*/
#define CHANSIZE 6
#define MXPHCOD 8
/* Masks for flags: */
#define ASSOCIATED 01 /* Phase comes from current event */
#define T_EXISTS 02 /* Arrival time exists */
#define USE_T 04 /* Should use arrival time */
#ifdef SLO
#define S_EXISTS 010 /* Slowness-azimuth exists */
#define USE_S 020 /* Should use slowness-azimuth */
#endif
typedef struct {
    STATION sta; /* Station structure */
    char chan[CHANSIZE]; /* Channel code */
    char phcode[MXPHCOD]; /* Phase code */
    TIME atime; /* Arrival time */
    float setime; /* Std. err. of arrival time (s) */
    char onset; /* 'e' or 'i' */
}

```

```

    char    pol;           /* Polarity: e.g. 'u' or 'd' */
    char    qual;          /* Quality code */
    float   amp;
    float   freq;
#ifdef SLO
    float   slowness;       /* Slowness (s/Rad) */
    float   azimuth;        /* Propagation azimuth (Rad) */
    float   seslow;         /* Std. err. of slowness (s/Rad) */
#endif
    char    flags;
} READING;

#ifdef SLO
/* Array subscripts for slowness-vector components */
#define EAST 0
#define NORTH 1
#endif

/*----- A S S O C I A T I O N   S T R U C T U R E -----*/
typedef struct {
    float   delta;          /* Epicentral distance (Rad) */
    float   aze;            /* Epicenter -> station azimuth (Rad) */
    float   azs;            /* Station -> epicenter azimuth (Rad) */
    float   s1[3];          /* Slowness vector at hypocenter */
    float   s2[3];          /* Slowness vector at station */
    int     brnum;          /* Branch number */
    int     ercode;         /* From body-wave subroutine */
    float   atresid;        /* Arrival-time residual */
    float   atsigma;        /* Arrival-time std. err. */
    float   atwt;           /* Arrival-time weight */
    float   atimp;          /* Arrival-time importance */
#ifdef SLO
    float   szresid[2];      /* slowness-azimuth residual (s/Rad) */
    float   szsigma;        /* Std. err. of slowness-azimuth (s/Rad) */
    float   szwt;           /* Weight of slowness-azimuth obs. */
    float   szimp;          /* Importance of slowness-azimuth obs. */
#endif
} ASSOC;

/*----- R E L O C   C O N T R O L   S T R U C T U R E -----*/
typedef struct {
    tbool   hold[4];        /* Coordinate-restraint flags */
    float   epcvthresh;     /* Convergence threshold for epicenter */
    float   fdcvthresh;     /* Convergence threshold for focal depth */
    float   otcvthresh;     /* Convergence threshold for origin time */
    float   min_depth;      /* Smallest allowed focal depth */
    float   max_depth;      /* Largest allowed focal depth */
    int     max_iter;       /* Iteration limit */
    int     max_damp;       /* Damping-step limit */
    float   lstart;         /* Initial damping parameter */
    float   lmin;           /* Minimum damping parameter */
    float   lfact;          /* Change in lambda per step */
    float   zdamp;          /* Relative depth damping */
    float   mu;             /* Frequency of erratic data */
    int     iprint;         /* Print-control flag */
} RLCTRL;

/* Subscripts for origin components */
#define ILAT 0
#define ILON 1
#define IDEPTH 2
#define ITIME 3

/* Print control */
#define NEVER 0
#define FINAL 1
#define ALL 2

/*----- O R I G I N   S T A T I S T I C S   S T R U C T U R E -----*/
typedef struct {
    float   chisq;          /* Chi squared */
    float   sumwt;          /* Sum of weights of readings */
    int     ndf;            /* No. of degrees of freedom */
    int     iter;           /* No. of iterations performed */
    int     ntasoc;         /* No. of associated arrival-time obs. */
    int     ntused;         /* No. of arrival-time obs. used */
#ifdef SLO
    int     nsasoc;         /* No. of associated slowness obs. */
    int     nsused;         /* No. of slowness obs. used */
#endif
    tbool   converged;
    tbool   held[4];
} STATIS;

/*----- B O D Y - W A V E   R E S U L T   S T R U C T U R E -----*/
typedef struct {
    int     bw_brnum;       /* Branch number */
    float   bw_t;          /* Travel time */

```

```
float bw_set;          /* Std. err. of t          */
float bw_s1[3];        /* Slowness vector at hypocenter */
float bw_s2[3];        /* Slowness vector at station    */
float bw_tdd;          /* dp/ddelta                    */
float bw_tdh;          /* dp/dh                        */
} BWRSLT;
#endif
```

```
/*          S T A B L E . H
 * USGS Calnet seismograph-station tables
 */
#ifndef STABLE_H
#define STABLE_H

#define CSIZE 4

/* Structure describing open station file */
typedef struct stable {
    char      *idsta;          /* List of station codes      */
    unsigned  nsta;            /* No. of codes in idsta      */
    FILE      *fp;             /* Station file                */
    long      offset;          /* Offset of info. of first sta.*/
} STABLE;

/* Structure containing info. for one station */
typedef struct stinfo {
    char      code[CSIZE];     /* Station code                */
    float     lat, lon;        /* Coordinates                  */
    int       ielev;           /* Elevation (meters)          */
    float     p_anom;          /* P-time anomaly              */
    #if 0
    float     s_anom;          /* S-time anomaly              */
    #endif 0
    float     dmag_anom;       /* Duration-magnitude anomaly  */
    float     amag_anom;       /* Amplitude-magnitude anomaly */
    int       inst;            /* Instrument code              */
    float     t_dflt;          /* Default period for amp_mag  */
    float     cal_dflt;        /* Calibration for amp_mag      */
} STINFO;

STABLE *stopen();
void stclose();
int stindex();
int stget();
#endif STABLE_H
```

```

/*      S T D T Y P . H
 * Version for PDP-11, VAX-11, and ISI V24
 */
#ifndef STDTP_H
#define STDTP_H

/* Specify machine characteristics */
#ifdef sun
#define USHORT
#define VOID /* Compiler supports void type */
#define UTINY /* Compiler supports unsigned char type */
#endif
#ifdef mc68000 /* Integrated Solutions UNIX */
#define USHORT /* Compiler supports unsigned short type */
#define VOID
#endif
#ifdef vax
#define USHORT /* Compiler supports unsigned short type */
#define UTINY /* Compiler supports unsigned char type */
#define VOID /* Compiler supports void type */
#endif
#ifdef pdp11
#define USHORT /* Compiler supports unsigned short type */
#ifdef makedev /* Crock to see if <sys/types.h> has been included */
#define VOID /* <sys/types.h> has "typedef short void" */
#endif
#endif

typedef char      tbits, tbool;
typedef int       bool;
typedef unsigned int sizetype;
typedef long      lbits;
typedef short     bits, metachar;
#ifdef USHORT
#ifdef sun
typedef unsigned short ushort;
#endif
#ifdef sparc
#undef USHORT
#else
USHORT
typedef unsigned ushort; /* Assumes 16-bit machine */
#endif
#endif
#ifdef TINY
typedef char      tiny;
#undef TINY
#define TINY(n) (char) (n)
#else
TINY
typedef char      tiny;
#define TINY(n) (char) ((n) & 0x80 ? (~0xF7 | (n)) : (n))
#endif
#ifdef UTINY
typedef unsigned char utiny;
#undef UTINY
#define UTINY(n) (unsigned char) (n)
#else
UTINY
typedef char      utiny;
#define UTINY(n) (unsigned) ((n) & 0xFF)
#endif
#ifdef VOID
#define void int
#endif
#define LURSHIFT(n,b) (((long) (n) >> (b)) & (0x7FFFFFFF >> (b-1)))
#endif STDTP_H

```

15. Tables

Table 1. Station Locations and Descriptions.

Station Code	Station Name	Station Location			Sensor Type	Began (year.day)	Rock and Comments
		Lat. (°N)	Lon. (°E)	El. (m)			
SFT2	Fern Peak	37.15244 37.153042	-121.81133 -121.812232	518	STS-2 or SV/H-1	1993.127‡	Serpentinite, adjacent to various Franciscan†. Surface insulation required.
MHC	Mount Hamilton	37.342	-121.642	1282	STS-1	1986.253	Franciscan (greenstone). Permanent UCB vault.
SFT4	San Antonio Valley	37.38883	-121.49450	600	STS-2 or SV/H-1	1993.228‡	Franciscan, metabasalt? Surface insulation unknown.
SFT5	Ingram Canyon	37.50483 37.505053	-121.32664 -121.328514	312	STS-2 or SV/H-1	1993.097‡	Franciscan graywacke (or metavolcanic)†. Surface insulation required.
CMB	Columbia	38.035	-120.383	719	STS-1	1992.176	Paleozoic marine limestone. Permanent UCB vault.

†Second set of coordinates derived from GPS, using medians. For SFT2 we used 230 GPS readings, for SFT5 1,985 readings. SFT2 GPS location is 104 m NW of the topographic-map location; SFT5 GPS location is 167 m WNW of its topographic-map location. It is not clear which type of location is "better", though the GPS locations are relative to a more modern datum.

‡In intermittent operation.

Table 2. Recording Periods and Instrument Specifications

Date on (day:hour:minute)	Date off	sps	sensor type	SFT2, New Almaden gain (\times)	DAS S/N:type*	notes
128:03:08	137:19:03	10	STS-2	1	0540:-02	†
190:19:41	196:02:19	20	SV/H-1	32	7062:-07	
196:03:57	201:20:35	20	STS-2	1	7062:-07	
201:21:20	210:01:35	20	STS-2	1	7062:-07	
210:03:57	~221:02:35	20	STS-2	1	7062:-07	Disk filled
Date on	Date off	SFT5, Ingram Canyon				
097:01:38	103:19:39	10	STS-2	1	0540:-02	†
104:01:25	110:21:14	10	STS-2	1	0540:-02	†
110:23:37	125:21:36	10	STS-2	1	0540:-02	†
110:23:37	125:21:36	10	STS-2	1	0540:-02	†
097:02:21	103:20:48	10	SV/H-1	128	0547:-02	†
104:01:32	~104:04:32	10	SV/H-1	128	0547:-02	DAS failed‡ †
110:23:20	125:22:08	10	SV/H-1	128	0547:-02	†
125:23:27	138:01:57	10	SV/H-1	32	0547:-02	†
191:02:12	197:02:07	20	SV/H-1	32	7064:-07	
197:03:31	205:02:34	20	STS-2	1	7064:-07	
205:03:13	~216:01:51	20	STS-2	1	7064:-07	Disk filled
216:20:14	222:18:41	20	STS-2	1	7064:-07	
197:03:11	205:03:24	20	Fluke 80TK	1	7080:-07	
205:03:53	216:20:30	20	Fluke 80TK	32	7080:-07	
216:23:33	222:19:17	20	Fluke 80TK	32	7080:-07	

On 72A-02, the data are recorded at 10 sps with 32-bit data from a 16-bit analog-to-digital converter (ADC). Decimation yields about 19-bits resolution. On a 72A-07, the sample rate is 20 sps with 32-bit recording from a 24-bit ADC. Gains are higher on the SV-1 and SH-1 because they are simple electromagnetic transducers, without preamplification or feedback.

For vault-temperature tests (DAS S/N 7080), sample rate was 20 sps, because this is the lowest rate available on a 72A-07, with 32-bit recording from a 24-bit ADC. These specifications far outreach the 80TK's and thermocouple's capabilities. Recording at 1 sps and 16 bits is appropriate when a spare channel of a 72A-02 is available.

*For example, "0540:-02" means DAS serial number 0540, which is a 72A-02.

†WWVB recorded and evaluated as backup to GPS clock. All other intervals (72A-07 recorders) had only the GPS clock.

‡DAS failure is due to programming incompatibility (a 100 sps timed recording of the main-sensor channels, with a repeat interval of 99 days, 23:59:59). The long repeat interval probably caused a DSP timeout after several hours, by overreaching DAS software capabilities. This stream was intended to induce high-sample-rate recording of the sensors during calibration, but was discontinued after this one recording interval.

Table 3. Events Extracted from Continuous Records, 1993

DAS Channels V N E	DAS S/N	Sensor Type	Site Name	Start Time (day hr min sec)	Length (s)	Event description(s) (L,R,T,V= Local, Regional, Teleseismic, Vehicular)
4 5 6	0540	STS-2	SFT5	097 01 44 45†	915	Calibration
1 2 3	0547	SV/H-1	SFT5	097 02 23 45	150	Calibration
4 5 6	0540	STS-2	SFT5	098 00 32 30	75	small L
1 2 3	0547	SV/H-1	SFT5	098 00 32 30	75	...
4 5 6	0540	STS-2	SFT5	098 02 10 30	165	R?
1 2 3	0547	SV/H-1	SFT5	098 02 10 30	165	...
4 5 6	0540	STS-2	SFT5	098 10 37 00	120	small L
1 2 3	0547	SV/H-1	SFT5	098 10 37 00	120	...
4 5 6	0540	STS-2	SFT5	098 14 28 15	180	L
1 2 3	0547	SV/H-1	SFT5	098 14 28 15	180	...
4 5 6	0540	STS-2	SFT5	098 15 59 00	135	L
1 2 3	0547	SV/H-1	SFT5	098 15 59 00	135	...
4 5 6	0540	STS-2	SFT5	098 16 49 45	135	R
1 2 3	0547	SV/H-1	SFT5	098 16 49 45	135	R
4 5 6	0540	STS-2	SFT5	098 19 21 00	1620	Dominican Republic, m5.1, 20-s L/R seen
1 2 3	0547	SV/H-1	SFT5	098 19 21 00	1620	...
4 5 6	0540	STS-2	SFT5	098 23 58 30	90	R
1 2 3	0547	SV/H-1	SFT5	098 23 58 30	90	...
4 5 6	0540	STS-2	SFT5	099 02 40 45	105	very small L?
1 2 3	0547	SV/H-1	SFT5	099 02 40 45	105	...
4 5 6	0540	STS-2	SFT5	099 06 58 00	120	small L
1 2 3	0547	SV/H-1	SFT5	099 06 58 00	120	...
4 5 6	0540	STS-2	SFT5	099 07 12 00	3600	New Zealand, m5.8 (not seen)
1 2 3	0547	SV/H-1	SFT5	099 07 12 00	3600	...
4 5 6	0540	STS-2	SFT5	099 09 53 00	60	very small L
1 2 3	0547	SV/H-1	SFT5	099 09 53 00	60	...
4 5 6	0540	STS-2	SFT5	099 10 54 45	75	very small R?
1 2 3	0547	SV/H-1	SFT5	099 10 54 45	75	...
4 5 6	0540	STS-2	SFT5	099 11 48 30	2560	Luzon, m5.4, 20-s L/R seen
1 2 3	0547	SV/H-1	SFT5	099 11 48 30	2560	...
4 5 6	0540	STS-2	SFT5	099 16 25 00	60	very small L?
1 2 3	0547	SV/H-1	SFT5	099 16 25 00	60	...
4 5 6	0540	STS-2	SFT5	099 17 37 15	105	very small R?
1 2 3	0547	SV/H-1	SFT5	099 17 37 15	105	...
4 5 6	0540	STS-2	SFT5	099 21 20 20	70	small L
1 2 3	0547	SV/H-1	SFT5	099 21 20 20	70	...
4 5 6	0540	STS-2	SFT5	099 21 52 00	90	L
1 2 3	0547	SV/H-1	SFT5	099 21 52 00	90	...
4 5 6	0540	STS-2	SFT5	100 00 15 00	360	Good T or R (unknown hypocenter), small L at end
1 2 3	0547	SV/H-1	SFT5	100 00 15 00	360	...

†File names for this event, for example, are: "93097.014445.sts2.sft5.4", "93097.014445.sts2.sft5.5", "93097.014445.sts2.sft5.6", and "93097.014445.sts2.sft5.ah".

Table 3 (continued)

DAS Channels V N E	DAS S/N	Sensor Type	Site Name	Start Time (day hr min sec)	Length (s)	Event description(s) (L,R,T,V= Local, Regional, Teleseismic, Vehicular)
4 5 6 1 2 3	0540 0547	STS-2 SV/H-1	SFT5 SFT5	100 03 30 20 100 03 30 20	60 60	L?, low-frequency, S-P~8 s ...
4 5 6 1 2 3	0540 0547	STS-2 SV/H-1	SFT5 SFT5	100 08 15 30 100 08 15 30	120 120	R ...
4 5 6 1 2 3	0540 0547	STS-2 SV/H-1	SFT5 SFT5	100 09 28 00 100 09 28 00	60 60	very small L? ...
4 5 6 1 2 3	0540 0547	STS-2 SV/H-1	SFT5 SFT5	100 10 02 45 100 10 02 45	60 60	very small L?, low-frequency ...
4 5 6 1 2 3	0540 0547	STS-2 SV/H-1	SFT5 SFT5	100 16 53 30 100 16 53 30	60 60	small L ...
4 5 6 1 2 3	0540 0547	STS-2 SV/H-1	SFT5 SFT5	101 00 04 10 101 00 04 10	125 125	very small R? ...
4 5 6 1 2 3	0540 0547	STS-2 SV/H-1	SFT5 SFT5	101 00 25 45 101 00 25 45	60 60	small L ...
4 5 6 1 2 3	0540 0547	STS-2 SV/H-1	SFT5 SFT5	101 06 45 50 101 06 45 50	60 60	small L ...
4 5 6 1 2 3	0540 0547	STS-2 SV/H-1	SFT5 SFT5	101 07 12 00 101 07 12 00	5400 5400	New Zealand, m5.8, and Volcano Island, m5.0, 20-s L/R seen ...
4 5 6 1 2 3	0540 0547	STS-2 SV/H-1	SFT5 SFT5	101 12 27 00 101 12 27 00	90 90	L ...
4 5 6 1 2 3	0540 0547	STS-2 SV/H-1	SFT5 SFT5	101 19 41 00 101 19 41 00	60 60	small L ...
4 5 6 1 2 3	0540 0547	STS-2 SV/H-1	SFT5 SFT5	102 03 21 00 102 03 21 00	75 75	L ...
4 5 6 1 2 3	0540 0547	STS-2 SV/H-1	SFT5 SFT5	102 06 43 45 102 06 43 45	90 90	very small L/R? ...
4 5 6 1 2 3	0540 0547	STS-2 SV/H-1	SFT5 SFT5	102 07 57 45 102 07 57 45	75 75	small L/R ...
4 5 6 1 2 3	0540 0547	STS-2 SV/H-1	SFT5 SFT5	102 08 40 10 102 08 40 10	65 65	very small L? ...
4 5 6 1 2 3	0540 0547	STS-2 SV/H-1	SFT5 SFT5	102 10 00 00 102 10 00 00	7200 7200	Noise sample, morning ...
4 5 6 1 2 3	0540 0547	STS-2 SV/H-1	SFT5 SFT5	102 21 47 00 102 21 47 00	75 75	very small L ...
4 5 6 1 2 3	0540 0547	STS-2 SV/H-1	SFT5 SFT5	102 22 00 00 102 22 00 00	7200 7200	Noise sample, afternoon ...
4 5 6 1 2 3	0540 0547	STS-2 SV/H-1	SFT5 SFT5	103 09 05 30 103 09 05 30	60 60	very small R? ...
4 5 6 1 2 3	0540 0547	STS-2 SV/H-1	SFT5 SFT5	103 19 19 30 103 20 42 45	960 180	Calibration Calibration
4 5 6	0540	STS-2	SFT5	104 06 04 00	3600	Large T, Fox Islands, Aleutians m5.6 h33

Table 3 (continued)

DAS Channels V N E	DAS S/N	Sensor Type	Site Name	Start Time (day hr min sec)	Length (s)	Event description(s) (L,R,T,V= Local, Regional, Teleseismic, Vehicular)
4 5 6	0540	STS-2	SFT5	104 11 54 45	75	very small R?
4 5 6	0540	STS-2	SFT5	104 12 14 30	90	L
4 5 6	0540	STS-2	SFT5	104 12 16 15	105	nice L
4 5 6	0540	STS-2	SFT5	104 19 58 30	90	small R or V?
4 5 6	0540	STS-2	SFT5	105 13 42 15	1065	small T, Fiji m4.9 h457
4 5 6	0540	STS-2	SFT5	105 16 26 45	75	very small R?
4 5 6	0540	STS-2	SFT5	105 21 36 45	75	small R or T? No QED hypocenter found
4 5 6	0540	STS-2	SFT5	106 04 14 00	960	small T, Alaska m5.2 h32
4 5 6	0540	STS-2	SFT5	106 14 18 30	3990	T, Fiji m5.9 h569; S SS
4 5 6	0540	STS-2	SFT5	106 17 16 15	75	small L
4 5 6	0540	STS-2	SFT5	106 20 34 00	120	R, S-P~23s
4 5 6	0540	STS-2	SFT5	107 12 43 45	255	very small R, d5.28; P and S
4 5 6	0540	STS-2	SFT5	107 13 37 00	105	R, S-P~10s
4 5 6	0540	STS-2	SFT5	107 15 51 30	120	L, S-P~7s
4 5 6	0540	STS-2	SFT5	107 20 43 30	1890	T, Fiji m5.7 h33; S SS surface
4 5 6	0540	STS-2	SFT5	108 09 25 45	2775	T, Peru m6.2 h91; P S SKS surface P'P'?
4 5 6	0540	STS-2	SFT5	108 14 22 45	3735	T, Pacific-Antarctic Ridge m5.5 h10; weak S, long ringing surface
4 5 6	0540	STS-2	SFT5	109 02 31 00	150	R, northern California m3.6
4 5 6	0540	STS-2	SFT5	109 18 09 10	140	odd L/R, emergent, S-P~10s
4 5 6	0540	STS-2	SFT5	109 21 14 50	6070	T, Indonesia m6.0 h82; PPP PKiKP SKS Sdiff SP/PS SS SKKS, several surface modes
4 5 6	0540	STS-2	SFT5	110 01 34 15	165	R, S-P~35s
4 5 6	0540	STS-2	SFT5	110 05 38 45	150	R, California-Nevada border d1.9
4 5 6	0540	STS-2	SFT5	110 08 25 45	3615	weak T, MacQuarie m? h10; surface wave only
4 5 6	0540	STS-2	SFT5	110 12 38 10	95	small R, S-P~10s
4 5 6	0540	STS-2	SFT5	110 12 43 30	105	R, S-P~10s
4 5 6	0540	STS-2	SFT5	110 16 36 30	690	small T, Fiji m5.6 h592; P, very small S
4 5 6	0540	STS-2	SFT5	110 20 56 30	930	Calibration
4 5 6	0540	STS-2	SFT5	110 23 53 10	200	small R? S-P~15s
1 2 3	0547	SV/H-1	SFT5	110 23 53 10	200	...
4 5 6	0540	STS-2	SFT5	111 01 38 00	135	small R or V, S-P~13s
1 2 3	0547	SV/H-1	SFT5	111 01 38 00	135	...
4 5 6	0540	STS-2	SFT5	111 10 06 30	150	small R, S-P~26s
1 2 3	0547	SV/H-1	SFT5	111 10 06 30	150	...
4 5 6	0540	STS-2	SFT5	111 15 45 15	105	small R (V?), S-P~15s
1 2 3	0547	SV/H-1	SFT5	111 15 45 15	105	...

Table 3 (continued)

DAS Channels V N E	DAS S/N	Sensor Type	Site Name	Start Time (day hr min sec)	Length (s)	Event description(s) (L,R,T,V= Local, Regional, Teleseismic, Vehicular)
4 5 6 1 2 3	0540 0547	STS-2 SV/H-1	SFT5 SFT5	112 00 27 30 112 00 27 30	3270 3270	R, S-P~60s, southern California ...
4 5 6 1 2 3	0540 0547	STS-2 SV/H-1	SFT5 SFT5	112 03 32 00 112 03 32 00	195 195	R, no S?, northern California ...
4 5 6 1 2 3	0540 0547	STS-2 SV/H-1	SFT5 SFT5	112 05 55 45 112 05 55 45	135 135	R, S-P~23s ...
4 5 6 1 2 3	0540 0547	STS-2 SV/H-1	SFT5 SFT5	112 06 37 00 112 06 37 00	90 90	R, S-P~10s ...
4 5 6 1 2 3	0540 0547	STS-2 SV/H-1	SFT5 SFT5	114 03 17 15 114 03 17 15	105 105	very small R, S-P~13s ...
4 5 6 1 2 3	0540 0547	STS-2 SV/H-1	SFT5 SFT5	114 06 22 30 114 06 22 30	120 120	R, S-P~13s ...
4 5 6 1 2 3	0540 0547	STS-2 SV/H-1	SFT5 SFT5	114 10 04 15 114 10 04 15	2565 2565	T, Fiji m5.4 h599; P S+SKS sS sSKS, nice little event ...
4 5 6 1 2 3	0540 0547	STS-2 SV/H-1	SFT5 SFT5	114 19 07 45 114 19 07 45	75 75	small L, nil S-P ...
4 5 6 1 2 3	0540 0547	STS-2 SV/H-1	SFT5 SFT5	115 09 31 00 115 09 31 00	480 480	large R, western Arizona m5.0 h10 d7.62 ...
4 5 6 1 2 3	0540 0547	STS-2 SV/H-1	SFT5 SFT5	115 18 03 30 115 18 03 30	2145 2145	small T, Chile m? h22; only surface ...
4 5 6 1 2 3	0540 0547	STS-2 SV/H-1	SFT5 SFT5	115 23 01 30 115 23 01 30	120 120	very small R, S-P~24s ...
4 5 6 1 2 3	0540 0547	STS-2 SV/H-1	SFT5 SFT5	116 14 58 00 116 14 58 00	135 135	R, S-P~36s ...
4 5 6 1 2 3	0540 0547	STS-2 SV/H-1	SFT5 SFT5	117 22 14 15 117 22 14 15	90 90	small L, S-P~6s ...
4 5 6 1 2 3	0540 0547	STS-2 SV/H-1	SFT5 SFT5	118 02 41 30 118 02 41 30	1350 1350	very small T, Kuril m5.2 h99; very weak S, SKS, others? ...
4 5 6 1 2 3	0540 0547	STS-2 SV/H-1	SFT5 SFT5	119 08 22 15 119 08 22 15	975 975	nice large R, western Arizona m5.4 h10 d7.64 ...
4 5 6 1 2 3	0540 0547	STS-2 SV/H-1	SFT5 SFT5	119 11 20 45 119 11 20 45	120 120	R, S-P~23s ...
4 5 6 1 2 3	0540 0547	STS-2 SV/H-1	SFT5 SFT5	119 12 22 45 119 12 22 45	195 195	nice large R, central California m? h5 d1.82, S-P~25s ...
4 5 6 1 2 3	0540 0547	STS-2 SV/H-1	SFT5 SFT5	120 12 08 00 120 12 08 00	90 90	small L (or V?), S-P~9s ...
4 5 6 1 2 3	0540 0547	STS-2 SV/H-1	SFT5 SFT5	120 19 24 15 120 19 24 15	90 90	nice L, S-P~3s ...

Table 3 (continued)

DAS Channels V N E	DAS S/N	Sensor Type	Site Name	Start Time (day hr min sec)	Length (s)	Event description(s) (L,R,T,V= Local, Regional, Teleseismic, Vehicular)
4 5 6	0540	STS-2	SFT5	121 00 18 00	75	small L or V or rock fall; only one phase
1 2 3	0547	SV/H-1	SFT5	121 00 18 00	75	...
4 5 6	0540	STS-2	SFT5	121 06 48 30	210	R, central California m? h10 d1.14; S-P~15s
1 2 3	0547	SV/H-1	SFT5	121 06 48 30	210	...
4 5 6	0540	STS-2	SFT5	121 18 24 30	150	small R? V?, S-P~32s
1 2 3	0547	SV/H-1	SFT5	121 18 24 30	150	...
4 5 6	0540	STS-2	SFT5	122 09 28 45	225	R, central California m? h10 d1.71
1 2 3	0547	SV/H-1	SFT5	122 09 28 45	225	...
4 5 6	0540	STS-2	SFT5	122 11 41 30	7710	large T, Sandwich m6.2 h33; SS+/- P'P' SKKP/PKKS long surface
1 2 3	0547	SV/H-1	SFT5	122 11 41 30	7710	...
4 5 6	0540	STS-2	SFT5	122 15 36 45	735	weak T, Tonga m5.6 h122; OK P, weak S
1 2 3	0547	SV/H-1	SFT5	122 15 36 45	735	...
4 5 6	0540	STS-2	SFT5	122 19 16 45	120	small R, S-P~14s
1 2 3	0547	SV/H-1	SFT5	122 19 16 45	120	...
4 5 6	0540	STS-2	SFT5	122 20 05 00	180	R, no S?
1 2 3	0547	SV/H-1	SFT5	122 20 05 00	180	...
4 5 6	0540	STS-2	SFT5	123 12 07 15	255	R, northern California m? h5 d2.14
1 2 3	0547	SV/H-1	SFT5	123 12 07 15	255	...
4 5 6	0540	STS-2	SFT5	123 12 48 15	105	L, S-P~7s
1 2 3	0547	SV/H-1	SFT5	123 12 48 15	105	...
4 5 6	0540	STS-2	SFT5	123 21 20 00	135	small distal R?
1 2 3	0547	SV/H-1	SFT5	123 21 20 00	135	...
4 5 6	0540	STS-2	SFT5	124 03 57 30	150	R, S-P~14s
1 2 3	0547	SV/H-1	SFT5	124 03 57 30	150	...
4 5 6	0540	STS-2	SFT5	125 21 19 00	990	Calibration
1 2 3	0547	SV/H-1	SFT5	125 22 04 00	210	Calibration
1 2 3	0547	SV/H-1	SFT5	126 02 40 30	90	small L, S-P~3
1 2 3	0547	SV/H-1	SFT5	126 03 13 15	165	small R or V, S-P~17
1 2 3	0547	SV/H-1	SFT5	126 10 49 45	150	small R (and foreshock?), S-P~12
1 2 3	0547	SV/H-1	SFT5	126 13 12 00	900	T, Brazil m5.9 h586 d65; P PCP P coda S Scoda
1 2 3	0547	SV/H-1	SFT5	126 20 09 30	90	very small L?, S-P~0
1 2 3	0547	SV/H-1	SFT5	126 20 47 45	105	small R?, S-P~10
1 2 3	0547	SV/H-1	SFT5	127 14 54 30	90	small L, S-P~7
4 5 6	0540	STS-2	SFT2	128 03 16 30	930	Calibration
4 5 6	0540	STS-2	SFT2	128 11 58 30	150	very small R, southern California m? h10 d4.4
1 2 3	0547	SV/H-1	SFT5	128 11 58 30	180	extremely small ...

Table 3 (continued)

DAS Channels V N E	DAS S/N	Sensor Type	Site Name	Start Time (day hr min sec)	Length (s)	Event description(s) (L,R,T,V= Local, Regional, Teleseismic, Vehicular)
1 2 3	0547	SV/H-1	SFT5	128 13 48 00	180	R, central California m? h10 d0.7, S-P~10
4 5 6	0540	STS-2	SFT2	128 13 48 00	180	... S-P~11
4 5 6	0540	STS-2	SFT2	128 20 26 00	90	small L, S-P~5
1 2 3	0547	SV/H-1	SFT5	128 20 26 00	90	... S-P~8
4 5 6	0540	STS-2	SFT2	129 09 24 00	1200	very small T, Guatemala m4.6 h71 d36; small surface wave only
1 2 3	0547	SV/H-1	SFT5	129 09 24 00	1200	...
4 5 6	0540	STS-2	SFT2	129 12 03 00	90	very small L?, or V?, S-P~3
1 2 3	0547	SV/H-1	SFT5	129 12 03 00	90	very small R, S-P~10
4 5 6	0540	STS-2	SFT2	129 12 07 15	75	very small L?, low frequency
1 2 3	0547	SV/H-1	SFT5	129 12 07 15	90	...
1 2 3	0547	SV/H-1	SFT5	129 12 33 45	100	very small L?, S-P~0?
4 5 6	0540	STS-2	SFT2	129 16 00 00	7200	Day noise sample, 9:00-11:00 am local
1 2 3	0547	SV/H-1	SFT5	129 22 18 00	480	small R, Oregon m4.8 h10 d8.1
4 5 6	0540	STS-2	SFT2	129 22 18 00	480	...
1 2 3	0547	SV/H-1	SFT5	130 00 29 45	110	small L?, S-P~5
4 5 6	0540	STS-2	SFT2	130 00 29 45	110	...
4 5 6	0540	STS-2	SFT2	130 04 15 45	2715	small T, East Pacific Rise m4.8 h10 d90; surface wave only
1 2 3	0547	SV/H-1	SFT5	130 04 15 45	2715	...
4 5 6	0540	STS-2	SFT2	130 06 49 15	120	small R, northern California m? h35 d3.9
1 2 3	0547	SV/H-1	SFT5	130 06 49 15	150	extremely small ...
1 2 3	0547	SV/H-1	SFT5	130 07 21 40	110	small L, S-P~9
4 5 6	0540	STS-2	SFT2	130 07 21 40	110	very small L or R, S-P~10
4 5 6	0540	STS-2	SFT2	130 08 10 00	7200	Night noise sample, 1:10-3:10 am local
1 2 3	0547	SV/H-1	SFT5	130 15 40 00	150	R, central California m? h10 d0.8, S-P~14
4 5 6	0540	STS-2	SFT2	130 15 40 00	150	... S-P~24
4 5 6	0540	STS-2	SFT2	131 02 58 20	1240	very small T, Galapagos m4.9 h10 d40; only weak surface wave
1 2 3	0547	SV/H-1	SFT5	131 02 58 20	1240	...
4 5 6	0540	STS-2	SFT2	131 03 44 40	200	2*L, S-P~6-7
1 2 3	0547	SV/H-1	SFT5	131 03 44 40	200	2*R, S-P~11
1 2 3	0547	SV/H-1	SFT5	131 04 16 30	90	very small L
4 5 6	0540	STS-2	SFT2	131 04 16 30	90	... S-P~1.5
4 5 6	0540	STS-2	SFT2	131 07 42 10	110	R, S-P~10
1 2 3	0547	SV/H-1	SFT5	131 07 42 10	110	... S-P~15

Table 3 (continued)

DAS Channels V N E	DAS S/N	Sensor Type	Site Name	Start Time (day hr min sec)	Length (s)	Event description(s) (L,R,T,V= Local, Regional, Teleseismic, Vehicular)
4 5 6 1 2 3	0540 0547	STS-2 SV/H-1	SFT2 SFT5	131 10 09 40 131 10 09 40	110 110	L, S-P-7 R, S-P-12
4 5 6	0540	STS-2	SFT2	131 10 32 00	75	very small L, S-P-0
4 5 6 1 2 3	0540 0547	STS-2 SV/H-1	SFT2 SFT5	131 15 58 45 131 15 58 45	75 90	small L, S-P-7 very small L, S-P-?
4 5 6 1 2 3	0540 0547	STS-2 SV/H-1	SFT2 SFT5	131 16 36 30 131 16 36 30	210 210	small L, S-P-6, and very small L (0.1 of amplitude) ... R, S-P-11, ... R ...
4 5 6 1 2 3	0540 0547	STS-2 SV/H-1	SFT2 SFT5	131 18 39 45 131 18 39 45	8715 8715	large T, Mindanao m6.1 h33 d103; P PP (ps)SKS, PS, many others, long surface wave ...
1 2 3 4 5 6	0547 0540	SV/H-1 STS-2	SFT5 SFT2	131 23 32 00 131 23 32 00	120 120	small R? or V?, S-P-22 very small R, S-P-?
4 5 6 1 2 3	0540 0547	STS-2 SV/H-1	SFT2 SFT5	132 00 47 30 132 00 47 30	240 240	small R, southern California m? h5 d4.4 ...
4 5 6 1 2 3	0540 0547	STS-2 SV/H-1	SFT2 SFT5	132 03 08 30 132 03 08 30	135 135	nice L, S-P-3 ... S-P-6
1 2 3 4 5 6	0547 0540	SV/H-1 STS-2	SFT5 SFT2	132 12 45 40 132 12 45 40	100 100	small L, S-P-6 ... S-P-5?
4 5 6 1 2 3	0540 0547	STS-2 SV/H-1	SFT2 SFT5	132 12 55 15 132 12 55 15	90 90	small L, S-P-0? very small L, S-P-0?
4 5 6 1 2 3	0540 0547	STS-2 SV/H-1	SFT2 SFT5	132 14 19 20 132 14 19 20	100 120	R, S-P-20 very small R, S-P-20?
4 5 6 1 2 3	0540 0547	STS-2 SV/H-1	SFT2 SFT5	132 17 44 45 132 17 44 45	105 105	small R, S-P-11 very small R, S-P-16
1 2 3 4 5 6	0547 0540	SV/H-1 STS-2	SFT5 SFT2	132 22 45 15 132 22 45 15	150 150	small R?, S-P-33 very small R, S-P-?
4 5 6 1 2 3	0540 0547	STS-2 SV/H-1	SFT2 SFT5	133 05 37 00 133 05 37 00	105 105	L, S-P-1 L, S-P-6
4 5 6 1 2 3	0540 0547	STS-2 SV/H-1	SFT2 SFT5	133 05 41 00 133 05 41 00	90 90	L, S-P-1 L?, S-P-1
4 5 6 1 2 3	0540 0547	STS-2 SV/H-1	SFT2 SFT5	133 11 09 15 133 11 09 15	90 90	small L, S-P-4 very small L, S-P-6
4 5 6 1 2 3	0540 0547	STS-2 SV/H-1	SFT2 SFT5	133 11 45 40 133 11 45 40	125 125	L, S-P-5 small L, S-P-6
4 5 6 1 2 3	0540 0547	STS-2 SV/H-1	SFT2 SFT5	133 12 05 00 133 12 05 00	4380 4380	large T, Alaska m6.4 h34 d32; strong P S surface ...
1 2 3 4 5 6	0547 0540	SV/H-1 STS-2	SFT5 SFT2	133 15 49 00 133 15 49 30	180 120	R, S-P-28, and aftershock? R, S-P-23

Table 3 (continued)

DAS Channels V N E	DAS S/N	Sensor Type	Site Name	Start Time (day hr min sec)	Length (s)	Event description(s) (L,R,T,V= Local, Regional, Teleseismic, Vehicular)
4 5 6	0540	STS-2	SFT2	133 19 17 45	75	very small L? or V?
1 2 3	0547	SV/H-1	SFT5	133 20 27 30	180	R, S-P~19?
4 5 6	0540	STS-2	SFT2	133 20 27 30	180	R?
4 5 6	0540	STS-2	SFT2	133 20 54 55	110	L, S-P~6
1 2 3	0547	SV/H-1	SFT5	133 20 54 55	140	very small R, S-P~12
4 5 6	0540	STS-2	SFT2	134 04 14 15	75	small L, S-P~3
1 2 3	0547	SV/H-1	SFT5	134 04 14 15	90	extremely small L, S-P~?
4 5 6	0540	STS-2	SFT2	134 04 50 30	90	small L, S-P~1?
1 2 3	0547	SV/H-1	SFT5	134 04 50 30	90	extremely small L?, S-P~?
4 5 6	0540	STS-2	SFT2	134 12 48 00	135	small R, S-P~19
1 2 3	0547	SV/H-1	SFT5	134 12 48 00	135	very small R, S-P~18
1 2 3	0547	SV/H-1	SFT5	134 17 24 20	210	small R?, S-P~25?
4 5 6	0540	STS-2	SFT2	134 17 24 20	210	very small R, S-P~?
4 5 6	0540	STS-2	SFT2	134 17 46 30	105	small R, S-P~10
1 2 3	0547	SV/H-1	SFT5	134 17 46 30	105	... S-P~12
1 2 3	0547	SV/H-1	SFT5	134 20 43 45	75	very small L?, S-P~?
4 5 6	0540	STS-2	SFT2	134 22 54 15	105	2(?)*L, S-P~?
1 2 3	0547	SV/H-1	SFT5	134 22 54 15	180	R, S-P~13
4 5 6	0540	STS-2	SFT2	134 23 45 00	195	L, S-P~7
1 2 3	0547	SV/H-1	SFT5	134 23 45 00	195	R, S-P~10
4 5 6	0540	STS-2	SFT2	135 00 31 30	120	L, S-P~6?, emergent and complex
1 2 3	0547	SV/H-1	SFT5	135 00 31 30	180	R, S-P~11
4 5 6	0540	STS-2	SFT2	135 00 43 00	135	L, S-P~6?, emergent and complex
1 2 3	0547	SV/H-1	SFT5	135 00 43 00	135	L, S-P~9
1 2 3	0547	SV/H-1	SFT5	135 00 52 20	90	small L, S-P~8
4 5 6	0540	STS-2	SFT2	135 00 52 20	90	small L?, S-P~5?
1 2 3	0547	SV/H-1	SFT5	135 01 01 20	120	2*L, S-P~8
4 5 6	0540	STS-2	SFT2	135 01 01 20	120	L's?
4 5 6	0540	STS-2	SFT2	135 01 15 15	105	L?, S-P~6
1 2 3	0547	SV/H-1	SFT5	135 01 15 15	105	R, S-P~10
4 5 6	0540	STS-2	SFT2	135 03 14 30	2250	2*T, Mexico m5.8 h33 d29 and m5.9 h33 d29; many phases and mixed
1 2 3	0547	SV/H-1	SFT5	135 03 14 30	2250	...
4 5 6	0540	STS-2	SFT2	135 04 26 20	700	small T, Mexico-Guatemala m4.8 h132 d34; only P
1 2 3	0547	SV/H-1	SFT5	135 04 26 20	700	...
4 5 6	0540	STS-2	SFT2	135 05 35 00	3810	small T, Kermadec m5.3 h33 d84; only P and surface (and aftershock?)
1 2 3	0547	SV/H-1	SFT5	135 05 35 00	3810	...

Table 3 (continued)

DAS Channels V N E	DAS S/N	Sensor Type	Site Name	Start Time (day hr min sec)	Length (s)	Event description(s) (L,R,T,V= Local, Regional, Teleseismic, Vehicular)
1 2 3	0547	SV/H-1	SFT5	135 08 18 35	205	R, California-Nevada border, m? h5 d1.9; S-P~25
4 5 6	0540	STS-2	SFT2	135 08 18 35	205	... S-P~30
1 2 3	0547	SV/H-1	SFT5	135 13 42 30	150	R? or V?, S-P~32
4 5 6	0540	STS-2	SFT2	135 18 49 35	90	L?, S-P~6
1 2 3	0547	SV/H-1	SFT5	135 18 49 35	90	R, S-P~10
4 5 6	0540	STS-2	SFT2	135 20 51 30	2820	small T, Pac-Ant Ridge m5.2 h10 d93; surface wave only
1 2 3	0547	SV/H-1	SFT5	135 20 51 30	2820	...
4 5 6	0540	STS-2	SFT2	135 21 39 00	75	very small L?
1 2 3	0547	SV/H-1	SFT5	135 21 39 00	75	...
4 5 6	0540	STS-2	SFT2	135 21 59 00	4260	large T, Andreanof m6.2 h33 d42; strong P S SS surface, also PcP+/-PP, 3 small aftershocks
1 2 3	0547	SV/H-1	SFT5	135 21 59 00	4260	...
1 2 3	0547	SV/H-1	SFT5	136 03 27 00	90	very small L?, S-P~?
1 2 3	0547	SV/H-1	SFT5	136 03 45 10	230	R, California-Nevada border, m? h5 d3.3; S-P~43
4 5 6	0540	STS-2	SFT2	136 03 45 10	230	small ... S-P~?
4 5 6	0540	STS-2	SFT2	136 06 40 00	1500	extremely small T, Mexico m4.6 h81 d35; weak surface wave only (barely worth keeping)
1 2 3	0547	SV/H-1	SFT5	136 06 40 00	1500	...
4 5 6	0540	STS-2	SFT2	136 09 31 45	1395	extremely small T, Mexico m4.6 h33 d35; surface wave only
1 2 3	0547	SV/H-1	SFT5	136 09 31 45	1395	...
4 5 6	0540	STS-2	SFT2	136 10 00 45	75	small L, S-P~2
1 2 3	0547	SV/H-1	SFT5	136 10 00 45	90	very small L, S-P~?
4 5 6	0540	STS-2	SFT2	136 15 33 30	75	very small L?
1 2 3	0547	SV/H-1	SFT5	136 15 33 30	75	extremely small L?
4 5 6	0540	STS-2	SFT2	136 20 00 20	100	L, S-P~5
1 2 3	0547	SV/H-1	SFT5	136 20 00 20	100	small L, S-P~8
4 5 6	0540	STS-2	SFT2	136 21 55 00	10380	large T, Tonga m6.1 h33 d72; strong P S, OK SKS, SS?, long surface with 360-72 degree round trip
1 2 3	0547	SV/H-1	SFT5	136 21 55 00	10380	...
1 2 3	0547	SV/H-1	SFT5	137 01 32 45	180	small R?
4 5 6	0540	STS-2	SFT2	137 01 32 45	180	very small R
1 2 3	0547	SV/H-1	SFT5	137 04 22 05	90	small L, S-P~9
4 5 6	0540	STS-2	SFT2	137 04 22 05	90	... S-P~6

Table 3 (continued)

DAS Channels V N E	DAS S/N	Sensor Type	Site Name	Start Time (day hr min sec)	Length (s)	Event description(s) (L,R,T,V= Local, Regional, Teleseismic, Vehicular)
4 5 6	0540	STS-2	SFT2	137 06 11 30	930	small T, AK m5.5 h33 d31; S and surface wave
1 2 3	0547	SV/H-1	SFT5	137 06 11 30	930	...
1 2 3	0547	SV/H-1	SFT5	137 06 30 45	150	small R, S-P-12
4 5 6	0540	STS-2	SFT2	137 06 30 45	150	... S-P-?
1 2 3	0547	SV/H-1	SFT5	137 08 45 25	255	R, Nevada m? h5 d3.3, S-P-44
4 5 6	0540	STS-2	SFT2	137 08 45 25	255	... S-P-53
1 2 3	0547	SV/H-1	SFT5	137 10 48 25	60	L?, S-P-0?
4 5 6	0540	STS-2	SFT2	137 10 57 40	95	L, S-P-5
1 2 3	0547	SV/H-1	SFT5	137 10 57 40	95	L, S-P-9
4 5 6	0540	STS-2	SFT2	137 16 15 00	5520	T, New Britain m5.8 h33 d91; P PP S-group(SKSac...) SP/PS SS SKKP/PKKS LR/LG
1 2 3	0547	SV/H-1	SFT5	137 16 15 00	5520	...
1 2 3	0547	SV/H-1	SFT5	137 18 10 30	90	very small L?, S-P-?
1 2 3	0547	SV/H-1	SFT5	137 23 09 15	75	very small L, S-P-0?
1 2 3	0547	SV/H-1	SFT5	137 23 21 00	3720	R, California-Nevada border (Eureka Valley) m6.0 h7 d2.8; S-P-42s and several large aftershocks
1 2 3	0547	SV/H-1	SFT5	138 00 56 30	240	R, California-Nevada border (Eureka Valley), m3.2 aftershock
1 2 3	0547	SV/H-1	SFT5	138 01 03 30	945	R, California-Nevada border (Eureka Valley), m4.5 aftershock
1 2 3	0547	SV/H-1	SFT5	138 01 22 20	745	R, California-Nevada border (Eureka Valley), several aftershocks
1 2 3	0547	SV/H-1	SFT5	138 01 44 00	180	R, California-Nevada border (Eureka Valley), aftershock
1 2 3	0547	SV/H-1	SFT5	138 01 53 30	180	Calibration
Spring-Summer Break						
1 2 3	7062	SV/H-1	SFT2	191 01 55 44	76	small CalNet L/R, m1.2 d0.18 h4.2, 37.22N-121.61E
1 2 3	7064	SV/H-1	SFT5	191 15 35 00	120	small R; S-P-15s
1 2 3	7062	SV/H-1	SFT2	191 15 35 00	120	
1 2 3	7064	SV/H-1	SFT5	191 20 48 02	1440	very small T, @COSTA RICA m5.2/5.4 d44 h33; P's only
1 2 3	7062	SV/H-1	SFT2	191 20 48 02	1440	... nothing seen
1 2 3	7062	SV/H-1	SFT2	192 09 30 00	120	small R; S-P-10s, m2.3 d0.79 h8.6, @36.58N-121.14E
1 2 3	7064	SV/H-1	SFT5	192 09 30 00	120	

Table 3 (continued)

DAS Channels V N E	DAS S/N	Sensor Type	Site Name	Start Time (day hr min sec)	Length (s)	Event description(s) (L,R,T,V= Local, Regional, Teleseismic, Vehicular)
1 2 3	7062	SV/H-1	SFT2	192 12 12 00	90	very small R; S-P-13s; m2.1 d2.04 h13.6, @38.04N-118.83E
1 2 3	7064	SV/H-1	SFT5	192 12 12 00	90	...
1 2 3	7062	SV/H-1	SFT2	192 13 47 30	2250	small T; NORTHERN CHILE m6.1 h33 d79; P's, S's, small L
1 2 3	7064	SV/H-1	SFT5	192 13 47 30	2250	...
1 2 3	7062	SV/H-1	SFT2	192 23 41 34	55	CalNet L/R, m2.2 d2.26 h2.9, @38.63N-119.66E; not seen
1 2 3	7064	SV/H-1	SFT5	192 23 41 34	55	... d1.73 ... was seen
1 2 3	7062	SV/H-1	SFT2	193 13 27 30	3600	nice T; HOKKAIDO, JAPAN m6.6 h17 d71; P's S's, relatively small L
1 2 3	7064	SV/H-1	SFT5	193 13 27 30	3600	...
1 2 3	7062	SV/H-1	SFT2	193 14 55 20	3000	T; SEA OF JAPAN m6.0 h33 d71; small P, no S, L
1 2 3	7064	SV/H-1	SFT5	193 14 55 20	3000	...
1 2 3	7062	SV/H-1	SFT2	193 16 11 20	2020	T; HOKKAIDO; JAPAN m5.9 h33 d71; P's, no S, very small L
1 2 3	7064	SV/H-1	SFT5	193 16 11 20	2020	...
1 2 3	7062	SV/H-1	SFT2	193 18 28 20	90	CalNet L/R, m2.2 d2.28 h3.1, @38.65N-119.63E; not seen
1 2 3	7064	SV/H-1	SFT5	193 18 28 20	90	very small CalNet L/R, m2.2 d1.76 h3.1, @38.65N-119.63E
1 2 3	7062	SV/H-1	SFT2	194 18 19 31	90	very small CalNet L/R, m2.4 d1.46 h10.1, @36.25N-120.38E
1 2 3	7064	SV/H-1	SFT5	194 18 19 46	90	small CalNet L/R, m2.4 d1.47 h10.1,
1 2 3	7064	SV/H-1	SFT5	195 03 09 29	60	CalNet L/R, m2.4 d1.52 h8.3, @37.62N-119.42E
1 2 3	7062	SV/H-1	SFT2	195 03 09 29	60	...; not seen"
1 2 3	7062	SV/H-1	SFT2	195 11 57 52	45	nice L, m1.9 d0.14 h4.3, 37.37N-121.31E
1 2 3	7064	SV/H-1	SFT5	195 11 57 52	45	...
1 2 3	7062	STS-2	SFT2	196 04 14 35	310	2*nice R, m3.1 d2.27 h1.9, @38.64N-119.65E; m3.0 d2.27 h3.3
1 2 3	7064	SV/H-1	SFT5	196 04 14 35	310	
1 2 3	7062	STS-2	SFT2	196 08 08 10	80	small CalNet L/R, m2.0 d0.71 h5.7, @36.62N-121.22E
1 2 3	7064	SV/H-1	SFT5	196 08 08 14	80	... d0.89 ...
1 2 3	7064	SV/H-1	SFT5	196 14 13 50	83	small CalNet L/R, m2.6 d1.52 h8.2, @37.62N-119.42E
1 2 3	7062	STS-2	SFT2	196 14 14 00	101	... d1.96 ...

Table 3 (continued)

DAS Channels V N E	DAS S/N	Sensor Type	Site Name	Start Time (day hr min sec)	Length (s)	Event description(s) (L,R,T,V= Local, Regional, Teleseismic, Vehicular)
1 2 3	7064	SV/H-1	SFT5	196 14 35 26	83	very small CalNet L/R, m2.2 d1.51 h7.0, @37.62N-119.42E
1 2 3	7062	STS-2	SFT2	196 14 35 26	83	very small CalNet L/R, m2.2 d1.95 h7.0, @37.62N-119.42E
1 2 3	7062	STS-2	SFT2	196 17 40 42	80	CalNet L/R, m2.4 d0.17 h4.7, @37.25N-121.64E
1 2 3	7064	SV/H-1	SFT5	196 17 40 46	80	CalNet L/R, m2.4 d0.35 h4.7, @37.25N-121.64E
1 2 3	7062	STS-2	SFT2	197 13 02 19	2111	very small T, @NEAR COAST OF NICRAGUA m4.8/5.1 d41 h33; sur- face wave only
1 2 3	7064	STS-2	SFT5	197 13 02 19	0	† ...
1 2 3	7062	STS-2	SFT2	197 21 32 56	2884	small T, @NEAR COAST OF CEN- TRAL CHILE m5.3/5.2 d84 h33; S's, surface; STS-2 anomalies
1 2 3	7064	STS-2	SFT5	197 21 32 56	0	† ...
1 2 3	7062	STS-2	SFT2	198 01 46 54	2231	small T, @NEAR COAST OF CEN- TRAL CHILE m5.4/5.0 d82 h33; S's, surface
1 2 3	7064	STS-2	SFT5	198 01 46 54	2231	...
1 2 3	7064	STS-2	SFT5	199 23 08 26	90	CalNet L/R, m2.8 d0.94 h18.8, @37.93N-120.27E
1 2 3	7062	STS-2	SFT2	199 23 08 37	90	... d1.45 ...
1 2 3	7064	STS-2	SFT5	200 07 38 37	86	small CalNet L/R, m2.2 d1.98 h3.4, @37.54N-118.83E
1 2 3	7062	STS-2	SFT2	200 07 38 47	104	... d2.40 ...
1 2 3	7062	STS-2	SFT2	200 08 35 58	85	very small CalNet L/R, m2.3 d1.72 h6.4, @38.50N-119.54E
1 2 3	7064	STS-2	SFT5	200 08 35 58	85	...
1 2 3	7064	STS-2	SFT5	200 12 32 13	90	small CalNet L/R, m2.9 d1.21 h39.4, @37.64N-119.81E
1 2 3	7062	STS-2	SFT2	200 12 32 23	90	... d1.67 ...
1 2 3	7062	STS-2	SFT2	200 16 00 54	120	2*very small CalNet L/R, m2.6 d1.76 h2.3 and m2.3 d1.76 h2.0, @38.83N-122.80E
1 2 3	7064	STS-2	SFT5	200 16 00 54	120	...
1 2 3	7062	STS-2	SFT2	201 07 48 43	5177	nice T, @SAMOA ISLANDS REGION m5.1/5.8 d73 h33; P's, S and SKS, long, ringy surface wave
1 2 3	7064	STS-2	SFT5	201 07 48 43	5177	...
1 2 3	7062	STS-2	SFT2	201 10 59 05	80	small CalNet L/R, m2.4 d0.78 h7.6, @36.58N-121.14E
1 2 3	7064	STS-2	SFT5	201 10 59 09	80	... d0.93 ...

Table 3 (continued)

DAS Channels V N E	DAS S/N	Sensor Type	Site Name	Start Time (day hr min sec)	Length (s)	Event description(s) (L,R,T,V= Local, Regional, Teleseismic, Vehicular)
1 2 3	7062	STS-2	SFT2	201 11 57 42	90	nice CalNet L/R, m3.0 d0.35 h4.3, @37.26N-121.64E
1 2 3	7064	STS-2	SFT5	201 11 57 42	90	...
1 2 3	7064	STS-2	SFT5	201 18 56 36	86	CalNet L/R, m2.7 d1.97 h3.6, @37.57N-118.84E
1 2 3	7062	STS-2	SFT2	201 18 56 45	104	... d2.40 ...
1 2 3	7062	STS-2	SFT2	201 21 00 11	0	†... d0.59 ...
1 2 3	7064	STS-2	SFT5	201 21 00 13	79	CalNet L/R, m2.1 d0.69 h9.6, @36.82N-121.28E
1 2 3	7062	STS-2	SFT2	201 23 56 03	95	CalNet L/R, m2.0 d0.88 h7.7, @36.55N-121.15E
1 2 3	7064	STS-2	SFT5	202 00 22 00	330	2*CalNet L/R, m3.6 d3.22 h4.6, @36.13N-117.69E
1 2 3	7062	STS-2	SFT2	202 00 22 00	330	... d3.30 ...
1 2 3	7062	STS-2	SFT2	202 20 42 48	92	very small CalNet L/R, m2.1 d0.49 h0.2, @37.32N-122.10E
1 2 3	7064	STS-2	SFT5	202 20 42 52	90	... d0.64 ...
1 2 3	7062	STS-2	SFT2	202 22 12 34	3062	T, @NEAR COAST OF GUATE- MALA m5.0/5.2 d36 h33; P's S's long surface
1 2 3	7064	STS-2	SFT5	202 22 12 34	3062	...
1 2 3	7062	STS-2	SFT2	203 04 38 33	75	CalNet L/R, m1.7 d0.14 h5.1, @37.37N-121.29E
1 2 3	7064	STS-2	SFT5	203 04 38 33	75	...
1 2 3	7062	STS-2	SFT2	203 05 05 37	3500	T, @NORTHERN COLOMBIA m6.0/5.9 d55 h19; P's L/R S's sur- face; and CalNet L/R, m2.2 d0.34 h6.2, @36.87N-121.59E
1 2 3	7064	STS-2	SFT5	203 05 05 37	3500	...; and ... d0.67 ...
1 2 3	7062	STS-2	SFT2	203 06 28 18	5560	T, @SOUTHERN EAST PACIFIC RISE m5.5/5.9 d92 h10; S's long sur- face
1 2 3	7064	STS-2	SFT5	203 06 28 18	5560	...
1 2 3	7064	STS-2	SFT5	203 10 58 16	90	CalNet L/R, m2.9 d1.93 h4.1, @37.55N-118.89E
1 2 3	7062	STS-2	SFT2	203 10 58 25	103	... d2.35 ...
1 2 3	7062	STS-2	SFT2	203 12 26 33	2203	small T, @MARIANA ISLANDS REGION m5.5/0.0 d80 h127; P's sur- face
1 2 3	7064	STS-2	SFT5	203 12 26 33	2203	...
1 2 3	7064	STS-2	SFT5	204 01 31 52	90	CalNet L/R, m2.7 d1.94 h5.2, @37.55N-118.89E
1 2 3	7062	STS-2	SFT2	204 01 32 01	103	very small ... d2.36 ...

Table 3 (continued)

DAS Channels V N E	DAS S/N	Sensor Type	Site Name	Start Time (day hr min sec)	Length (s)	Event description(s) (L,R,T,V= Local, Regional, Teleseismic, Vehicular)
1 2 3	7062	STS-2	SFT2	204 06 18 20	91	small CalNet L/R, m1.8 d0.19 h5.6, @36.99N-121.70E
1 2 3	7064	STS-2	SFT5	204 06 18 29	93	... d0.59 ...
1 2 3	7062	STS-2	SFT2	204 14 30 27	95	small CalNet L/R, m2.1 d0.89 h7.1, @36.55N-121.14E
1 2 3	7064	STS-2	SFT5	204 14 30 29	95	very small ... d0.97 ...
1 2 3	7064	STS-2	SFT5	204 16 48 27	4170	very small T, @FIJI ISLANDS REGION m5.3/0.0 d77 h632; surface
1 2 3	7062	STS-2	SFT2	204 16 48 27	4170	...?
1 2 3	7064	STS-2	SFT5	205 07 29 49	101	CalNet L/R, m2.6 d1.92 h13.8, @38.07N-119.01E
1 2 3	7062	STS-2	SFT2	205 07 30 00	104	very small ... d2.41 ...
1 2 3	7062	STS-2	SFT2	205 09 44 18	91	small CalNet L/R, m1.7 d0.20 h3.1, @37.16N-121.57E
1 2 3	7064	STS-2	SFT5	205 09 44 22	92	... d0.39 ...
1 2 3	7062	STS-2	SFT2	205 13 31 35	93	small CalNet L/R, m2.1 d0.61 h10.4, @37.75N-122.12E
1 2 3	7064	STS-2	SFT5	205 13 31 36	93	very small ... d0.67 ...
1 2 3	7062	STS-2	SFT2	205 20 35 58	2800	small T, @VANUATU ISLANDS m5.4/5.3 d84 h208; S/SKS, surface
1 2 3	7064	STS-2	SFT5	205 20 35 58	2800	
1 2 3	7062	STS-2	SFT2	206 20 05 52	92	small CalNet L/R, m1.8 d0.17 h6.5, @37.24N-121.62E
1 2 3	7064	STS-2	SFT5	206 20 05 56	92	... d0.35 ...
1 2 3	7062	STS-2	SFT2	207 03 09 56	94	CalNet L/R, m2.3 d0.35 h9.2, @36.85N-121.59E
1 2 3	7064	STS-2	SFT5	207 03 10 03	94	... d0.69 ...
1 2 3	7062	STS-2	SFT2	207 17 29 05	95	small CalNet L/R, m2.2 d0.81 h10.6, @37.71N-122.55E
1 2 3	7064	STS-2	SFT5	207 17 29 09	95	... d0.99 ...
1 2 3	7064	STS-2	SFT5	207 22 01 12	100	small CalNet L/R, m2.7 d1.72 h0.0, @38.63N-119.68E
1 2 3	7062	STS-2	SFT2	207 22 01 24	103	very small ... d2.24 ...
1 2 3	7064	STS-2	SFT5	207 23 47 16	103	extremely small CalNet L/R, m2.7 d2.32 h7.7, @37.51N-118.41E
1 2 3	7062	STS-2	SFT2	207 23 47 25	106	... d2.73 ...
1 2 3	7064	STS-2	SFT5	208 11 48 48	1632	very small T, @GULF OF ALASKA m5.0/4.5 d27 h10; surface wave only
1 2 3	7062	STS-2	SFT2	208 11 48 48	1632	
1 2 3	7064	STS-2	SFT5	208 17 21 00	420	Peculiar low-frequency; probably sur- face wave of m3.6 d3.73 h7.3, @40.30N-124.50E
1 2 3	7062	STS-2	SFT2	208 17 21 00	420	very small ... d3.78 ...

Table 3 (continued)

DAS Channels V N E	DAS S/N	Sensor Type	Site Name	Start Time (day hr min sec)	Length (s)	Event description(s) (L,R,T,V= Local, Regional, Teleseismic, Vehicular)
1 2 3	7064	STS-2	SFT5	208 21 17 45	1200	CalNet L/R, m3.9 d3.72 h8.0, @40.30N-124.49E; strong surface wave
1 2 3	7062	STS-2	SFT2	208 21 17 45	1200	... d3.78 ...
1 2 3	7064	STS-2	SFT5	209 05 56 30	2070	small T, OFF COAST OF MEXICO, m5.0/4.7 d31 h10; S, good surface
1 2 3	7062	STS-2	SFT2	209 05 56 30	2070	
1 2 3	7062	STS-2	SFT2	209 07 17 26	4354	very small T, @SOLOMON ISLANDS m5.2/5.4 d86 h70; surface wave
1 2 3	7064	STS-2	SFT5	209 07 17 26	4354	...
1 2 3	7062	STS-2	SFT2	209 12 43 23	150	CalNet L/R, m1.9 d0.40 h7.0, @36.82N-121.54E and m1.3 d0.40 h6.8, @36.82N-121.54E
1 2 3	7064	STS-2	SFT5	209 12 43 30	94	small ... d0.71 ..
1 2 3	7062	STS-2	SFT2	209 18 19 41	5060	T, @SOLOMON ISLANDS m5.4/6.0 d89 h33; SKS, S, surface
1 2 3	7064	STS-2	SFT5	209 18 19 41	5060	...
1 2 3	7064	STS-2	SFT5	209 20 59 31	101	CalNet L/R, m2.7 d1.93 h2.9, @37.56N-118.89E
1 2 3	7062	STS-2	SFT2	209 20 59 41	103	very small ... d2.35 ...
1 2 3	7064	STS-2	SFT5	210 04 36 00	3000	very small T, KERMADEC ISLANDS REGION, m5.1/4.8 d84 h35; surface wave
1 2 3	7062	STS-2	SFT2	210 04 36 00	3000	...
1 2 3	7064	STS-2	SFT5	210 07 47 05	102	small CalNet L/R, m2.8 d2.11 h6.2, @37.51N-118.67E
1 2 3	7062	STS-2	SFT2	210 07 47 05	102	very small ... d2.52 ...
1 2 3	7062	STS-2	SFT2	210 08 47 45	2355	small T, NEAR COAST OF GUATE- MALA, m4.8/0.0 d37 h33; P, S?, sur- face wave
1 2 3	7064	STS-2	SFT5	210 08 47 45	2355	...
1 2 3	7064	STS-2	SFT5	210 15 51 12	106	very small CalNet L/R, m2.6 d2.83 h2.7, @37.18N-117.80E
1 2 3	7062	STS-2	SFT2	210 15 51 12	106	
1 2 3	7062	STS-2	SFT2	210 20 21 38	1083	very small T, @GUERRERO, MEX- ICO m5.0/4.2 d27 h33; surface wave
1 2 3	7064	STS-2	SFT5	210 20 21 38	1083	...
1 2 3	7062	STS-2	SFT2	210 21 35 15	94	CalNet L/R, m2.0 d0.57 h8.0, @36.77N-121.28E
1 2 3	7064	STS-2	SFT5	210 21 35 15	94	... d0.73 ...
1 2 3	7062	STS-2	SFT2	211 01 07 48	1812	small T, @OFF COAST OF MEX- ICO m4.9/5.2 d33 h10; S, surface wave
1 2 3	7064	STS-2	SFT5	211 01 07 48	1812	...

Table 3 (continued)

DAS Channels V N E	DAS S/N	Sensor Type	Site Name	Start Time (day hr min sec)	Length (s)	Event description(s) (L,R,T,V= Local, Regional, Teleseismic, Vehicular)
1 2 3	7064	STS-2	SFT5	211 03 42 35	150	small CalNet L/R, m2.8 d2.74 h1.8, @37.21N-117.90E
1 2 3	7062	STS-2	SFT2	211 03 42 35	150	very small ... d3.12 ...
1 2 3	7064	STS-2	SFT5	211 07 26 10	120	CalNet L/R, m3.0 d1.76 h4.2, @38.69N-119.66E
1 2 3	7062	STS-2	SFT2	211 07 26 10	120	small ... d2.29 ...
1 2 3	7062	STS-2	SFT2	211 18 23 23	94	CalNet L/R, m2.0 d0.40 h6.7, @36.82N-121.54E
1 2 3	7064	STS-2	SFT5	211 18 23 23	94	small ... d0.71 ...
1 2 3	7062	STS-2	SFT2	212 09 37 40	1940	very small T, NORTHERN COLOM- BIA, m4.3/0.0 d53 h150; surface wave
1 2 3	7064	STS-2	SFT5	212 09 37 40	1940	...
1 2 3	7064	STS-2	SFT5	212 23 49 19	92	CalNet L/R, m2.7 d0.35 h7.2, @37.38N-121.73E (not seen)
1 2 3	7062	STS-2	SFT2	212 23 49 19	92	very small ... d0.35 ... (poor)
1 2 3	7062	STS-2	SFT2	213 01 16 08	2390	very small T, LEEWARD ISLANDS, m4.5 d56 h69; small surface wave
1 2 3	7064	STS-2	SFT5	213 01 16 08	2390	...
1 2 3	7064	STS-2	SFT5	213 20 41 45	390	small CalNet L/R, m3.7 d3.73 h8.0, @40.31N-124.49E; P, surface wave
1 2 3	7062	STS-2	SFT2	213 20 41 45	390	... d3.79 ...
1 2 3	7062	STS-2	SFT2	214 01 53 00	150	small CalNet L/R, m3.3 d3.78 h7.9, @40.31N-124.48E (missed on SFT5)
1 2 3	7062	STS-2	SFT2	214 03 24 47	2830	T, @KYUSHU, JAPAN m5.5/5.2 d84 h33; P?, S/SKS, small surface wave
1 2 3	7064	STS-2	SFT5	214 03 24 47	2830	...
1 2 3	7062	STS-2	SFT2	214 09 01 00	480	small CalNet L/R, m3.6 d3.72 h7.8, @40.31N-124.48E; surface wave
1 2 3	7064	STS-2	SFT5	214 09 01 00	480	...
1 2 3	7062	STS-2	SFT2	214 10 21 00	300	small CalNet L/R, m3.5 d3.78 h7.6, @40.31N-124.48E (missed on SFT5)
1 2 3	7062	STS-2	SFT2	215 07 22 37	5400	nice T, @QUEEN CHARLOTTE ISLANDS REGION m5.5/5.9 d15 h10; P, S, long surface wave
1 2 3	7064	STS-2	SFT5	215 07 22 37	5400	...
1 2 3	7062	STS-2	SFT2	215 12 56 35	4105	T, @EGYPT m6.1/5.7 d110 h10 (SKS?) and @QUEEN CHARLOTTE ISLANDS REGION m5.0/5.2 d15 h10 (larger signal; P, surface)
1 2 3	7064	STS-2	SFT5	215 12 56 35	4105	...

Table 3 (continued)

DAS Channels V N E	DAS S/N	Sensor Type	Site Name	Start Time (day hr min sec)	Length (s)	Event description(s) (L,R,T,V= Local, Regional, Teleseismic, Vehicular)
1 2 3	7062	STS-2	SFT2	215 17 45 25	95	very small CalNet L/R, m2.0 d0.96 h6.8, @36.56N-121.15E
1 2 3	7064	STS-2	SFT5	215 17 45 25	95	...
1 2 3	7062	STS-2	SFT2	216 11 46 00	9540	small T, @SOUTHERN SUMATERA, INDONESIA m6.1/6.4 d128 h33; PP, small SKS, long surface wave; (and T, @BANDA SEA m5.4/5.3 d113 h33, not seen); and T, @NEAR COAST OF NORTHERN CHILE m6.1/6.0 d79 h33, possibly seen
1 2 3	7062	STS-2	SFT2	216 14 37 02	104	small CalNet L/R, m3.0 d2.41 h9.8, @37.64N-118.84E
1 2 3	7062	STS-2	SFT2	216 20 20 07	99	small CalNet L/R, m2.7 d1.47 h4.9, @36.05N-120.60E
1 2 3	7064	STS-2	SFT5	216 20 20 07	99	very small ... d1.57 ... (seismologist at site)
1 2 3	7064	STS-2	SFT5	217 10 19 31	120	small CalNet L/R, m2.8 d2.00 h9.1, @37.55N-118.80E
1 2 3	7062	STS-2	SFT2	217 10 19 31	120	very small ... d2.43 ...
1 2 3	7064	STS-2	SFT5	217 12 05 15	5565	2 * small T, NEW BRITAIN REGION, m5.1/5.1 d91 h33; surface wave; and @FIJI ISLANDS REGION m5.3/0.0 d77 h616; P, S, SKS?, spectral-only surface wave 0.06 Hz
1 2 3	7062	STS-2	SFT2	217 12 05 15	5565	2 * very small ...; and ...; P
1 2 3	7062	STS-2	SFT2	217 18 36 44	93	small CalNet L/R, m1.9 d0.64 h9.5, @36.66N-121.29E
1 2 3	7064	STS-2	SFT5	217 18 43 58	94	small CalNet L/R, m1.9 d0.75 h0.9, @37.89N-120.51E
1 2 3	7062	STS-2	SFT2	217 18 44 10	0	†extremely small ... d1.27 ..
1 2 3	7064	STS-2	SFT5	218 00 31 45	150	CalNet L/R, m3.1 d2.06 h15.4, @39.39N-120.25E
1 2 3	7062	STS-2	SFT2	218 00 31 45	150	very small ... d2.55 ...
1 2 3	7064	STS-2	SFT5	218 11 01 42	98	small CalNet L/R, m2.6 d1.45 h8.4, @36.27N-120.36E
1 2 3	7062	STS-2	SFT2	218 11 01 42	120	... d1.46 ...
1 2 3	7064	STS-2	SFT5	218 23 34 31	93	CalNet L/R, m2.6 d0.54 h7.2, @37.61N-122.00E
1 2 3	7062	STS-2	SFT2	218 23 34 31	93	nice ... d0.48 ..
1 2 3	7062	STS-2	SFT2	219 00 12 21	6100	nice T, @NORTHEAST OF TAIWAN m6.0/0.0 d90 h158; P, pP, SKS (clearly split), S, sS, small surface wave
1 2 3	7064	STS-2	SFT5	219 00 12 21	6100	...

Table 3 (continued)

DAS Channels V N E	DAS S/N	Sensor Type	Site Name	Start Time (day hr min sec)	Length (s)	Event description(s) (L,R,T,V= Local, Regional, Teleseismic, Vehicular)
1 2 3	7062	STS-2	SFT2	219 06 16 00	1560	T, OFF COAST OF JALISCO, MEX- ICO m4.9/0.0 d24 h33; P, S, surface wave
1 2 3	7064	STS-2	SFT5	219 06 16 00	1560	...
1 2 3	7064	STS-2	SFT5	219 13 43 30	120	CalNet L/R, m3.5 d1.39 h18.4, @36.47N-120.17E
1 2 3	7062	STS-2	SFT2	219 13 43 30	120	... d1.48 ...
1 2 3	7062	STS-2	SFT2	219 18 03 00	15420	T, @SOUTH OF FIJI ISLANDS m6.0/0.0 d83 h560; P's, SKS/S's, non-iasp91 phases, weak surface wave; and weak CalNet L/R, m2.1 d1.39 h18.7, @36.46N-120.17E; and T, @HOKKAIDO, JAPAN REGION m6.1/5.9 d72 h33; Ps, S's, SS, core phases, long surface wave
1 2 3	7064	STS-2	SFT5	219 18 03 00	15420	...
1 2 3	7062	STS-2	SFT2	220 08 45 55	23650	T, @SOUTH OF MARIANA ISLANDS (GUAM) m7.0/8.0 d85 h61; P's, S's, SS, non-iasp91 phases, core phases, LONG surface wave (maybe with 2nd-round); and CalNet L/R, m3.8 d2.84 h0.0, @37.07N- 117.80E
1 2 3	7064	STS-2	SFT5	220 08 45 55	23650	...
1 2 3	7062	STS-2	SFT2	220 20 14 40	4340	T, @MARIANA ISLANDS m5.3/5.8 d84 h60; P's, S's, core phases, surface wave
1 2 3	7064	STS-2	SFT5	220 20 14 40	4340	...
1 2 3	7064	STS-2	SFT5	221 02 19 15	195	CalNet L/R, m3.0 d2.07 h15.4, @39.39N-120.24E; and CalNet L/R, m3.0 d2.05 h17.3, @39.38N-120.26E
1 2 3	7062	STS-2	SFT2	221 02 19 15	195	small ... d2.55 ...; and very small ... d2.54 ...
1 2 3	7064	STS-2	SFT5	221 02 49 00	900	Unidentified surface wave (similar to offshore northern California)
1 2 3	7064	STS-2	SFT5	221 05 32 45	150	CalNet L/R, m3.0 d2.46 h2.5, @37.85N-118.25E
1 2 3	7064	STS-2	SFT5	221 09 26 42	4160	T, @MARIANA ISLANDS m5.3/5.5 d84 h60; P's, SKS/S, surface wave

Table 3 (continued)

DAS Channels V N E	DAS S/N	Sensor Type	Site Name	Start Time (day hr min sec)	Length (s)	Event description(s) (L,R,T,V= Local, Regional, Teleseismic, Vehicular)
1 2 3	7064	STS-2	SFT5	221 11 50 00	12300	T, @HINDU KUSH REGION, AFGHANISTAN m5.8/0.0 d105 h230; PP, SKS; and T, @HINDU KUSH REGION, AFGHANISTAN m6.3/0.0 d105 h233; Pdiff, sPdiff, PP, sPKiKP, unknown phase@13:04:30, sSKSac, unknown phase@13:11:20, core phases, unknown phase @13:17:04, unknown phase @13:26:10, unknown phase@13:27:07, unknown phase @13:32:40 (maybe odd surface wave); and small CalNet L/R, m2.1 d0.74 h18.2, @38.11N-121.88E
1 2 3	7064	STS-2	SFT5	221 21 40 18	94	CalNet L/R, m2.1 d0.69 h5.4, @36.84N-121.57E
1 2 3	7064	STS-2	SFT5	221 23 08 41	99	very small CalNet L/R, m2.1 d1.53 h12.9, @37.69N-119.41E
1 2 3	7064	STS-2	SFT5	222 01 04 57	15603	T, @OFF West COAST OF S. ISLAND, New Zealand m0.0/7.1 d105 h33; Pdiff, PKiKP/PP, SKS's, possibly sSdiff, SP/PS, core phases, large unknown phase@01:34:35, surface wave
1 2 3	7064	STS-2	SFT5	222 05 52 56	101	CalNet L/R, m2.6 d1.98 h0.1, @37.57N-118.84E
1 2 3	7064	STS-2	SFT5	222 09 04 50	12970	T, @MARIANA ISLANDS m5.5/0.0 d85 h60; P's, possibly S, weak surface wave; and T, @NORTH ISLAND, NEW ZEALAND m5.8/6.0 d95 h33; P's, PP, PKiKP, SKS/S's, SP/PS, SS, core phases, large unknown phase@10:28:50, surface wave

Annotations of the form "m5.8/[6.0] d95 h33" mean, for example, "body- [and surface-] wave magnitude of 5.8 [6.0], distance of 95°, hypocentral depth of 33 km". The order of these groups is variable, but the units are consistent. For CalNet events, the coda M_L magnitude is given.

"?" means uncertainty.

"Event description(s)" annotation are somewhat inconsistent over the duration of the experiment, tending toward more detail (and more of it automatically generated) as time went on.

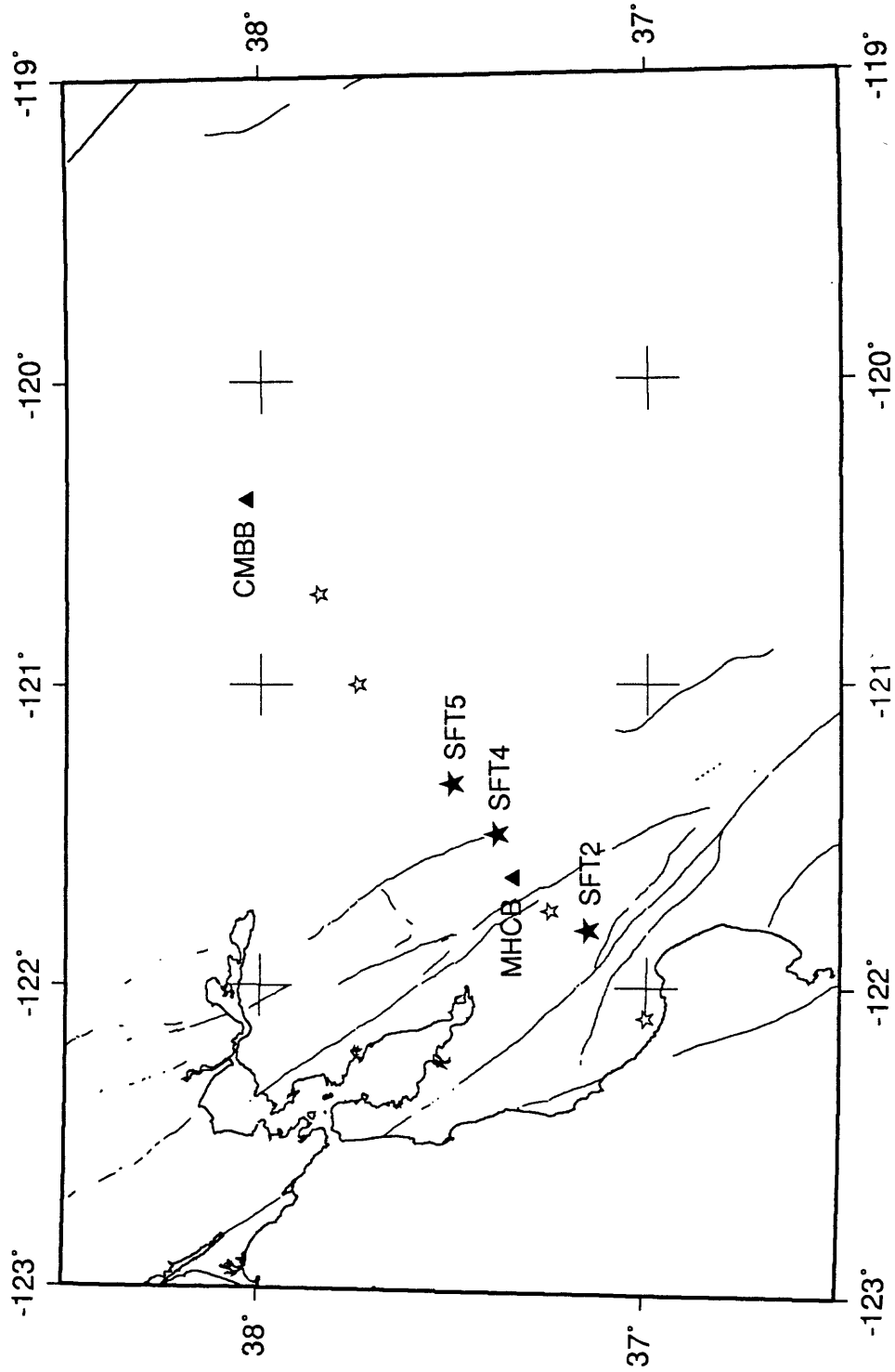


Figure 1a. Broadband sites along our seismograph array. Solid stars are the three existing USGS vaults. Open stars are the remaining sites proposed but not yet built. Solid triangles are the two U.C. Berkeley broadband sites within the array.

BROADBAND VAULT DESIGN

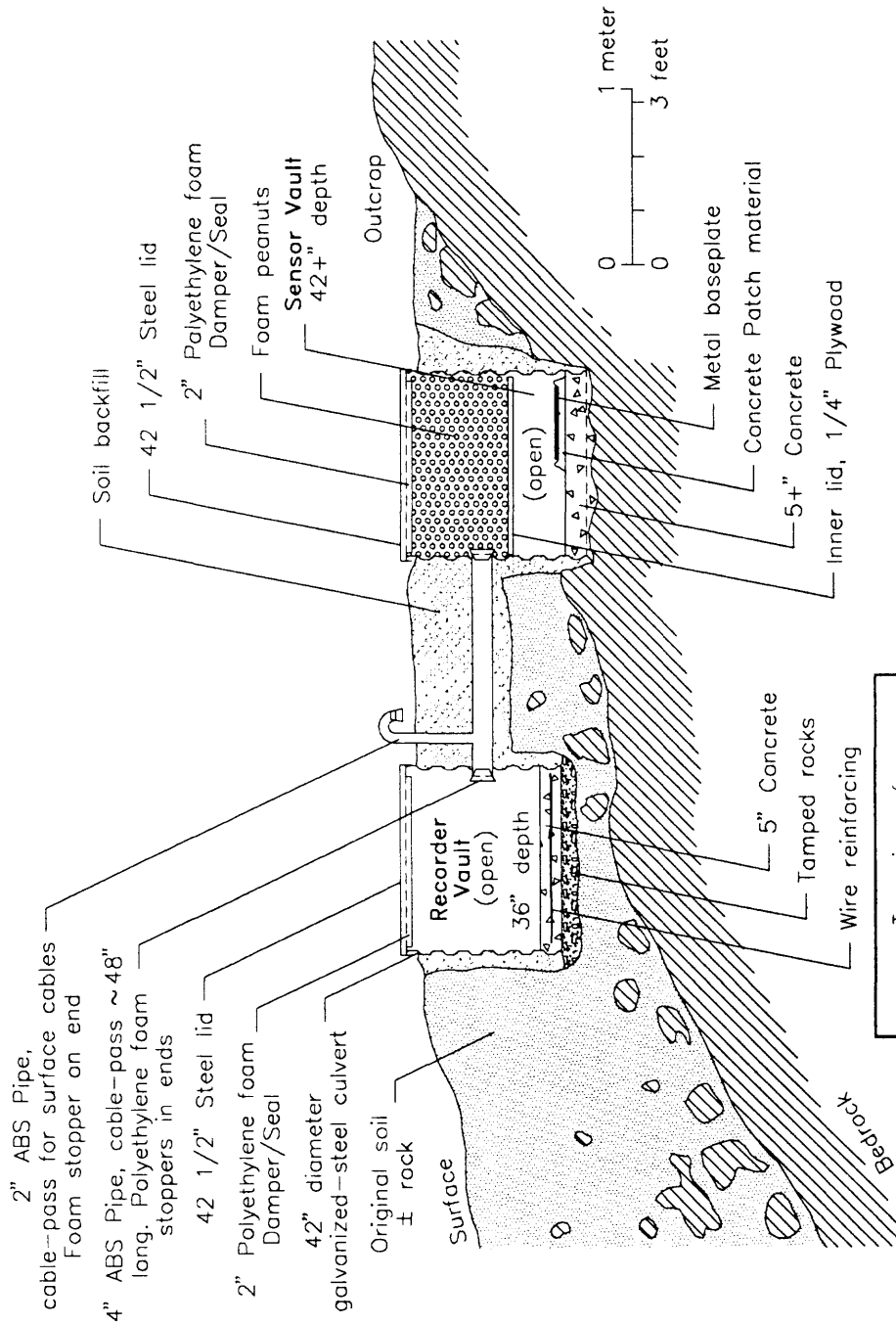
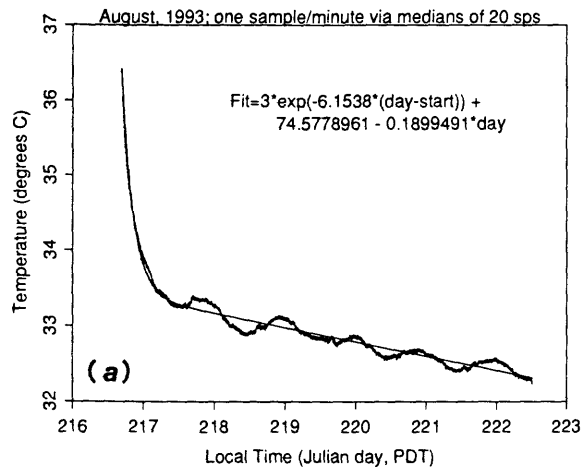


Figure 1b. Diagram of the secure winter-rated vaults we developed for this study. Early versions had only 36-inch deep sensor vaults and lacked the 2-inch ABS cable-pass to the surface. Barbed-wire fence surrounds the site wherever livestock-incursions might occur. Inset shows plan surface view of recorder vault, with chain and padlock passing through lid handles and down into basal concrete on both sides of culvert. Sensor-vault inner-lid legs not shown.

Ingram Canyon Vault, STS-2 Dome Temperature



Ingram Canyon Vault, STS-2 Dome Temperature

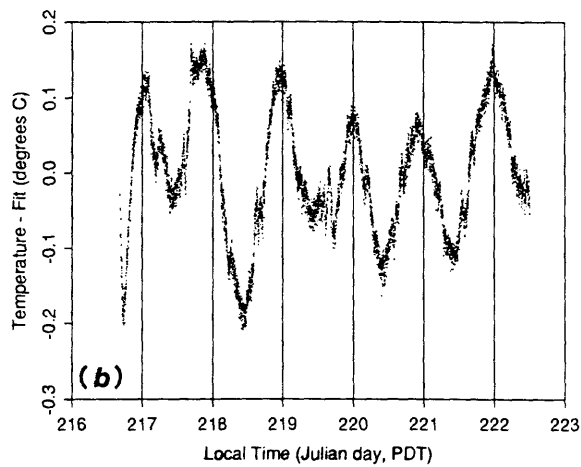


Figure 2. Temperature variation of Streckeisen™ STS-2 dome in the Ingram Canyon vault, SFT5, over a 5.8-day period following vault closure. This vault was 36 inches deep and shaded by wooden pallets and a tarp. (a) Raw dome temperature and a hand-fit linear-plus-exponential function. The fit function removes the exponential cool-down following vault closure, and removes a slow, possibly weather-induced decline. (b) Raw temperature minus the fit, hence the diurnal cycle of the sensor dome.

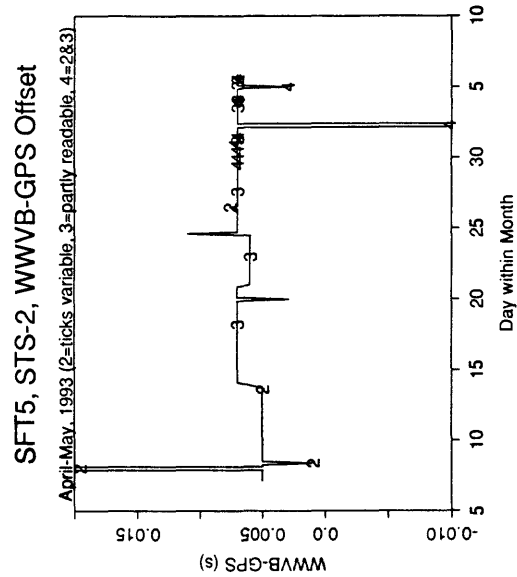
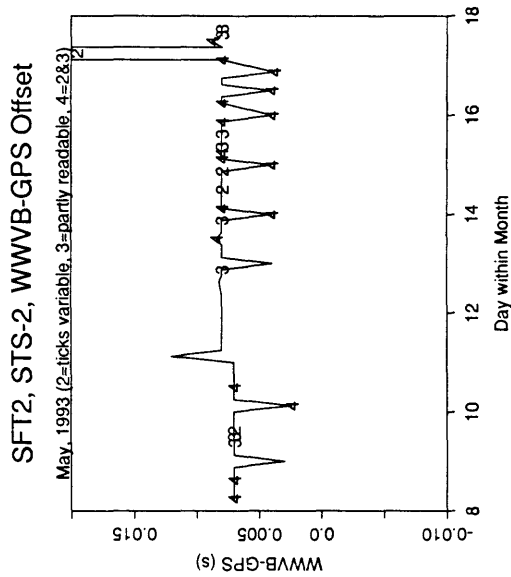
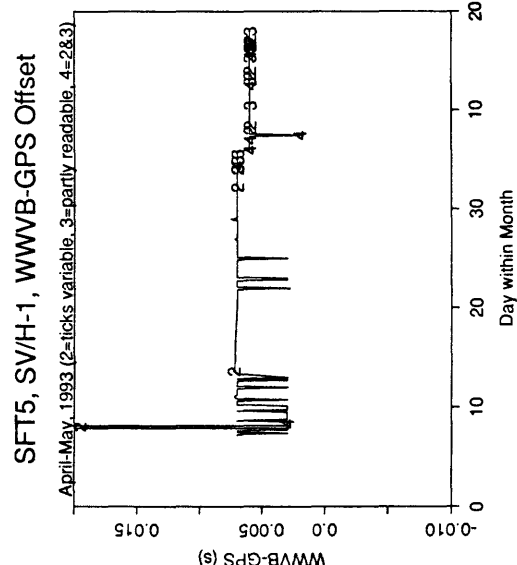


Figure 3. WWVB time versus GPS time. WWVB time-code was sampled at 250 sps (4 ms sample interval) and read with program *ahwvvb*, a robust signal processor and interpreter (Appendix F). GPS was programmed for a 5-minutes-per-hour duty cycle (extendable to 20 minutes in bad signal conditions), and was used to correct the internal TCXO of the RefTek™ DAS. In our experience, the Trimble™ GPS receiver rarely failed to obtain a good signal lock at our open hill-top sites. The digits in the body of the plots derive from *ahwvvb* error messages and warnings ("2" means the interval between second marks varied significantly during the 80-s WWVB sample; "3" means the time code was only partly decipherable; "4" means both conditions apply). Note that, apart from an advance of about 5 ms in WWVB, the only large deviations between the time standards are accompanied by the warning that the interval between second marks varied significantly during the WWVB sample—an indication of a badly corrupted WWVB signal.



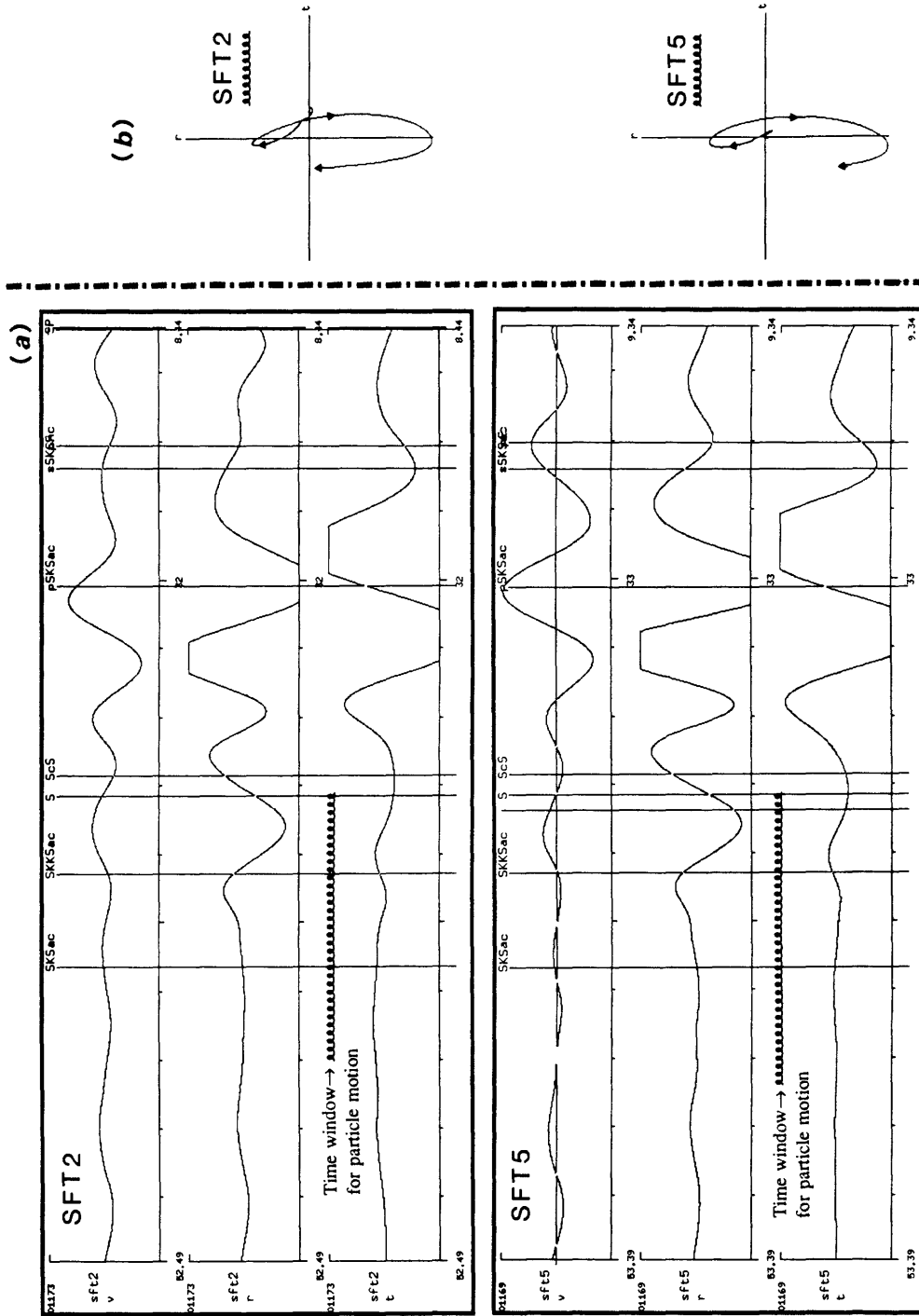


Figure 4. A clear example of splitting—the large Taiwanese earthquake (1993:219:00:12:21). (a) About 135 s of the seismograms, the shear phases with their *isp91* times identified. (b) The polarization diagrams, showing horizontal particle motion in a time window from before SKS_{ac} up to the time of S. Data are from STS-2 sensors at SFT2 (New Almaden) and SFT5 (Ingram Canyon). Splitting of these core phases is apparent—the polarization diagrams in (b) would be linear and in the radial, "r", direction if SKS_{ac} and tSKS_{ac} were not split. The degree and orientation of the splitting is outwardly similar between the two stations, though SFT5 is 70 km northeast of the San Andreas fault and 40 km northeast of the Calaveras fault. SFT2 lies between those two faults.