NASA Technical Memorandum 104579

Radar Ocean Wave Spectrometer (ROWS) Preprocessing Program (PREROWS2.EXE)

User's Manual and Program Description

Charles R. Vaughn

January 1993

NASA Technical Memorandum 104579

Radar Ocean Wave Spectrometer (ROWS) Preprocessing Program (PREROWS2.EXE)

User's Manual and Program Description

Charles R. Vaughn
Laboratory for Hydrospheric Processes
Wallops Flight Facility
Wallops Island, Virginia

National Aeronautics and
Space Administration
Wallops Flight Facility
Wallops Island, Virginia 23337

1993

TABLE OF CONTENTS

1.0 INTRODUCTION

The Radar Ocean Wave Spectrometer (ROWS) is an airborne radar system that measures the backscattered radar signal from the ocean's surface. Jackson et al. (1985) describe the radar subsystem, while Ward (1992) describes the recent modifications to the data acquisition subsystem. The data acquisition subsystem records ROWS data and auxiliary data in a sequence of 1024 byte records on an 8 mm digital tape using an EXABYTE 8200. Each of the 1024 byte records includes a header, 700 bytes of radar signal, time, the radar antenna rotation angle (shaft angle), aircraft pitch and roll from the aircraft INS, and the time delay between the time of radar pulse firing and the time recording starts.

PREROWS2 (for PREprocessing of ROWS data, version 2) is a compiled pc program written in Microsoft QuickBasic 4.5. The program allows previewing a ROWS data tape for data quality, copying of appropriate sections of data to a DOS file, and the subsequent reviewing of these DOS files. PREROWS2 is a modified version of PREROWS and is written for the ROWS data acquisition subsystem as improved for use with the Grand Banks ERS-1 SAR validation mission. Most of the features of PREROWS2 can be used with the earlier data. However, there will be some problems with the data presentation; as will be readily evident if the program is used for these earlier missions.

The copy feature provides for transferring a sequence of records from the 8 mm tape to a DOS file on a pc. This is usually advantageous for intensive data review and processing because of the relatively slow output speed of the EXABYTE. In addition, if the operator wants to step backwards through a file, the EXABYTE 8200 is prohibitively slow, whereas the operation is very quick with a random access device.

The location of the aircraft as determined by a GPS receiver, and some of the outputs from the aircraft INS system, were available for recording. These data are not yet recorded directly into the ROWS data stream; rather they are recorded independently on a separate pc. In each data record, more than 150 unused bytes exists that can be used for ancillary data. A provision exists in PREROWS2 to merge the separately recorded data with the ROWS data onto a new data tape.

The user should remember that PREROWS2 is not a polished commercial program. There are many areas where, time and value permitting, improvements can be made. In addition, each mission usually has its own peculiarities that require program modifications. The discussion that follows highlights some of the general features that can be improved and some of the specific code that is deficient in some way. Program areas that may require continuing maintenance, such as the MERGE feature are discussed in detail.

2.0 USER INSTRUCTIONS

PREROWS2 operates on either an EXABYTE file taken with the ROWS radar, or on a ROWS file copied from the original 8 mm tape to a pc hard disk. This program is menu driven with the

following primary functions:

1. Produce a time log of an EXABYTE tape.

2. Locate any record within a ROWS file that exists on tape or in a DOS file.

3. Copy a block of records from an EXABYTE tape to a DOS file.

4. Convert time and aircraft pitch and roll on an 8 mm tape file to ASCII values and output to a DOS file. The output is an average of 100 successive pulses.

5. Produce successive screen displays of radar waveforms annotated with ancillary data such as aircraft pitch and roll, radar antenna rotation angle, radar trigger delay, and both system time and pc "tick" time.

6. Produce a "waterfall" screen display of radar waveforms.

7. Merge an original ROWS 8 mm tape with INS, GPS, and other data onto a second 8 mm tape.

These functions are explained more fully below.

[NOTE: PREROWS2 implements procedures for use with the ERS-1 SAR validation mission. The earlier missions SAXON and SWADE should be processed with PREROWS, the predecessor to the present version. If PREROWS2 is used with these earlier missions, the user should be aware that certain features won't work and others might show spurious results.]

When first run, PREROWS2 displays an information box and ten small highlighted boxes at the bottom. These boxes represent the ten function keys **FK1** through **FK10**. This highest level menu allows the user to select the source of the data file. Use **FK2** or **FK3** to select a DOS or EXABYTE file respectively. The information box is blank in this menu because no data source has been selected.

[NOTE: The EXABYTE must be turned on before pressing FK3. If an EXABYTE is not on, the computer will lockup and have to be rebooted.]

Two other function keys are active: **FK9** presents a display of the version and date of the program being run; **FK10** exits the program and returns operation to DOS.

After selecting a file source, new function key selections are shown. These new selections allow a user to perform several activities. The selections available from the highest level DOS and EXABYTE menus are described next.

2.1 DOS menu

The sole purpose of this menu is to allow opening or closing a file, or to calling another menu where actual processing can be done. When first entering this menu from the main menu (or after selecting **FK9** at this level), the information box is essentially blank. The record pointer is at -1 to indicate that no DOS file is open. The ten function keys now provide the following choices:

FK1: MainMnu

Returns to the MAIN MENU.

FK2: OpnFile

Allows operator to open a DOS data file. The program stays at this menu level but now displays the complete file path and name, file starting and stopping times, the total number of records in the file, the present record number, and the time associated with this record. The record number cannot be changed at this menu level. **FK3** is used to change the record number.

[NOTE: The program error trapping for entry of the name of a non-existent file is faulty. The program will recover properly by hitting the ENTER key a number of times.]

FK3: PreProc

Goes to another menu that allows selection of the record number for processing, display of the radar waveform in oscilloscope fashion, and display of the radar waveform in a color coded waterfall display.

FK4 - 8: NOT USED

FK9: ClsFile

Closes any open DOS file.

FK10: EXIT

Exits the program and returns to the DOS prompt.

2.1.1 DOS menu FK3 functions

The DOS PreProc function key accesses another screen that defines a new set of operations for the keys. Two of these operations display the radar waveform on the computer screen. The

function keys provide the following operations:

FK1: DosMen

Returns to the DOS menu.

FK2: NOT USED

FK3: Rec #N

Selects a record number at which to start other operations.

FK4: NOT USED

FK5: Wtrfal

Produces a waterfall display of ROWS waveforms starting with the active record. Three pieces of information are needed after making this selection: the starting and ending byte numbers for the display and which of six (at present) color selections to use for the display. The color selections will be discussed below. The starting and ending byte numbers are referenced to the radar waveform itself; ie., byte 1 is the first radar waveform byte (which will be byte 51 in the actual record); the highest byte number allowed is 700.

The maximum number of bytes that can be displayed on each waveform is 420. If the difference between the stopping and starting points on the waveform is greater than 420, the end of each waveform will be dropped.

FK6: DspDat

Displays a single ROWS waveform and annotates it with ancillary data. The operator can select to step singly through a file by pressing the keyboard **f** key; or, by pressing **ALT-f** the program steps automatically through the file. Once past the first record, the file can be displayed in reverse order by pressing the keyboard **b** key to backup a single record, or **ALT-b** to continuously display backwards through the file.

After pressing **FK6**, you are asked to enter the first and last bytes for display. The byte range for a waveform is from one to 700. The speed of display depends on the number of bytes to be shown.

[NOTE: There is no error trapping for an input byte number greater than 700; the program will crash if the ending byte is greater than 700. Also, there is no error trapping if the starting byte number is greater than the ending

byte number.]

FK7 - 9: NOT USED

FK10: EXIT

Exits the program and return to the DOS prompt.

2.2 EXABYTE menu

This highest level EXABYTE menu differs from the highest level DOS menu in that processing can be initiated; tape logging and tape merging are done here. The processing activities that parallel the DOS activites are accessed by the PreProc key, just like they are under the DOS menu. The ten function keys now provide the following:

FK1: MainMnu

Returns to the MAIN MENU.

FK2: PreProc

Goes to another menu that allows selection of the file number and record number for processing, display of successive radar waveforms in oscilloscope fashion, display of radar waveforms in a color coded waterfall display, copying a block of records from the EXABYTE to a DOS file, decoding and copying time and aircraft pitch and roll angles to a DOS file, and merging an original EXABYTE data tape with ancillary data (GPS, INS etc. data) to a second EXABYTE.

FK3 - 4: NOT USED

FK5: LogExb

Logs all files on a tape. The log gives the starting and ending time of each file and the number of records in each file. The information for each file includes the starting and starting times of each continuous sequence of records within the file and the number of records in the sequence. The output goes to a DOS file and the screen. See section 2.5 for more details.

FK6: NOT USED

FK7: Merge

Merges INS, GPS, and other ancillary data with ROWS data on an EXABYTE

tape. The merge requires an ASCII file with file name information and radar gain settings. The merge is done to a second EXABYTE. See section 2.3 for details.

FK8: NOT USED

FK9: Initize

Reinitializes the EXABYTE tape deck.

FK10: EXIT

Exits PREROWS2 and return to the DOS prompt.

2.2.1 EXABYTE menu FK2 functions

This selection provides functions that parallel those of **FK3** under the DOS menu.

[NOTE: The first time this menu appears after a new tape has been put in the EXABYTE, FK2 should be selected and a file number entered. This should be done even if FILE #1 is being used and a one appears as the file number in the information box. In certain circumstances, the record number counter is off if this selection isn't done first.]

Besides those functions that parallel the DOS functions, this selection provides for copying a block of records from the tape to a DOS file, and for creating a time sequence of a 100 pulse average of aircraft pitch and roll that is output to a DOS file.

FK1: MainMnu

Returns to the previous menu

FK2: File N

Moves the readhead to the beginning of file number N on the tape. If there are less than N files on the tape, the read head SHOULD be positioned to the end of the last file on the tape. The menu display should show the correct file number, stopping time, and total records for the file. The file starting time will not be shown until record 1 is selected and the tape is actually positioned there.

[NOTE: If the tape is positioned to a file other than #1, and FK2 is chosen AND FILE #1 then chosen, a tape rewind will be performed and all EXABYTE pointers reinitialized. This is a good way to reset pointers if the record number or file number showing on the display don't seem to be

correct.]

FK3: Rec #N

Selects the record number within a file. If the selected number is greater than the number of records in the file, the tape will stop at the end of the last record in the file, the total number of records displayed, and the ending time of the file displayed correctly.

[NOTE: If a record number is chosen that is less than the present record number, the EXABYTE will move backward VERY SLOWLY to the proper record number. Always move forward through a tape unless it is absolutely necessary to do otherwise.]

FK4: CpyFile

Copies a sequence of records from an 8 mm tape to a DOS file. You will be asked to supply a starting and ending record number for copying.

The starting record number can be before the number where the tape drive readhead is currently located. If it is, read the NOTE above for FK3.

FK5: WtrFall

Produces a color coded waterfall display of the received waveform signal strength. The display starts with the current record number. Three pieces of information are needed: the starting and ending byte numbers for the display and the color selection (one of six) to use for the display. The color selections will be discussed below. The starting and ending byte numbers are referenced to the radar waveform itself; ie., byte 1 is the first radar waveform byte, (byte 51 in the actual record).

The maximum number of bytes (radar samples) that can be displayed on each waveform is 420. If the difference between the stopping and starting bytes on the waveform is greater than 420, the waveform will be truncated.

FK6: DspData

Displays a single ROWS waveform and annotates it with ancillary data. The first waveform will be the record where the read head is presently located. Read the section above for DspData using a DOS file for additional information.

The display can be paused at the end of a waveform by pressing the keyboard p. Any subsequent key will resume plotting. Anytime a plot is being produced, you can abort plotting and return to the calling menu by pressing ESC.

[NOTE: Plotting does not terminate properly at the end of a file. Plotting continues beyond the end-of-file mark (or beyond the end of data if no EOF is present) and the record counter continues incrementing. If such a situation arises, hit ESC. It will then be necessary to reset everything by either going back to file #1 or by returning to the previous menu and doing a RESET.]

FK7: PtchRol

Converts time, and aircraft pitch and roll in an 8 mm tape file to ASCII values and outputs to a DOS file. Pitch and roll are averaged for one second.

FK8 - 9: NOT USED

FK10: EXIT

Exits PREROWS2 and return to the DOS prompt.

2.3 Merge GPS, INS and ROWS data

During the ERS-1 SAR validation mission, GPS and INS data were collected and saved in pc files separate from the actual ROWS data. Additionally, a notebook was used to record information that is necessary for processing the radar data. The original ROWS data is saved in a continuous sequence of 1024 byte records on EXABYTE tapes. The records have sufficient unused bytes that they can accommodate the relevant parts of the GPS, INS, and notebook data.

These instructions pertain to the MERGE feature of the program PREROWS2.BAS. This feature provides for merging data from an original ROWS Exabyte tape with INS and GPS data that are in pc files. The merged files are stored on tape on a second Exabyte. In addition to the INS and GPS data, there is a provision to include a mission name, date and takeoff time, along with a time bias between the clocks that recorded time for the INS file and the ROWS file. Finally, there is a provision to include IF gain data during the merge.

2.3.1 General merging instructions

The following five steps are necessary to do a proper merge:

1. Set up two EXABYTE tape drives for use.

   Two EXABYTE drives are needed on the same SCSI line. The drive from which the original ROWS data is being read must be set to **SCSI address 5**. The drive to which the merged data is written must be set to **address 0**.

2. Create a merge information file.

Create a file that contains information about the mission. This file tells the merge routine where to find the other data files and provides additional information that can be used for later tape identification and data processing. A sample MERGE information file looks like this:

```
ERS-1 Nfnd* Mission name (10 characters maximum)
11\14\91* Mission date
23:35* Mission takeoff time (UTC)
D:\ERS1\1114\PM\1114P.INS* INS data file name
D:\ERS1\1114\PM\1114P.GPS* GPS data file name
35* INS ahead of ROWS by 35 seconds (decimal secs)
23:59:10    28*  *
01:21:30    26* Table for total system IF gain.
01:22:26    29*  *
```

A minimum of seven lines MUST be present for a good merge and they must be in the order shown above. Some of the lines may be left blank, but they must be included. A more detailed description of the seven lines follows:

Line 1: An appropriate name, no longer than 10 character, can be given to the mission. It is useful to give a name so that the merged tape can be identified at a later date if its label is lost.

Line 2: The mission date should be the date corresponding to the UTC of the airplane takeoff time.

Line 3: The airplane takeoff time is in UTC. It doesn't need to be very accurate, but it should be earlier than the first ROWS data.

Lines 4 & 5: These are the pc file names of the GPS and INS data respectively. It is a good idea to include the complete DOS path with the file name. It is not necessary to have INS or GPS data. Just leave blank lines where necessary.

Line 6: The ROWS pc and the INS pc each records time from its own clock. If there is a bias between these two clocks it can be corrected during the merge. In general, the bias cannot be determined without comparing the pitch or roll data from the two sources and determining any phase shift between them.

Lines 7 -: Starting at line 7 ROWS the total system IF gain can be included with the merge. The time of a gain change is recorded in the first column and the gain in the second. There must be at least one space between the two columns. The first gain is the gain when the mission begins. If the first gain isn't known, enter a -99. A maximum of 100 gain changes per

mission can be accommodated.

The program requires that the first times in the INS and GPS files and the aircraft takeoff time in the Information file not differ by more than $1^h\ 59^m$.

It is not necessary to include the comments (or the asterisk, if there is no comment). However, **at least one space must exist after the last character of the actual data (or file name)** even if no comment exists. If some of the data is missing it is probably a good idea to include a few spaces and then an asterisk and comment to the effect that there is no data. At least one line must exist that represents system gain, even if the gain is unknown.

3 Insure that the INS data that covers the period of time of the 8 mm tape is in one pc file,

The INS data for a single flight must be in a single pc file. If the data is located in several file, it is necessary to concatenate the files into one file. It is important that the files be concatenated in correct time sequence.

4. Edit the GPS data file to eliminate "spurious" records,

The GPS data for a single flight must be in a single pc file. If the data is located in several file, it is necessary to concatenate the files. It is important that the data be concatenated in correct time sequence. An occasional GPS file will have what appears to be a spurious line at the top of the data. **Any spurious data lines must be removed** before the program will work correctly.

5. Run the merge program.

a. Run PREROWS2.

b. From the MAIN MENU select Function Key 3 (EXABYTE).

c. From the EXABYTE menu select Function Key 7 (MERGE).

d. Enter the name of the MERGE INFORMATION FILE.

The program will now merge the original ROWS tape with the available data identified in the merge information file. Each ROWS record is read and decoded. Ancillary data is similarly read and decoded and then time synchronized with the ROWS record. Be aware that this process takes about 10 times longer than the time it took to record the original data. Thus, if there are several hours of data on the original tape, merging will take 10 times several hours!

Each merged tape file(s) record includes the following:

1. Raw, unprocessed, ROWS radar return;

2. Antenna gain (IF attenuation);

3. "Corrected" antenna shaft angle;

4. "Corrected" pc real-time without the spurious FF FF bytes that occurred;

   [NOTE: After this program was written it was discovered that other spurious real-time words can occur. The user should be aware of the possibility they may exist.]

5. GPS latitude and longitude;

6. INS block of 10 words presently stored on floppy disk;

7. pc "tick time" stored as an eleventh word with the INS data;

8. A "NO DATA" indicator (FF FF or -32768) for missing data;

9. A modified header that properly reflects the above record structure.

10. The mission name and date.

    [NOTE: The mission name and date are stored in the last 18 bytes of each record. The existence of this information is not recognized by the header. See section 4.0 below for more information.]

After the last record of the original data tape is read and written to the merged tape, an EOF will be written to the new tape.

2.3.2 INS data

INS data from a digital Litton 92 was recorded on a separate pc during the ERS-1 SAR underflights. A DAS-429PC/HC interface card from Excalibur Systems Inc was used in the pc. The 32 bit word format in the pc file is dictated by the Excalibur interface. The specification is taken from page 9 of the DAS-429PC/HC manual.

The format of a 32 bit INS data word adheres to the ARINC standard; 20 bits represent data, 4 bits represent status, and parity, and the remaining 8 bits specify a data label. The DAS-429PC/HC reformats each ARINC word as follows:

BYTE 1		BYTE 2		BYTE 3		BYTE 4	
pc bit	ARINC bit	pc bit	ARINC bit	pc bit	ARINC bit	pc bit	ARINC bit
7 Data	13	7 Label	01	7 Sign	29	7 Data	21
6 Data	12	6 Label	02	6 Data (MSB)	28	6 Data	20
5 Data (LSB)	11	5 Label	03	5 Data	27	5 Data	19
4 Data (or SDI)	10	4 Label	04	4 Data	26	4 Data	18
3 Data (or SDI)	9	3 Label	05	3 Data	25	3 Data	17
2 SSM status	31	2 Label	06	2 Data	24	2 Data	16
1 SSM status	30	1 Label	07	1 Data	23	1 Data	15
0 Parity status	32	0 Label	08	0 Data	22	0 Data	14

Each data word is a maximum of 20 (or 18) bits plus the sign bit. Within the 32 bit word, these 20 (or 18) bits are obtained by taking bytes 3, 4 and 1 in that order with the most significant bit (MSB) as the sign of the data. The data bits need to be right justified in the new 32 bit data word. The 20 bit words use bits 4 and 3 of byte 1 as the least significant bits of the new 32 bit data word. For the present data set, only BCD latitude and longitude use the 20 bit word format. The 10 ARINC data words, and their word order on disk are:

decimal label	hex	description	# (bits)	Range	Resolution	
8	08	BCD latitude	22	-90,+90	0.1	min
9	09	BCD longitude	22	-180,+180	0.1	min
202	CA	grnd speed (bin)	15	0 - 4095	0.125	kts
203	CB	track angle	15	-180,+180	0.0055	deg
204	CC	true heading	15	-180,+180	0.0055	deg
212	D4	pitch angle	15	-180,+180	0.0055	deg
213	D5	roll angle	15	-180,+180	0.0055	deg
241	F1	altitude	20	-131072, 131072	0.125	ft
214	D6	pitch rate	15	-128,+128	0.0055	deg/sec
215	D7	roll rate	15	-128,+128	0.0055	deg/sec

Time is recorded with each INS record. This time is the "tick" time in the pc that records the INS data. The INS pc real-time clock is set each day by voice call from one of the airplane crew who reads a master clock. The time used is UTC. At the time the real-time clock is set, the "tick" time is automatically reset to the correct count by the computer. During a mission, the same procedure is followed simultaneously with the ROWS computer clock. However, during data reduction the time synchronization between the ROWS and INS data has to be done using the real-time clock in the ROWS data records. The "tick" time from the ROWS pc cannot be used because it loses excessive amounts of time through interrupts during the actual data acquisition process.

2.3.3 GPS Latitude and Longitude data

PREROWS2 assumes that the GPS data file is formatted using the NMEA Global Positioning

Satellite Position data format, which is ASCII. A typical line of data in the GPS NEMA format reads:

$GPGGA,124559.0,4855.9774,N,05434.0569,W,1,6,01.4,+99,M,+000,M

A comma delimits the data fields. The second field is time in hours, minutes, and seconds, without delimiters. The third field is latitude, the fourth indicates whether latitude is north or south. The fifth field is longitude, the sixth indicates whether longitude is east or west. Latitude and longitude are in degrees and minutes with no delimiter between. The above line is interpreted as 12 hours 45 minutes, 59.0 seconds with a position of 48 degrees 55.9774 minutes North latitude, 054 degrees 34.0569 West longitude. The remaining fields are ignored by this program.

Data in the GPS pc file is updated once per second on the GPS second. At present (1992) GPS time is running ahead of UTC by 8 seconds. This time difference must be compensated for when merging GPS with other data sets having time recorded as UTC. This time difference is written into the subroutine **TimeGPS$**.

The MERGE subroutine converts GPS latitude and longitude into minutes of arc as a long signed integers (32 bits) with the least significant bit representing 0.0001 minutes of arc. Thus, a sample GPS output

4855.9774,N,05434.0569,E,

is converted to 32 bit integers as

(48 x 60 + 55.9774) x 10^4 x (+1) = 29359774
(54 x 60 + 34.0569) x 10^4 x (-1) = -32740569

With the first byte in a ROWS record defined as number one (as opposed to zero), latitude and longitude are merged with the 1024 byte ROWS record. Latitude starts at byte 769 and longitude starts at byte 773.

Before the GPS data can be integrated with the ROWS data, time needs to be synchronized properly between the two data sets. GPS data is recorded once per second; ROWS 100 times per second. The merged tapes will have GPS data recorded only once per second; the intervening GPS values are represented by a "no data" word (FF FF FF FF). This procedure presents a problem for later data processing. Section 4.0 discusses the problem.

2.4 Waterfall display color codes

The radar data is digitized by an 8 bit A/D converter, thus giving a range of integers from 0 to 255. Within the QuickBasic 4.5 language, a VGA monitor can display 16 colors. PREROWS2 has six color selections, each defining the integer intervals to be displayed by the 16 colors. The

subroutine **Levels** defines the integer ranges for the six selections. The choice of levels for the integer ranges was based on actual waterfall plots; the choices being made based on the visual appearance of the plot. Table I shows the color selections and the associated integer ranges that go with each color:

TABLE I

COLOR	COLOR SELECTIONS 1	2	3	4	5	6
Black	0-8	0-9	0-10	0-10	0-15	0-10
Blue	9-14	10-11	11	11-14	16-30	11-12
Green	15-16	12-13	12	15-16	31-50	13-14
Cyan	17-18	14-15	13	17-18	51-60	15-16
Red	19-21	16-17	14	19-20	61-70	17-18
Magenta	22-24	18-19	15	21-22	71-80	19-20
Brown	25-26	20-21	16	23-24	81-85	21-23
White	27-29	22-23	17	25-26	85-90	24-27
Gray	30-32	24-25	18	27-28	91-95	28-30
Lt Blue	33-37	26-27	19-20	29-30	96-100	31-35
Lt Green	38-42	28-29	21-22	31-33	101-110	36-40
Lt Cyan	43-48	30-31	23-24	34-35	111-120	41-45
Lt Red	49-56	32-33	25-26	36-45	121-130	46-50
Lt Magenta	57-63	34-35	27-40	46-55	131-140	51-60
Yellow	64-70	36-50	41-60	56-80	141-220	61-75
Bright White	70-255	51-255	61-255	81-255	221-255	76-255

2.5 LOG TAPE

One of the first things to do after a mission is to produce a listing of the times data was acquired. This is accomplished with the LOGEXB function key (**FK5**) in the first EXABYTE menu. The tape log subroutine skips in increments of 100 records when looking for time breaks. This saves considerable time while doing the log because only one out of 100 records has to be decoded. Because the program skips 99 records, a time break in a recording sequence can only be determined to within ±49 records. In addition, for the worst case, the number of records in a sequence can be off by as much as 198 records, or about two seconds in time. Since the log is only used to indicate the approximate location of breaks in the data, this imprecision shouldn't

cause a problem. If more precision is required in locating such breaks, it can be obtained by stepping through a file one record at a time using **FK3** (Rec #N) and looking for time jumps on the display. The following is a sample of a tape log:

FILE NUMBER: 1

Start Time pc tick	Stop Time pc tick	Start Rec #	Stop Rec #	Number Recs
01:39:37	01:39:34.02	1	2	2
01:39:45.23	01:51:58.54	101	73302	73202
01:54:31.45	02:00:23.08	73401	108461	35061

End of file.
Start Time: 01:39:37 Stop time: 02:00:23
Total Records: 108561
END OF DATA.

[NOTE: The program does not give the exact numbers for the Stop Record # and Record length (ie., number of records) for the last continuous sequence of records in a file. If there is only one long sequence of non-broken continuous records in a file, these numbers MAY be correct. In any case, the Starting Time, Stopping Time, and Total Records information at the end of each file should be correct. If any numbers are incorrect, the errors should be small; ie., record numbers may be in error by one or two and the time by less than 0.1 secs.]

There is a potentially serious problem with the ROWS data recording that will not be easily detected with the logging subroutine. The subroutine relies on the pc "tick" timer to determine the time between two records that are 100 records apart. However, the "tick" timer looses time if another device interrupts the computer from updating the timer. At the same time, the computer interrupt can prevent new records (radar pulses) from being recorded. If this happen, there will still be 100 records per second recorded, but the real time gap will be greater than one second. This happens, in particular, when the EXABYTE tries to write to a bad spot on tape. The write time can exceed many timer counts. The seriousness of this problem has not been fully explored.

3.0 RECORD FORMAT

ROWS records contains a header, the backscattered radar signal from a single radar pulse, and a variety of ancillary data. The structure of each record consists of a header terminated with two ff(hex) bytes, followed by data in the same sequence as the definitions in the header. The header is composed of a sequence of pairs of two byte (16 bit) integers that designate the type of data recorded (first integer) and the number of bytes allocated to that data type (second integer).

3.1 Raw data record format

The following data types and lengths are defined for the raw recorded data starting with the ERS-1 mission:

TYPE	LENGTH (bytes)	TYPE DEFINITION
0000	50	The header
0001	700	ROWS radar return waveform
0002	4	Pc clock "tick time"
0003	2	Waveform digitizer trigger delay
0004	5	Time from pc real-time clock
0006	2	Aircraft roll angle
0005	2	Aircraft pitch angle
0007	1	Antenna shaft "angle" (No data recorded)
0008	2	Antenna shaft angle
0009	4	Observer latitude
0010	4	Observer longitude
0011	80	INS data
ffff	0	Empty data type (header terminator)
	856	(total bytes presently defined in the 1024 byte record)

The header itself uses the first 50 bytes of each record (two bytes each for the 12 data types plus two bytes for the terminator). After the header are the 806 data bytes defined by the header. The remaining 168 (1024 - 856) bytes are not used. Note that the data types do not need to be recorded in a monotonically increasing order; but, if the order is changed, the header also needs to be rearranged to reflect this new order.

The data types have the following formats:

TYPE description

0000 The header

0001 ROWS radar return waveform.

700 radar samples from an 8 bit A/D converter. The samples are saved as successive 8 bit binary integers.

0002 Pc clock "tick time".

Number of pc clock "ticks" since midnight as determined by the data acquisition computer. There are approximately 18.20648 clock ticks per second. The result is stored as a 32 bit (4 byte) binary integer.

0003 Waveform digitizer trigger delay.

The time delay from the radar pulse transmission ("main bang) to the start of waveform digitizing. The delay is determined with a 10 MHz clock (thus providing a resolution of 100ns) and is recorded as a 16 bit integer. This delay corresponds to the range from the aircraft to the first recorded point on the ROWS radar return waveform.

0004 Time from pc real-time clock.

A five byte field is reserved for time, although only three bytes are used for pc time. The encoding format is binary coded decimal (BCD). Byte 5 (right most) represents hours; byte 4, minutes; byte 3, seconds. Bytes two and one are not used (they are reserved for the possibility that another time code may be used in place of the pc real-time clock.) The actual data encoding is for the two nibbles of each byte to be the binary representation of the two digits of each time unit.

0006 Aircraft roll angle.

12 bit integer representing 1/(2^12) of 360 degrees or 0.08789 th of a degree.

0005 Aircraft pitch angle.

12 bit integer representing 1/(2^12) of 360 degrees or 0.08789 th of a degree.

0007 Antenna shaft "angle"

This data type is a carryover from earlier ROWS missions when the full shaft angle word was not available. At present, nothing is recorded here, although the header indicates otherwise.

0008 Antenna shaft angle.

The rotation angle of the ROWS antenna is obtained from a 14 bit shaft angle encoder whose output is in BCD format with a resolution of 0.1 degrees. The four nibbles represent hundreds of degrees, tens of degrees, degrees, and tenths of a degree. Bits 15 and 16 are not used by the encoder.

[NOTE: Bit 15 flags which antenna is used for that record. The altimeter horn is indicated by a 1, the scanning antenna by a zero (0).]

0009 Observer latitude.

Not yet directly available to ROWS.

0010 Observer longitude

Not yet directly available to ROWS.

0011 Inertial Navigation System (INS) data.

Not yet directly available to ROWS.

[Note: This data type represents more than one piece of data. The INS data type has its own internal structure which, for the merged data, at least, contains eleven variables.]

3.2 Data types for modified data

In principle, the integer designating a data type can be arbitrary within the range of the two bytes allocated for it. However, it seems useful to reserve a group of integers for a specific class of data. The classes suggested are "raw ROWS collected data", "raw ROWS data after some type of preprocessing", "raw ancillary data merged after flight with the ROWS data", and "merged ancillary data that has been processed". Various miscellaneous data types can be assigned numbers outside the range of numbers assigned to the broader classes.

Type range	Description
00000 - 00998	Original ROWS collected data.
01000	Designates ASCII text that is entered either during, or after, the mission for purposes of annotation.
00999	Flags a file of original data that has been selected so that it only contains a time contiguous sequence of consistent "good" data. Length is 0
01001 - 09999	Raw ancillary data.
10000 - 10999	Original ROWS data after some type of preprocessing. The last three digits should be the same as those of the original data. The particular type of processing can be indicated by the second digit from the left. This typing method evidently allows 10 different methods for processing a particular data type before some other arbitrary type needs to be assigned.
21001 - 29999	Processed ancillary data of types 01001 - 09999.

The actual data types, length (in bytes), and formats used for the merged data are:

TYPE	LENGTH	DESCRIPTION
01001	1	Antenna gain in dB (IF attenuation) from flight log sheets.
01009	4	Latitude from GPS. A signed long integer with South latitude represented by a negative number. The integer will represent one-ten thousandth of a degree of arc. Thus, one degree will be represented as 100000. The largest number that needs to be represented is ± 90 degrees, which is ± 9000000 in the suggested format.
01010	4	Longitude from GPS with West longitude represented as a negative number. The largest number will be ± 90 degrees. See latitude format for more information.
01011	80	INS data including independent pc 'tick' time. This data type contains a block of eleven words, ten taken from the LTN-92/ARINC-429 digital output from the aircraft INS, and one taken from the pc "tick" timer in the pc that recorded the ARINC data. The data format will be the same as recorded on the disk file (ie., no format changes are made).
10020	4	pc real-time clock data with spurious hex FF FF words set to the correct time. A spurious FF FF time word can appear at the start of a new second of time. It is not clear which second (the previous one or the new one) this belongs with. However, at a ROWS update rate of 100 pulses per second, the assignment of the spurious FF FF to the previous time will be adequate.
10021	1	Antenna shaft angle with the one degree bit corrected for dropouts. Several bits from the shaft angle encoder don't operate. This data type results from a correction algorithm for the angle. The accuracy of the correction remains to be determined. However, it is probably better than ± 0.4 degrees of arc. Bit 15 still flags the antenna type that obtained the data for the record.

4.0 DISCUSSION

PREROWS2.EXE is a "working" program, subject to modifications as the need arises. As such it is an evolutionary product. The immediate predecessor to this program, PREROWS, started in late 1990. Because of the evolutionary nature of the program, there are obvious pieces of unneeded code, non-uniformity of code structure, half written subroutines, and other symptoms of non-top down development. Future modifications of a substantive nature will primarily be needed only as dictated by a change in the actual ROWS hardware. It should be mentioned that PREROWS is still needed for looking at the data tapes collected prior to August of 1991.

There are several additions to PREROWS2 that might be effected for future missions, even if no overall hardware changes occur. Starting with the ERS-1 SAR validation mission, the ROWS transmitter/receiver switches, on a pulse-to-pulse basis, between a fixed nadir pointing antenna (altimeter mode) and the off-angle scanning antenna. Thus, a ROWS data file has altimeter mode data interleaved with azimuthally scanning mode data. PREROWS2 thus displays interleaved waterfall or waveform displays that are difficult to interpret. For both the waterfall and waveform displays, a choice should be given to present the data from one antenna or the other.

At present the altimeter mode data is separated from the azimuthally scanned data using a stand alone program that splits the data into two pc files. Each file can then be looked at using PREROWS2. The ability to separate these two data types into two files should be made part of the COPY feature (FK4: CpyFile, under section 2.2.1 above).

The ROWS data acquisition software does not presently put an EOF at the end of a recorded file; nor does it insert an EOF whenever the radar is shut off but the tape recorder left on. (This causes a large time gap between successive records.) Because of this, a single ROWS tape file can be many hundreds of megabytes long with time gaps occurring throughout. Since there are not presently any ROWS tapes with multiple files on them, PREROWS2 software hasn't been tested for possible "bugs" while using a multiple file tape. In addition, the merge feature of PREROWS2 does not have the capability of merging more than one file onto a single tape. If future missions have multiple files on a tape, PREROWS2 will have to be modified accordingly.

Most of the subroutines in the present program do not read the header to determine which data types are present; they assume a series of specific data types in a specific order. A more flexible program would read the header and decode the data as indicated by the data type. The tapes produced with the MERGE feature have several data types that differ from the raw data. PREROWS2 processes the data properly because the modified record contains the same variables (corrected for errors, etc.) with the same lengths and in the same places as the original header defines. The main purpose of defining these particular data types is to make some hypothetical future user of the tape aware that s/he is not working with the raw data.

The method used to merge GPS data with the ROWS data has one major drawback. GPS data is only merged with the record that corresponds to the closest exact second at which it is recorded. Thus, there are 99 records without GPS data for each one with data. Later, when data

is separated into two files, one with altimeter data, the other with the scanning antenna data, there can be very long stretches without GPS data in one of the two files. In fact, if there were an exact even number of pulses per second, all GPS data would be merged into one or the other of the separated file, leaving one file without any GPS data.

For future ROWS operations, more attention should be given to properly using the header in each record. To date, most of the data analysis software doesn't anticipate future needs. Most of this software demands a header with 50 bytes and data located exactly where the original ROWS acquisition software put it and with the same format. This rigidity defeats the purpose of having a header. For instance, the merge program allows a project name and date to be put in a data record. Unfortunately there is no room within the 50 header bytes to flag the existence of this information. The merged records have a project name and date located in bytes 1007 through 1024; unacknowledged by the header and potentially unappreciated by some future user of the tapes. In addition, several of the variables in a merged tape record can have a varying type throughout the file. This will happen, for instance, with the real-time when the FF FF has been corrected, or with the shaft angle for those angles that have been corrected.

The tape logging subroutine should be changed to use anything other than the "tick" timer as a time source. One approach would be to use the real-time clock and count the number of pulses that occur between two successive seconds. If the count deviated by more than some allowable amount (± 2) then the data for that second could be discarded.

5.0 PROGRAM DESCRIPTION

PREROWS2.EXE is a Microsoft QuickBasic 4.5 compiled program. The Microsoft library BRUN45.LIB and a library of routines to interface with the EXABYTE 8200 tape deck have been linked with the PREROWS2 source code to produce a single program. More than 90 subroutines are comprised in the program. Access to the EXABYTE 8200 is provided through the library of QuickBasic routines written by APtek, Inc. of Rockville, MD.

The main program defines the data structures, declares the subroutines and functions, dimensions the global variables (including four variables from the two data structures), five constants, and two string variables, and then calls one subroutine in an infinite loop.

One data structure (**FileStats**) holds the filename (for a DOS file), a file number (for a tape file), the current record number, the total records in a file, a pointer to the present byte in a record, and the starting and ending times of a file.

The second data structure (**Dta**) defines variables that hold the decoded data from a 1024 byte data record. As presently written, this data structure rigidly defines the variable types and lengths. If any of the data string lengths change (for instance, if the header length changed from 50 bytes, or the INS data were longer than 80 bytes), then this data type would have to be changed and the program recompiled.

Most of the subroutines have some internal documentation. This section will only highlight those routines most likely to be affected by future changes in the ROWS hardware. For the most part, the discussion of the subroutines follow alphabetically.

FUNCTION ADTriggerDelay returns the time delay between the time of transmission of a radar pulse and the start of the A/D converter. The delay is returned as meters below the airplane. The actual delay is determined by a simple counter. There is a count bias of 74 units written into the program to accommodate a hardware induced bias of the same amount. If the hardware bias is changed, this subroutine will have to have its bias changed.

SUB AscIIPitchRoll provides a time sequence of aircraft pitch and roll from a file on 8 mm tape. Pitch and roll are averaged over 100 records and output to a DOS file a one sample per second. The average can be altered by changing the value of the variable NAve% in the subroutine.

FUNCTION ClockTime$ presently returns **Dta.TrueTime**, which is the time as recorded from the pc real-time clock. Most routines that need time call **ClockTime$**. If a time source other than the pc real-time clock is available for display or data processing, it is only necessary to have **ClockTime$** return this new time. This indirect way of accessing time saves the labor of searching all routines for a specific time (**Dta.TrueTime**, for instance) and replacing that specific time with the new time.

Dta also provides the variables **Dta.TickTime**, which is the time returned by the pc "tick" timer, and **Dta.AccurateTime**. This latter time was provided by an external time code generator for missions prior to the ERS-1 SAR Validation Mission, and did not exist after August, 1991. Since **Dta.AccurateTime** was recorded in the same byte locations **Dta.TrueTime**, it is necessary to insure the correct time is used with the particular mission be looked at. The forerunner program to PREROWS2 uses **Dta.AccurateTime**.

SUB DataTypes is part of a package of subroutines that will be used to read a header and process the data according to the data types specified by that header. It is incomplete and not used.

SUB DecodeHeader translates the header string into a two dimensional short integer array with each row having the numerical data type and the number of bytes in that type.

SUB DecodeRecord assigns the individual variables in a ROWS record to the data structure **Dta**.

SUB DosFile defines the highest level DOS menu.

SUB DosPreProcessing defines the menu that calls some of the actual data manipulation subroutines. There are presently five undefined function keys in this menu that can be used to perform additional operations.

SUB ExbFile is the companion subroutine to **DosFile** and defines the highest level EXABYTE menu.

SUB ExbPreProcessing is the companion subroutine to **DosPreProcessing**. There are only two undefined function keys available for additional operations.

SUB ExbStatus is provided to interpret all the status codes return by the EXABYTE drive after it has been accessed. This routine has not been completed and is not presently use.

SUB ExbTapeLog has been explained in some detail in section 2.5 above.

SUB FileN moves the EXABYTE readhead to the file number that is entered. At present there is only one file on any ROWS 8mm tape. If a larger file number is chosen the tape drive will search until the end-of-data (EOD) is encountered.

SUB GPSDisplay simply prints, if present, the GPS latitude and longitude on the radar waveform displays. At present GPS only exists in a ROWS record if it has been merged with the record.

SUB GPSLatLong strips GPS latitude and longitude from a GPS NEMA data format record. Details of the GPS format and other related information is contained in section 2.3 above

FUNCTION IntergerTIME converts a time string that is in hours, minutes, and seconds with delimited by colons (:) into seconds as a long integer.

SUB MERGE should be useable to correct errors in ROWS data and save the corrected records to the new tape - even if no INS or GPS files are available for merging. PREROWS2 hasn't been tested to see if all subroutines will work properly if no INS or GPS file names are available. However, if there are program "bugs" that prohibit "merging" in this manner, they should be easy to fix. With the present structure of PREROWS2 it seems **MERGE** is the most logical place to do real data preprocessing in a bulk mode.

FUNCTION NameDATE is used by **MERGE** to put a mission name and date onto the new tape As noted in section 4.0 this information "hangs out" in a region of a ROWS record that is not recognized by the header as having information.

FUNCTION NewHEADER rigidly defines the header structure of a merged data tape. As mentioned in several sections above, this rigidity should be removed in some future version of this program.

SUB ParameterDisplay recognizes the GPS merged data type and displays the data, if present The subroutine needs to have code added to recognize the other data types present on a merged tape, and to read and display the mission name and date, if added.

FUNCTION PitchRoll has a 120 degree bias "correction" written in because of miswiring of the shaft angle encoder. This subroutine will have to be changed if the wiring is corrected.

SUB PlotRadReturn sets up the logic for using either a DOS file or EXABYTE file, for moving stepping single or continuously through a file, or for stepping forwards or backwards (for a DOS file). The actual plotting is done in the two subroutines **StepForwards** and **StepBackwards.**

SUB PresentFile prints the file status information to the screen. The routine doesn't presently recognize the merged data types.

FUNCTION ShaftAngle partially corrects for a serious error in the recorded data from the shaft angle encoder. The correction is only performed in the **MERGE** subroutine.

FUNCTION TimeGPS has an 8 second subtraction from GPS time to synchronize it with UT. This correction may have to be updated for future missions.

SUB Waterfall plots very slowly. If increased speed is required it may be necessary to write an assembly language routine to write directly to video ram. This routine should be written so that it can be linked with PREPROWS2 during compilation.

6.0 REFERENCES

Jackson, F. C., W. T. Walton, and P. L. Baker. 1985. "Aircraft and Satellite Measurements of Ocean Wave Directional Spectra Using Scanning-Beam Microwave Radars. J.G.R., **90**: 987-1004.

Ward, J. L. 1992. **A PC-Based Data Acquistion System as Applied to the Radar Ocean Wave Spectrometer**, NASA Technical Memorandum 104560.

APPENDIX I: PROGRAM SOURCE CODE

This section contains the PREROWS2 source code. Additional comments have been added to the original program. These are printed in a bold italic font.

```
'   PREROWS2.BAS   (PREprocessing of ROWS data)                [crv 10/13/92]
'
' QuickBASIC source code for reading from or writing to a ROWS "standard"
' 1024 byte record data file. The file can be either a DOS file or a
' file on an EXABYTE streaming tape.
'
' The "standard" format starts with a header that describes the subsequent
' DATA bytes in the record. The header has pairs of integers (2 bytes each),
' the first designating a data type, the second designating the number of
' data bytes used by that type. The sum of the number of bytes in the header
' plus the bytes in all the data types must equal 1024.
'
' The Aptek QuickBasic library qbx.lib is needed for proper compilation
' of this program.
'
OPTION BASE 1

TYPE FileStats
    FileName AS STRING * 40          ' The filename with directory and path cannot be longer than 40 characters.
    FileNo AS INTEGER
    Record AS LONG
    TotalRecords AS LONG
    ByteLoc AS LONG
    TStart AS STRING * 8
    TEnd AS STRING * 8
END TYPE

TYPE Dta
    Header AS STRING * 50            ' The present header is fixed at 50 characters.
    Radar AS STRING * 700
    AccurateTime AS STRING * 11      ' It is probably not a good idea to use a data structure for the decoded
    TickTime AS STRING * 11          ' variables. The problem with this approach is that it lacks flexibility if
    TrueTime AS STRING * 8           ' more data needs to be added to a ROWS record.
    TriggerDelay AS SINGLE
    Pitch AS SINGLE
    Roll AS SINGLE
    ShaftAngle AS SINGLE
    CorrShaftAngle AS SINGLE
    Latitude AS SINGLE
    Longitude AS SINGLE
    InsData AS STRING * 80           ' The INS data word is limited to 80 characters.
END TYPE

DECLARE SUB AsciiPitchRoll ()        ' Time, pitch, roll to separate ASCII file.
DECLARE SUB BegEndByte (A%, B%)      '
DECLARE SUB Blank ()                 '
DECLARE SUB CheckExbStatus (A%)      '
DECLARE SUB ColorLevels ()           ' Define colors for waterfall display.
DECLARE SUB CopyDosToDos ()          ' This feature hasn't been implemented
DECLARE SUB CopyExbToDos ()          '
DECLARE SUB DataTypes ()             ' Display the types of data.
DECLARE SUB DecodeHeader (H$)        ' This subroutine hasn't been completed. It's presently unused.
DECLARE SUB DecodeRecord ()          ' Decodes the 1024 byte data record.
DECLARE SUB DfnKy (A$)               ' Define the function keys.
DECLARE SUB DosFile ()               ' DOS will be the source device.
```

```
DECLARE SUB DosPreProcessing ()                          ' Display the DOS preprocessing menu.
DECLARE SUB DriveReady ()                                ' Test if EXABYTE drive is ready.
DECLARE SUB EndOfData ()                                 ' Advance the EXABYTE tape to the end of data
DECLARE SUB EnterFileName (A$, B$)                       ' Keyboard entry of DOS filename.
DECLARE SUB EnterRecord (A&)                             '
DECLARE SUB ExbFile ()                                   ' An EXABYTE tape has the source file.
DECLARE SUB ExbPreProcessing ()                          '
DECLARE SUB ExbReset (A%)                                ' Reset the EXABYTE tape deck.
DECLARE SUB ExbStatus (A$, B%)                           ' Sense code definitions for EXABYTE.
DECLARE SUB ExbTapeLog ()                                ' Produce complete EXABYTE tape log.
DECLARE SUB FileN ()                                     ' Advance the EXABYTE tape to file N.
DECLARE SUB FileOpen (A$)                                ' Check if DOS file is already open.
DECLARE SUB FirstMenu ()                                 ' Main Menu.
DECLARE SUB FkSet (A$())                                 ' Set the function key names.
DECLARE SUB GPSDisplay (A$)                              ' Displays GPS data on waveform display.
DECLARE SUB Help2 ()                                     ' Help menu for PlotRadReturn
DECLARE SUB IDLE ()                                      ' Pause program until operator keystroke.
DECLARE SUB IFGains (F%, TTo&, GT&())                    '
DECLARE SUB INFOFILE (MN$, MD$, MT&, GT&(), IFN$, GFN$, ITL!)
DECLARE SUB Keystroke (A%)                               ' Trap keyboard keystrokes.
DECLARE SUB LastRecord ()                                ' Set file to the final record.
DECLARE SUB Levels (A%)                                  ' Selects the waterfall display colors.
DECLARE SUB MERGE ()                                     ' Merges raw ROWS data on one EXB tape with INS, GPS, and
                                                         ' IF Gain data onto a second EXB tape.
DECLARE SUB MergeGPS (M$, S&, R1&, R2&, G&, GF%)
DECLARE SUB MergeINS (MR$, MH$, ST&, RT2#, InsT#, InsF%, InsTL$)
DECLARE SUB Message (A%, B$)                             ' Display color message.
DECLARE SUB MINMAX (A&(), N, x&, Xi, Y&, Yi)             ' Find the maximum and minimum values of a set of numbers.
DECLARE SUB NoFile ()                                    ' No file open warning.
DECLARE SUB OpenExistingFile ()                          ' Open a DOS ROWS file.
DECLARE SUB OpenNewFile (A$)                             ' Open a new DOS file for results output.
DECLARE SUB ParameterDisplay ()                          ' Display for PlotRadReturn
DECLARE SUB PlotRadReturn ()                             ' Plot radar return waveform.
DECLARE SUB PresentFile ()                               ' Display open file stats.
DECLARE SUB ProcessMenu ()                               ' Display the processing menu.
DECLARE SUB ReadRecord ()                                ' Read a new data record.
DECLARE SUB RecordN ()                                   ' Set file to record N.
DECLARE SUB ResetDosStats ()                             ' Initialize DOS variables.
DECLARE SUB ResetDta ()                                  ' Initialize decoded data variables.
DECLARE SUB ResetExbStats ()                             ' Initialize Exb variables.
DECLARE SUB ResetActiveStats ()                          ' Initialize operating variables.
DECLARE SUB Rewind ()                                    ' Rewind EXABYTE tape deck.
DECLARE SUB STARTINGTIME (S&, M&, I&, G&)                ' Used by MERGE to time synchronize GPS, INS, and ROWS files.
DECLARE SUB StepBackwards (A%, B%, C%)                   ' Step backward through DOS file.
DECLARE SUB StepForwards (A%, B%, C%)                    ' Step forward through DOS or EXB file.
DECLARE SUB Update ()                                    ' Update the screen display
DECLARE SUB ValidROWSFile (A$, B%)                       ' Check is file is valid ROWS file.
DECLARE SUB Version ()                                   ' Display the PREROWS version and date.
DECLARE SUB Waterfall ()                                 ' Color coded waterfall display of pulses.
DECLARE SUB WaveFormPlot (A%, B%)                        ' Plot radar waveform.
DECLARE SUB WriteRecord (F$, F%)                         ' Write a ROWS record to a DOS or EXB file.

DECLARE FUNCTION AccurateTime$ (A$)                      ' Decode the external box time into hh:mm:ss.
DECLARE FUNCTION ADTriggerDelay& (A$)                    ' Decode the A/D converter trigger time delay
DECLARE FUNCTION ByteLoc& ()                             ' Calculates the byte pointer in the active file.
DECLARE FUNCTION ClockTime$ ()                           ' Selects AccurateTime or PcTime for display.
DECLARE FUNCTION ColorSelection% ()                      ' Color planes for waterfall display.
DECLARE FUNCTION FKeyOnly$ ()                            ' Select a function key from the menu.
DECLARE FUNCTION GPSLatLong$ (LL$)                       ' Finds latitude and longitude from a GPS data file.
DECLARE FUNCTION IntegerTIME& (T$)                       ' Converts hours, minutes, and seconds to decimal seconds.
DECLARE FUNCTION KeyCode% (A$)                           ' Return keyboard scancode.
DECLARE FUNCTION NameDATE$ (N$, D$)                      ' Insert the mission name and date into a merged ROWS file.
```

```
DECLARE FUNCTION NewHEADER$ ()                  ' Constructs a new ROWS header.
DECLARE FUNCTION PcClockTime$ (A$)              ' Decode the pc real-time clock.
DECLARE FUNCTION PcTime$ (A&)                   ' Decode the pc tick time into hh:mm:ss.ss
DECLARE FUNCTION PitchRoll! (A$)                ' Convert aircraft pitch and roll to decimal degrees.
DECLARE FUNCTION TimeGPS& (D$)                  ' Find time from a GPS file.
DECLARE FUNCTION RotationAngle! (A$)            ' Decode the shaft angle of rotation.
DECLARE FUNCTION ShaftAngle$ ()                 ' Corrects the raw shaft angle.

DIM SHARED ColorLevel(0 TO 255)  AS INTEGER
DIM SHARED FileSource AS STRING * 3             ' DOS or EXABYTE?
DIM SHARED Header(2, 20) AS INTEGER             ' Col 1, type; col 2, length
DIM SHARED IERR AS INTEGER                      ' Contains the error code returned from the most recent EXABYTE call.
DIM SHARED Level(0 TO 14) AS INTEGER
DIM SHARED NewRecord  AS STRING * 1024          ' This is THE ROWS record
DIM SHARED RowOffSet AS INTEGER, ColOffSet AS INTEGER
DIM SHARED SenseStatus(0 TO 25) AS INTEGER      ' Contains the status of the EXABYTE after the most recent operation.

DIM SHARED Active AS FileStats                  ' Three variable have the data structure FileStats. Active contains the
DIM SHARED Dos AS FileStats                     ' information from the currently active file, variables Dos and Exb contain
DIM SHARED Exb AS FileStats                     ' the information from whichever (or both) file(s) are open.

DIM SHARED Dta AS Dta                           ' Define the Dta variable to be the structure Dta.
DIM SHARED NoData AS STRING * 2
DIM SHARED NoDataLong AS STRING * 4

CONST TicksPerSec = 18.20648#                   ' Computer clock ticks per second.

CONST BlankLine$ = "                                                                "
NoData$ = MKI$(-9999)                           ' -9999 indicates no data in 2 byte data string.
NoDataLong$ = MKL$(-999999999)                  ' -999999999 indicates no data in a four byte data string.

CONST HeaderLength% = 50                        ' Length of record header.
CONST ReadDriveId% = 5                          ' SCSII ID for drive being read from.
CONST WriteDriveId% = 0                         ' SCSII ID for drive being written to.

SCREEN 0
CLS

ResetActiveStats
ResetDosStats
ResetExbStats
ResetDta

FileSource$ = "  "

DO

   FirstMenu

LOOP

END



BadFileName:

   Message 17, "ILLEGAL FILENAME or "
   Message 18, "File doesn't exist."
   RESUME NEXT
```

```
'                    **** BEGIN SUBROUTINES ****

'   Decodes the time delay between the radar mainbang and the triggering
'   of the A/D converter. The result is given in meters below the aircraft.
'
'   The delay is affected in hardware by counting down in a 16 bit
'   register from FFFF hex. Including the bias mentioned below, the
'   conversion of the delay to range can be done by representing the
'   integer as a 4 byte string and then converting it to a signed
'   long integer as shown below.
'
' CALLed by:        DecodeRecord
'
FUNCTION ADTriggerDelay& (Bytes$)

    Bias% = 74!                                    ' System induced bias to time delay (up through 7/25/91)

    IF Active.Record = 1 AND FileSource$ = "DOS" THEN

        ADTriggerDelay& = 0

    ELSE

        Count& = CVL(Bytes$ + CHR$(0) + CHR$(0))
        ADTriggerDelay& = 50 * .15 * (65535 - Count& - Bias%)

    END IF

END FUNCTION
```

```
'     Convert time, and aircraft pitch, and roll in a ROWS EXABYTE file to ASCII values. Time is formatted as
' hh:mm:ss and pitch and roll are formatted as decimal degrees. Since the ROWS records data 100 times per second,
' the pitch and roll are averaged over NAve% = 100 records. The output is to a DOS file at one record per second.

' CALLed by ExbPreProcessing
'
SUB AscIIPitchRoll

    DIM StartRecord AS LONG,  StopRecord AS LONG,   Skip AS LONG

    NAve% = 100                                          ' Number of records to average.
    Active = Exb

    PresentFile
    Message 17, "Enter the name of the output file"
    OpenNewFile FileName$

    IF LEN(FileName$) = 0 THEN
        EXIT SUB
    ELSE
        OPEN FileName$ FOR OUTPUT AS #1
    END IF

    LOCATE 18, 1: PRINT BlankLine$
    Dos.FileName = FileName$
    Active.FileName = Exb.FileName
    RecordN
    Update
    StartRecord& = Active.Record - 1

    DO
        LOCATE 18, 1: PRINT BlankLine$
        StopRecord = 0
        Message 17, "Enter the last record number to copy"
        EnterRecord StopRecord
        IF StopRecord < StartRecord& THEN
            Message 15, "ILLEGAL ENTRY"
            SLEEP 1
            LOCATE 15, 1: PRINT BlankLine$
        END IF
    LOOP WHILE StopRecord < StartRecord&

    Format$ = "######  ####.####  ####.####"
    PitchSum& = 0
    RollSum& = 0
    AngleConvert! = 360 / (1! * NAve% * (2 ^ 14))
    k% = NAve% - 1

    DO WHILE Active.Record <= StopRecord

        FOR i = Active.Record TO Active.Record + k%
            CALLS XRCHR(NewRecord, 1, IERR%)
            Active.Record = Active.Record + 1
            RollSum& = RollSum& + CVL(MID$(NewRecord, 762, 2) + CHR$(0) + CHR$(0))
            PitchSum& = PitchSum& + CVL(MID$(NewRecord, 764, 2) + CHR$(0) + CHR$(0))
        NEXT

        Hrs$ = HEX$(CVI(MID$(NewRecord, 757, 1) + CHR$(0)))
        Min$ = HEX$(CVI(MID$(NewRecord, 758, 1) + CHR$(0)))
        Sec$ = HEX$(CVI(MID$(NewRecord, 759, 1) + CHR$(0)))
        Seconds& = VAL(Hrs$) * 3600 + VAL(Min$) * 60 + VAL(Sec$)
```

```
        PitchAve! = 120! - AngleConvert! * PitchSum&
        RollAve! = 120! - AngleConvert! * RollSum&
        PRINT #1, USING Format$; Seconds&; PitchAve!; RollAve!
        PitchSum& = 0
        RollSum& = 0

        IF (IERR% AND 128) OR (IERR% AND 64) OR (IERR% AND 8) THEN EXIT DO
        DecodeRecord
        Update

    LOOP

    Active.ByteLoc = ByteLoc
    DecodeRecord
    CLOSE #1
    Update

END SUB

'*****************************************************************************************************************
                                                                                                    SUB BegEndByte
'    Allows user input of the starting sample (byte) number of the radar waveform for use with waveform plotting.
' The waveform has a total of 700 points on a single returned waveform.
'
' CALLed by:        PlotRadReturn;        Waterfall
'
SUB BegEndByte (WfStart%, WfStop%)

    CLS
    LOCATE 13, 16
    PRINT "Enter the beginning and ending byte numbers (separated by a"
    LOCATE 14, 16
    PRINT "comma) that you want to plot from each radar waveform."
    LOCATE 15, 30
    INPUT ; WfStart%, WfStop%
    CLS

END SUB

'*****************************************************************************************************************
                                                                                                        SUB Bold
' CALLed by:        DosFile

                        ' This routine probably doesn't need to be used, or , if it is, it should
                        ' be used more consistently.
SUB Blank

    Message 17, "This function key does nothing."

END SUB
```

```
                                                                                    FUNCTION ByteLoc&
'    Calculates the byte location of the readhead (EXABYTE) or file
'  pointer (DOS file) in an open file.
'
' CALLed by:      ASCIIPitchRoll;    CopyExbToDos;     DosPreProcessing;
'                 ExbPreProcessing;  ExbTapeLog;       OpenExistingFile;
'                 StepBackwards;     Waterfall
'
FUNCTION ByteLoc&

    ByteLoc = (Active.Record - 1) * 1024 + 1

END FUNCTION

'*****************************************************************************************************************
                                                                                    SUB CheckExbStatus
' CALLed by:      PlotRadReturn
'
SUB CheckExbStatus (SS%)

    SELECT CASE SS%

        CASE 128

            VIEW PRINT
            LOCATE 25, 28
            PRINT "END OF FILE #"; Active.FileNo
            IDLE
            EXIT SUB

        CASE 8

            VIEW PRINT
            LOCATE 25, 28
            PRINT "End of data. No EOF on last file."

        CASE 2

            CLS
            PRINT "EXABYTE NOT READY"
            IDLE
            EXIT SUB

        CASE 0

        CASE ELSE

            PRINT "SenseStatus(2) = "; SS%
            IDLE

    END SELECT

END SUB
```

```
'    Returns the time of the present file record. This function is used
' throughout the program so that any change in byte locations for the
' system time will only require an addition here and the addition of an
' appropriate subroutine that does the actual time determination.
'
' CALLed by:      CopyExbToDos;     ExbPreProcessing;   ExbTapeLog;
'                 FileN;            GPSDisplay;         MERGE;
'                 OpenExistingFile; ParameterDisplay;   PcClockTime;
'                 PresentFile;      RecordN;            Update;
'                 Waterfall
'
FUNCTION ClockTime$

'    ClockTime$ = Dta.AccurateTime
'                 "Accurate" Time was recorded pre-August 1991
'
'    ClockTime$ = Dta.TickTime
'                 TickTime is taken from the pc internal tick counter
'                 that counts the number of ticks since the previous
'                 midnight. A standard pc has 18.20648 ticks per
'                 second.
'
   ClockTime$ = Dta.TrueTime
'                 TrueTime is taken from the internal pc real-time
'                 clock.
'
END FUNCTION
```

```
' Define the color levels assigned to the 8 bit integer data values for waterfall display color coding.
'
' CALLed by:        Waterfall
'
SUB ColorLevels

    Levels ColorSelection%                          ' Select the color levels

    FOR i = 0 TO 255
       SELECT CASE i
          CASE 0 TO Level%(0)
             ColorLevel(i) = 0                      'BLACK

          CASE Level%(0) + 1 TO Level%(1)
             ColorLevel(i) = 1                      'BLUE

          CASE Level%(1) + 1 TO Level%(2)
             ColorLevel(i) = 2                      'GREEN

          CASE Level%(2) + 1 TO Level%(3)
             ColorLevel(i) = 3                      'CYAN

          CASE Level%(3) + 1 TO Level%(4)
             ColorLevel(i) = 4                      'RED

          CASE Level%(4) + 1 TO Level%(5)
             ColorLevel(i) = 5                      'MAGENTA

          CASE Level%(5) + 1 TO Level%(6)
             ColorLevel(i) = 6                      'BROWN

          CASE Level%(6) + 1 TO Level%(7)
             ColorLevel(i) = 7                      'WHITE

          CASE Level%(7) + 1 TO Level%(8)
             ColorLevel(i) = 8                      'GRAY

          CASE Level%(8) + 1 TO Level%(9)
             ColorLevel(i) = 9                      'LIGHT BLUE

          CASE Level%(9) + 1 TO Level%(10)
             ColorLevel(i) = 10                     'LIGHT GREEN

          CASE Level%(10) + 1 TO Level%(11)
             ColorLevel(i) = 11                     'LIGHT CYAN

          CASE Level%(11) + 1 TO Level%(12)
             ColorLevel(i) = 12                     'LIGHT RED

          CASE Level%(12) + 1 TO Level%(13)
             ColorLevel(i) = 13                     'LIGHT MAGENTA

          CASE Level%(13) + 1 TO Level%(14)
             ColorLevel(i) = 14                     'YELLOW

          CASE IS > Level%(14)
             ColorLevel(i) = 15                     'BRIGHT WHITE

       END SELECT
    NEXT
END SUB
```

```
' Allows user selection of which color level array to use for plotting a waterfall display.
'
' CALLed by:        ColorLevels
'
FUNCTION ColorSelection%

    MaxSelection! = 6!                          ' Defines the total number of color coded
                                                ' arrays that have been defined in the
                                                ' subroutine Levels.

    DO

        Message 17, "Which color selection do you want?"
        Message 18, "1 through" + STR$(MaxSelection!)
        k$ = INPUT$(1)
        C% = VAL(k$)

    LOOP WHILE C% > MaxSelection! AND C% < 1!

    ColorSelection% = VAL(k$)

END FUNCTION
```

```
'   Copy a sequence of records from an EXABYTE file to a DOS file.
'
' CALLed by:        ExbPreProcessing
'
'    [MODS.: 1/10/92; corrected read/write loop to copy all desired records, including the last one.
'          : 7/01/92; took out inline code to write records to the DOS file and put code in a subroutine.]
'
SUB CopyExbToDos

    DIM StartRecord AS LONG, StopRecord AS LONG, Skip AS LONG

    Active = Exb
    PresentFile
    Message 17, "Enter the name of the file you want to COPY to"
    OpenNewFile FileName$
    FileNo% = FREEFILE
    IF LEN(FileName$) = 0 THEN
        EXIT SUB
    ELSE
        OPEN FileName$ FOR BINARY AS #FileNo%
    END IF
    LOCATE 18, 1: PRINT BlankLine$
    Dos.FileName = FileName$
    Active.FileName = Exb.FileName
    RecordN

    DO
        LOCATE 18, 1: PRINT BlankLine$
        StopRecord = 0
        Message 17, "Enter the last record number to copy"
        EnterRecord StopRecord
        IF StopRecord < StartRecord& THEN
            Message 15, "ILLEGAL ENTRY"
            SLEEP 1
            LOCATE 15, 1: PRINT BlankLine$
        END IF
    LOOP WHILE StopRecord < StartRecord&

    DO WHILE Active.Record <= StopRecord + 1
        WriteRecord "DOS", FileNo%
        IF IERR% AND 128 THEN EXIT DO
        ReadRecord
        DecodeRecord
        Update
    LOOP

    Active.ByteLoc = ByteLoc
    Exb = Active
    Dos.TotalRecords = LOF(1) / 1024
    Dos.TEnd = ClockTime$
    SEEK #FileNo%, 1
    FileSource$ = "DOS"
    ReadRecord
    DecodeRecord
    Dos.Record = 2
    Dos.TStart = ClockTime$
    Dos.ByteLoc = (Dos.Record - 1) * 1024 + 1
    FileSource = "EXB"
    Active = Exb

END SUB
```

```
'   [NEEDS WORK]  At present this routine is not used.
'
' Define the type of data stored in each data type within a 1024 byte record.
'
' CALLed by:  [not yet used]
'
SUB DataTypes

    DIM DataType$(0 TO 99)

    DataType$(0) = "HEADER"
    DataType$(1) = "UNPROCESSED RADAR RETURN"
    DataType$(2) = "COMPUTER CLOCK 'TICKS' SINCE MIDNIGHT"
    DataType$(3) = "DIGITIZER TIME DELAY FROM RADAR FIRING"
    DataType$(4) = "EXTERNAL CLOCK TIME (high precision)"
    DataType$(5) = "AIRCRAFT PITCH ANGLE"
    DataType$(6) = "AIRCRAFT ROLL ANGLE"
    DataType$(7) = "ANTENNA SHAFT 'ANGLE'"
    DataType$(8) = "'CORRECTED' ANTENNA SHAFT 'ANGLE'"
    DataType$(9) = "OBSERVER LATITUDE"
    DataType$(10) = "OBSERVER LONGITUDE"
    DataType$(11) = "INS DATA"
    DataType$(12) = ""
    DataType$(13) = ""
    DataType$(14) = ""
    DataType$(15) = ""
    DataType$(16) = ""
FOR i = 1 TO 16
PRINT DataType$(i)
NEXT i
SLEEP 5
END SUB
```

```
' Decode the ROWS header record into a 2xn dimensional integer array, where n is the number of types designated
' in the header, column 1 designates the data type, and column 2 gives the length of the data type in bytes.
'
' CALLed by:     [not yet used]
'
SUB DecodeHeader (H$)

    FOR j = 0 TO 1
        FOR k = j + 2 ^ j TO LEN(H$) - 2 STEP 4
            Head1% = CVI(MID$(H$, k, 1) + CHR$(0))
            Head2% = CVI(CHR$(0) + MID$(H$, k + 1, 1))
            Header(j + 1, (k + 3 - 2 * j) / 4) = Head1% + Head2%
        NEXT k
    NEXT j

END SUB
```

SUB DecodeRecord

```
'    Decode ROWS 1024 byte data record into its various variables.  The data is contained in NewRecord$ which
' is obtained from the subroutine ReadRecord.
'
' CALLed by:      AscIIPitchRoll;    CopyExbToDos;    DosPreProcessing
'                 ExbPreProcessing;  ExbTapeLog;      FileN
'                 OpenExistingFile;  RecordN;         StepBackwards
'                 StepForwards;      Waterfall
'
SUB DecodeRecord

        ' Header                                        : bytes 1 to 50
    Dta.Header = MID$(NewRecord, 1, 50)

        ' Radar return                                  : bytes 51 to 750
    Dta.Radar = MID$(NewRecord, 51, 700)

        ' p.c. internal tick counter time          : bytes 751 to 754
    Dta.TickTime = PcTime$(CVL(MID$(NewRecord, 751, 4)))

        ' AD trigger                                   : bytes 755 to 756
    Dta.TriggerDelay = ADTriggerDelay&(MID$(NewRecord, 755, 2))

        ' p.c. realtime clock time                     : bytes 757 to 758
    Dta.TrueTime = PcClockTime$(MID$(NewRecord, 757, 3))

        ' External "black box" time used until August 1991
    Dta.AccurateTime = AccurateTime$(MID$(NewRecord, 757, 5))

        ' aircraft roll from INS                       : bytes 762 to 763
    Dta.Roll = PitchRoll!(MID$(NewRecord, 762, 2))

        ' aircraft pitch from INS                      : bytes 764 to 765
    Dta.Pitch = PitchRoll!(MID$(NewRecord, 764, 2))

        ' antenna pointing angle                       : bytes 767 to 768
    Dta.ShaftAngle = RotationAngle!(MID$(NewRecord, 767, 2))

END SUB
```

****

SUB DfnKy

```
'
' CALLed by:      FkSet
'
SUB DfnKy (Kt$)                          ' Assign labels to the Function keys.

    FOR i = 1 TO 10
        KEY i, MID$(Kt$, 7 * (i - 1) + 1, 7)
    NEXT i
    KEY ON

END SUB
```

```
'   Top menu for selecting preprocessing functions of a DOS file.
'
' CALLed by:        FirstMenu
'
SUB DosFile

    DIM KeyText$(10)

    FileSource$ = "DOS"
    Active = Dos

    DO

        KeyText$(1) = "MainMnu":        KeyText$(2) = "OpnFile"
        KeyText$(3) = "PreProc":        KeyText$(4) = "       "
        KeyText$(5) = "       ":        KeyText$(6) = "       "
        KeyText$(7) = "       ":        KeyText$(8) = "       "
        KeyText$(9) = "ClsFile":        KeyText$(10) = "Exit   "

        FkSet KeyText$()

        CLS

        PresentFile

        Message 17, "Select some sort of DOS activity"

        FkCase$ = FKeyOnly$

        SELECT CASE FkCase$

            CASE "MainMnu"  ' Return to the main menu

                Dos = Active
                EXIT SUB

            CASE "OpnFile"

                OpenExistingFile

            CASE "       "

                Blank

            CASE " "

                IF FREEFILE > 1 THEN
                    Blank
                ELSE
                    NoFile
                END IF

            CASE " "

                IF FREEFILE > 1 THEN
                    Blank
                ELSE
                    CopyDosToDos
                END IF

            CASE "PreProc"
```

```
            IF FREEFILE > 1 THEN
                DosPreProcessing
            ELSE
                NoFile
            END IF

        CASE "ClsFile"

            ResetDosStats

            Active = Dos

        CASE "Exit   "
            END

    END SELECT

    Dos = Active

LOOP

END SUB
```

```
'   Primary menu for selecting preprocessing operations to perform on a DOS file.
'
' CALLed by:        DosFile
'
SUB DosPreProcessing

    DIM KeyText$(10)

    Active = Dos

    DO

        KeyText$(1) = "DosMenu":        KeyText$(2) = "       "
        KeyText$(3) = "Rec #N ":        KeyText$(4) = "       "
        KeyText$(5) = "WtrFall":        KeyText$(6) = "DspData"
        KeyText$(7) = "       ":        KeyText$(8) = "       "
        KeyText$(9) = "       ":        KeyText$(10) = "Exit   "

        FkSet KeyText$()

        CLS

        PresentFile

        '     put some screen text here to explain some of the FKs

        Message 17, "Select some sort of ROWS preprocessing activity"

        FkCase$ = FKeyOnly$

        SELECT CASE FkCase$

            CASE "DosMenu"

            CASE "       "

            CASE "Rec #N "

                RecordN
                ReadRecord
                DecodeRecord
                Active.ByteLoc = ByteLoc

            CASE "WtrFall"

                Waterfall

            CASE "DspData"

                PlotRadReturn

            CASE "Exit   "
                END

        END SELECT

        Dos = Active

    LOOP UNTIL FkCase$ = "DosMenu"

END SUB
```

```
                                                                                    SUB DriveReady
'    Test if EXB-8200 is ready to accept a new command; wait until previous read or write operation is finished.
'
' CALLed by:      ExbReset;                ExSPACE;          MERGE
'                 WriteRecord

SUB DriveReady

   DO
      CALLS XREADY(IERR%)
   LOOP WHILE IERR% <> 0

END SUB

'*****************************************************************************************************************
                                                                                    SUB EnterFileName
' CALLed by:      OpenExistingFile

SUB EnterFileName (FileName$, ValidName$)

   FileName$ = ""
   ValidName$ = "Y"
   ON ERROR GOTO BadFileName
   LOCATE 18, 35
   INPUT ; FileName$
   FileName$ = UCASE$(FileName$)
   IF LTRIM$(FileName$) = "" THEN EXIT SUB
   IF ERR <> 0 THEN FileName$ = ""
   OPEN FileName$ FOR BINARY AS #5

   SELECT CASE FileSource$

      CASE "DOS"

         IF ERR <> 0 THEN
            ResetDosStats
            ValidName$ = "N"
         ELSE
            ValidROWSFile ValidName$, 5!
            IF ValidName$ = "N" THEN ResetDosStats
         END IF

      CASE "EXB"

         IF ERR <> 0 THEN
            ValidName$ = "N"
         ELSE
            ResetDosStats
         END IF

   END SELECT

   ON ERROR GOTO 0
   CLOSE #5

END SUB
```

SUB EnterRecord

```
' Keyboard entry of record number at which processing will begin.
'
' CALLed by:      AscIIPitchRoll;     CopyExbToDos;     RecordN

SUB EnterRecord (S&)

    DO

        NewRec& = 0
        LOCATE 18, 37
        INPUT ; NewRec&

    LOOP WHILE NewRec& <= 0

    S& = NewRec&

END SUB
```

****

SUB ExbFile

```
' CALLed by:      FirstMenu

SUB ExbFile                                    ' Operate on an EXABYTE tape file.

    DIM KeyText$(10)

    FileSource$ = "EXB"
    Active = Exb

    CALLS XSENSE(SenseStatus(0), IERR%)
    IF (SenseStatus(2) AND 64) = 0 THEN
        IF Active.TStart = "        " THEN
            ' This condition only occurs if PREROWS is started from
            ' the top and the tape in not at LBOT.
            ExbReset (ReadDriveId%)
            ResetExbStats
            Active = Exb
        END IF
    END IF

    DO

        KeyText$(1) = "MainMnu":        KeyText$(2) = "PreProc"
        KeyText$(3) = "      ":         KeyText$(4) = "      "
        KeyText$(5) = "LogExb ":        KeyText$(6) = "      "
        KeyText$(7) = "Merge  ":        KeyText$(8) = "      "
        KeyText$(9) = "Initize":        KeyText$(10) = "EXIT   "

        FkSet KeyText$()

        CALLS XSENSE(SenseStatus(0), IERR%)
        IF (SenseStatus(2) AND 6) THEN
            ' This condition only occurs if the tape deck door has
            ' been opened while this program is in or below this
            ' DO LOOP location.
            ResetExbStats
            Active = Exb
        END IF

        CLS
        PresentFile
```

```
        Message 17, "Select some sort of EXABYTE activity."
        FkCase$ = FKeyOnly$

        SELECT CASE FkCase$

            CASE "MainMnu"                          ' F1

                EXIT SUB                            ' Return to first menu

            CASE "PreProc"

                ExbPreProcessing

            CASE "      "

            CASE "LogExb "                          'Produce a log of contiguous sequences
                                                    'of records on an ExaByte tape.
                ExbTapeLog

            CASE "Merge  "

                MERGE
                CALLS SETTGT(ReadDriveId%, 0)
                EXIT SUB                            'Return to first menu

            CASE "Initize"

                ExbReset (ReadDriveId%)
                ResetExbStats
                Active = Exb

            CASE "EXIT   "

                END

        END SELECT

        Exb = Active

    LOOP

END SUB
```

```
' CALLed by:      ExbFile

SUB ExbPreProcessing

    DIM KeyText$(10)

    DO

        KeyText$(1) = "ExbMenu":        KeyText$(2) = "File N "
        KeyText$(3) = "Rec #N ":        KeyText$(4) = "CpyFile"
        KeyText$(5) = "WtrFall":        KeyText$(6) = "DspData"
        KeyText$(7) = "PtchRol":        KeyText$(8) = "       "
        KeyText$(9) = "       ":        KeyText$(10) = "Exit   "

        FkSet KeyText$()
        CLS
        Active = Exb
        PresentFile

        Message 17, "Select some sort of ROWS preprocessing activity"

        FkCase$ = FKeyOnly$

        SELECT CASE FkCase$

            CASE "ExbMenu"                  ' F1

                EXIT SUB

            CASE "File N "                              ' F2

                FileN
                Active.FileName = RTRIM$("Exabyte file # ") + STR$(Active.FileNo)
                ReadRecord
                DecodeRecord
                Active.ByteLoc = ByteLoc
                Active.TStart$ = ClockTime$

            CASE "Rec #N "                  ' F3

                RecordN

            CASE "CpyFile"                  ' F4

                CopyExbToDos

            CASE "WtrFall"                  ' F5

                Waterfall

            CASE "DspData"                  ' F6

                PlotRadReturn

            CASE "PtchRol"                  ' F7

                AsciIPitchRoll

            CASE "       "
```

```
        CASE "Exit  "                          ' F10

            END

    END SELECT

    Exb = Active
    Update

  LOOP

END SUB
```

****


```
' CALLed by:      ExbFile; MERGE

SUB ExbReset (Id%)                    ' Reset the EXABYTE tape deck.

  CONST RecSize% = 1024
  CLS
  Message 17, "RESETTING DRIVE"
  CALLS SETTGT(Id%, 0)
  CALLS XSMODE(0, RecSize%, 0, 0, 0, 0, 0, 5, 5, 0, IERR%)
  CALLS XRESET(IERR)
  DriveReady

END SUB
```

```
'   SenseStatus(2) error code definitions for EXABYTE drive.

' CALLed by:        PlotRadReturn
'
SUB ExbStatus (Sense$, SenseCode%)

    DIM Sense2(0 TO 15) AS STRING
    DIM Sense3 AS LONG
    DIM Sense19(0 TO 7) AS STRING
    DIM Sense20(0 TO 7) AS STRING
    DIM Sense21(0 TO 3) AS STRING

    Sense2$(0) = "No sense data":          Sense2$(1) = "Not used"
    Sense2$(2) = "Not ready":              Sense2$(3) = "Medium error"
    Sense2$(4) = "Hardware error":         Sense2$(5) = "Illegal request"
    Sense2$(6) = "Unit attention":         Sense2$(7) = "Data protect"
    Sense2$(8) = "Check for blank tape":   Sense2$(9) = "       "
    Sense2$(10) = "Aborted":               Sense2$(11) = "Aborted command"
    Sense2$(12) = "Not used":              Sense2$(13) = "Volume overflow"
    Sense2$(14) = "Not used":              Sense2$(15) = "Not used"

    Sense19$(0) = "Beginning of tape":     Sense19$(1) = "Tape not present"
    Sense19$(2) = "Tape motion error":     Sense19$(3) = "Error counter overflow"
    Sense19$(4) = "Media error":           Sense19$(5) = "ExaByte buffer error"
    Sense19$(6) = "SCSI bus parity error"
    Sense19$(7) = "Reset since last status"

    Sense20$(0) = "Formatter error":       Sense20$(1) = "Servo system error"
    Sense20$(2) = "Media error":           Sense20$(3) = "Under run error"
    Sense20$(4) = "Error during filemark write"
    Sense20$(5) = "Tape is write protected"
    Sense20$(6) = "Tape mark error detected"
    Sense20$(7) = "Transfer abort error"

    Sense21$(0) = "Write splice error, overshoot"
    Sense21$(1) = "Write splice error, no gap track"
    Sense21$(2) = "Physical end of tape"

    CALLS XSENSE(SenseStatus(0), IERR%)

    IF (SenseStatus(0) AND 128) THEN

        IF (SenseStatus(2) AND 1) THEN


        ELSEIF (SenseStatus(2) AND 2) THEN


        ELSEIF (SenseStatus(2) AND 4) THEN


        ELSEIF (SenseStatus(2) AND 8) THEN


        ELSEIF (SenseStatus(2) AND 16) THEN
```

```
ELSEIF (SenseStatus(2) AND 32) THEN

ELSEIF (SenseStatus(2) AND 64) THEN

ELSEIF (SenseStatus(2) AND 128) THEN

END IF
IF SenseStatus(19) AND NOT 0 THEN
    IF (SenseStatus(19) AND 1) THEN

    ELSEIF (SenseStatus(19) AND 2) THEN

    ELSEIF (SenseStatus(19) AND 4) THEN

    ELSEIF (SenseStatus(19) AND 8) THEN

    ELSEIF (SenseStatus(19) AND 16) THEN

    ELSEIF (SenseStatus(19) AND 32) THEN

    ELSEIF (SenseStatus(19) AND 64) THEN

    ELSEIF (SenseStatus(19) AND 128) THEN

    END IF
END IF
IF (SenseStatus(20) AND NOT 0) THEN
    IF (SenseStatus(20) AND 1) THEN

    ELSEIF (SenseStatus(20) AND 2) THEN
```

```
            ELSEIF (SenseStatus(20) AND 4) THEN

            ELSEIF (SenseStatus(20) AND 8) THEN

            ELSEIF (SenseStatus(20) AND 16) THEN

            ELSEIF (SenseStatus(20) AND 32) THEN

            ELSEIF (SenseStatus(20) AND 64) THEN

            ELSEIF (SenseStatus(20) AND 128) THEN

            END IF
        END IF
        IF (SenseStatus(21) AND NOT 0) THEN
            IF (SenseStatus(21) AND 1) THEN

            ELSEIF (SenseStatus(21) AND 2) THEN

            ELSEIF (SenseStatus(21) AND 4) THEN

            END IF
        END IF
    ELSE
        Message 17, "ExaByte sense data invalid for last Call to ExaByte"
    END IF
END SUB
```

```
'   Produce a log of an EXABYTE tape. The log will indicate all time gaps within each tape file, the total number
'   of records in each file, and the starting and stopping time of each file.

' CALLed by:        ExbFile
'
SUB ExbTapeLog

    DIM NewName AS STRING * 1
    DIM BeginRecordTime AS STRING * 11, EndingRecordTime AS STRING * 11
    DIM BeginRecordNumber AS LONG
    DIM TimeTicks(0 TO 1)  AS LONG

    CONST RecordIncrement% = 100
    CONST RecordsPerSec = 100
    CONST RecsPerTick% = RecordsPerSec / TicksPerSec
    CONST TestTime = (1! * RecordIncrement% / RecsPerTick%) + RecsPerTick%

    NewName$ = "N"

    DO
        CLOSE #3
        Message 17, "Enter the NAME of the DOS file for the log output."
        LOCATE 18, 35
        INPUT ; FileLog$
        IF FileLog$ = "" THEN EXIT SUB
        OPEN FileLog$ FOR APPEND AS #3

        IF LOF(3) > 0 THEN

            Message 17, "WARNING, A FILE OF THE SAME NAME ALREADY EXISTS."
            Message 18, "Do you want to use a different filename? (Y/N)"
            LOCATE 19, 35
            NewName$ = UCASE$(INPUT$(1))

        END IF

    LOOP WHILE NewName$ = "Y"

    Rewind
    CLS

    Ps1$ = "\       \   "                                   'Print using format string for disk output.
    Ps2$ = "###### "                                        'Print using format string for screen output.

    Header1$ = "Start Time     Stop Time     Start    Stop   Number"
    Header2$ = "  pc tick       pc tick      Rec #   Rec #    Recs"

    DO                                                      ' Loop on EOF until EOD

        NonRecords% = 0
        Active.Record = 1
        Active.ByteLoc = 1
        BeginRecordNumber& = 1
        ReadRecord
        DecodeRecord
        BeginRecordTime = ClockTime$
        Active.TStart = BeginRecordTime

        PRINT #3,
        PRINT #3, SPACE$(20) + "FILE NUMBER: "; Active.FileNo: PRINT
        PRINT SPACE$(30) + "FILE NUMBER: "; Active.FileNo: PRINT
```

```
PRINT #3, Header1$: PRINT #3, Header2$
PRINT SPACE$(10) + Header1$: PRINT SPACE$(10) + Header2$
SaveLine = CSRLIN
SaveCol = POS(0)

DO                                          ' Loop on file until EOF

   RecordsMoved% = RecordIncrement% - 1
   DO                                       ' Check for time gap within record increment

      TimeTicks&(0) = CVL(MID$(NewRecord, 751, 4))

      CALLS XSPACE(RecordsMoved%, IERR%)
      CALLS XSENSE(SenseStatus(0), IERR%)
      Active.Record = Active.Record + RecordsMoved%

      AtEOF% = SenseStatus(2) AND 128
      AtEOM% = SenseStatus(2) AND 64
      AtEOD% = SenseStatus(2) AND 8

      IF AtEOF% OR AtEOM% OR AtEOD% THEN ' EOF test.
         Active.Record = Active.Record - SenseStatus(6)
         EXIT DO
      END IF

      ReadRecord
      DecodeRecord

      COLOR 7, 0: LOCATE 24, 15
      PRINT "PRESENT RECORD #: ";
      COLOR 4, 7: PRINT Active.Record - 1;
      COLOR 7, 0: PRINT "  TIME: ";
      COLOR 4, 7: PRINT " " + ClockTime$;
      COLOR 7, 0

      TimeTicks&(1) = CVL(MID$(NewRecord, 751, 4))
      DeltaT& = TimeTicks&(1) - TimeTicks&(0)

   LOOP WHILE (DeltaT& < TestTime)

   EndingRecordTime = PcTime$(TimeTicks&(0))
   ContinuousRecords = Active.Record - BeginRecordNumber& - RecordsMoved%

   PRINT #3, USING Ps1$; BeginRecordTime$; EndingRecordTime$;
   PRINT #3, USING "######  "; BeginRecordNumber&; Active.Record - RecordsMoved% - 1; ContinuousRecords

   LOCATE SaveLine, SaveCol
   PRINT SPACE$(10);
   PRINT USING Ps1$; BeginRecordTime$; EndingRecordTime$;
   PRINT USING Ps2$; BeginRecordNumber&; Active.Record - RecordsMoved% - 1; ContinuousRecords

   BeginRecordTime$ = PcTime$(TimeTicks&(1))
   BeginRecordNumber& = Active.Record - 1

   IF (AtEOF% + AtEOM% + AtEOD%) > 0 THEN EXIT DO

LOOP

Active.TEnd = PcTime$(TimeTicks&(1))            'ClockTime$

PRINT #3, SPACE$(10); "End of file."
```

```
        PRINT #3, "Start Time: "; Active.TStart; "  Stop time: "; Active.TEnd
        PRINT #3, "Total Records:"; Active.Record

        PRINT SPACE$(10); "End of file."
        PRINT "Start Time: "; Active.TStart; "  Stop time: "; Active.TEnd
        PRINT "Total Records:"; Active.Record

        CALLS XSPACE(1, IERR%)
        Active.FileNo = Active.FileNo + 1

    LOOP WHILE (IERR% AND 8!) = 0

    PRINT #3, "END OF DATA."

    CLOSE #3

    Active.FileNo = Active.FileNo - 1
    Active.FileName = RTRIM$("Exabyte file # ") + STR$(Active.FileNo)
    Active.ByteLoc = ByteLoc
    Active.TEnd = ClockTime$
    Active.TotalRecords = Active.Record

    PRINT : PRINT "PRESS ANY KEY TO CONTINUE": Wait$ = INPUT$(1)

END SUB
```

```
' CALLed by:      ExbPreProcessing

SUB FileN

    DO
        CLS
        PresentFile
        Message 17, "Enter the number of the file you want to process."
        LOCATE 18, 35
        INPUT ; NewFileNo%
        IF NewFileNo% = 0 THEN EXIT SUB
    LOOP WHILE NewFileNo% < 0

    SkipFile% = NewFileNo% - Active.FileNo

    SELECT CASE NewFileNo%
        CASE IS = Active.FileNo

        CASE IS > Active.FileNo

            DO WHILE SkipFile% > 0

                ReadRecord                              ' Create sense data to test for EOD

                IF IERR% AND 8 THEN                     ' If EOD.
                    Text$ = "EOD reached; tape set to beginning of last file."
                    Message 15, Text$
                    CALLS XSKIPF(-2, IERR%)
                    CALLS XSKIPF(1, IERR%)
                    EXIT DO
                ELSEIF (IERR% AND 128) <> 0 THEN
                    CALLS XSKIPF(-1, IERR%)
                ELSE
                    IF Active.FileNo = 0 THEN SkipFile% = NewFileNo% - 1
                    CALLS XSKIPF(1, IERR%)
                END IF

                SkipFile% = SkipFile% - 1!

            LOOP

        CASE IS < Active.FileNo

            IF NewFileNo% = 1 THEN
                CALLS XREWND(IERR%)
            ELSE
                CALLS XSKIPF(SkipFile% - 1, IERR%)
                CALLS XSKIPF(1, IERR%)
            END IF

            SkipFile% = 0

    END SELECT

    ResetExbStats
    Active = Exb
    DecodeRecord
    Active.TStart = ClockTime$
    Active.FileNo = NewFileNo% - SkipFile%

END SUB
```

```
' CALLed by:        OpenExistingFile

SUB FileOpen (Close1$)

    Close1$ = "Y"

    IF FREEFILE <> 1 THEN

        CLS
        Message 17, RTRIM$(Active.FileName) + " is presently open"
        Message 19, "Do you want to open a different file (Y/N)?"

        DO
            Close1$ = UCASE$(INPUT$(1))
        LOOP WHILE Close1$ <> "Y" AND Close1$ <> "N"

        IF Close1$ = "Y" THEN
            ResetDosStats
            Active = Dos
        END IF

    END IF

END SUB
```

```
' CALLed by:        MAIN

SUB FirstMenu                                   ' **** Initial program menu.

   DIM KeyText$(10)

   DO

      KeyText$(1) = "      ":                   KeyText$(2) = "DOSFILE"
      KeyText$(3) = "EXBFILE":                  KeyText$(4) = "      "
      KeyText$(5) = "      ":                   KeyText$(6) = "      "
      KeyText$(7) = "      ":                   KeyText$(8) = "      "
      KeyText$(9) = "VERSION":                  KeyText$(10) = "EXIT  "

      FkSet KeyText$()
      FkCase$ = ""
      FileSource$ = "  "
      ResetActiveStats
      ResetDta

      CLS
      PresentFile

      LOCATE 14, 18
      PRINT "This program is used to perform preprocessing and"
      LOCATE 15, 18
      PRINT "low level processing on ROWS data files."
      LOCATE 20, 20
      PRINT "PRESS THE FUNCTION KEY INDICATING THE SOURCE"
      LOCATE 21, 20
      PRINT "OF THE ROWS DATA FILE YOU WANT TO PROCESS."
      FkCase$ = FKeyOnly$

      SELECT CASE FkCase$

         CASE "DOSFILE"                 ' F2

            DosFile

         CASE "EXBFILE"                 ' F3

            ExbFile

         CASE "VERSION"                 ' F9

            Version

         CASE "EXIT  "                  ' F10

            END

      END SELECT

   LOOP

END SUB
```

FUNCTION FKeyOnly$

```
' CALLed by:      DosFile;          DosPreProcessing;  ExbFile
'                 ExbPreProcessing; FirstMenu

FUNCTION FKeyOnly$

   DO

      k$ = ""
      Char$ = ""

      DO
         Char$ = INKEY$
      LOOP UNTIL Char$ <> ""

      DO UNTIL LEN(Char$) = 0
         k$ = k$ + Char$
         Char$ = INKEY$
      LOOP

      k$ = k$ + Char$

      IF LEN(k$) <> 7 THEN BEEP

   LOOP UNTIL LEN(k$) = 7

   FKeyOnly$ = k$

END FUNCTION
```

****

SUB FkSet

```
' CALLed by:      DosFile;          DosPreProcessing;  ExbFile;
                  ExbPreProcessing; FirstMenu

SUB FkSet (Keys$())

      Kt$ = ""

      FOR i = 1 TO 10
         Kt$ = Kt$ + Keys$(i)
      NEXT

      DfnKy Kt$

END SUB
```

```
' CALLed by:        ParameterDisplay

SUB GPSDisplay (LatLong$)

    RowOffSet% = 18
    ColOffSet% = 45

    COLOR 3
    LOCATE RowOffSet%, ColOffSet% - 1
    PRINT CHR$(201); STRING$(25, CHR$(205)); CHR$(187)
    FOR i = 0 TO 4
        LOCATE RowOffSet% + i + 1, ColOffSet% - 1: PRINT CHR$(186)
        LOCATE RowOffSet% + i + 1, ColOffSet% + 25: PRINT CHR$(186)
    NEXT
    LOCATE RowOffSet% + 6, ColOffSet% - 1
    PRINT CHR$(200); STRING$(25, CHR$(205)); CHR$(188)
    COLOR 7: LOCATE RowOffSet% + 1, ColOffSet% + 4
    PRINT "GPS location at";
    LOCATE RowOffSet% + 2, ColOffSet% + 8
    PRINT ClockTime$
    COLOR 4: LOCATE RowOffSet% + 1, ColOffSet% + 7

    LOCATE RowOffSet + 4, ColOffSet% + 3
    COLOR 10
    PRINT "LATITUDE  : ";
    LOCATE RowOffSet + 5, ColOffSet% + 3
    PRINT "LONGITUDE : ";

    PresentLat# = CVL(MID$(LatLong$, 1, 4)) / 10000
    PresentLong# = CVL(MID$(LatLong$, 5, 4)) / 10000

    IF PresentLong# <> -99999.9999# THEN
        LOCATE RowOffSet + 5, ColOffSet% + 15
        PRINT USING "+##.####"; PresentLong#
    END IF
    IF PresentLat# <> -99999.9999# THEN
        LOCATE RowOffSet + 4, ColOffSet% + 15
        PRINT USING "+##.####"; PresentLat#
    END IF

    COLOR 15

END SUB
```

```
'    Strips latitude and longitude from GPS records, converts them to long integer string variables.

' CALLed by:        MergeGPS
'
FUNCTION GPSLatLong$ (LatLong$)

    Start% = 1
    LatPos% = 1
    DO UNTIL Start% = 3                                    ' Skip first two comma delimited fields.
        LatPos% = INSTR(LatPos%, LatLong$, ",") + 1
        Start% = Start% + 1
    LOOP

    Lat# = VAL(MID$(LatLong$, LatPos%, 2)) + (VAL(MID$(LatLong$, LatPos% + 2, 7))) / 60
    Lng# = -1 * (VAL(MID$(LatLong$, LatPos% + 12, 3)) + (VAL(MID$(LatLong$, LatPos% + 15, 7))) / 60!)
    Lat& = Lat# * 10 ^ 4
    Lng& = Lng# * 10 ^ 4
    GPSLatLong$ = MKL$(Lat&) + MKL$(Lng&)

END FUNCTION
```

```
                                                                                                    SUB Help2
' CALLed by:        PlotRadReturn
'
SUB Help2

    LOCATE 27, 1: COLOR 12
    PRINT "[ALT-F] Continuous forward.  [F] forward 1 record."; ""
    IF FileSource$ = "DOS" THEN
        LOCATE 28, 1
        PRINT "[ALT-B] Continuous backward.  [B] backward 1 record."
    END IF
    PRINT "ESC - exit."
    COLOR 15

END SUB

'*****************************************************************************************************************
                                                                                                    SUB IDLE
' CALLed by:        CheckExbStatus;     MERGE; Version

SUB IDLE
    Message 22, "SYSTEM SLEEPING."
    Message 23, "HIT ANY KEY TO CONTINUE."
    DO: k$ = INKEY$: LOOP WHILE k$ <> ""
    DO: k$ = INKEY$: LOOP WHILE k$ = ""
END SUB

'*****************************************************************************************************************
                                                                                                    SUB IFGains
'    Reads the IF gains and corresponding times of gain changes and converts them into long
'    integers in a two dimensional array.
'
' CALLED by:        INFOFILE
'
SUB IFGains (FileNo%, TTo&, GT&())

    DIM IfGain(100) AS STRING

    NumberGains% = 0
    DO                                                  ' Read gain table. 100 gain changes max.
        NumberGains% = NumberGains% + 1
        LINE INPUT #FileNo%, IfGain(NumberGains%)
        IF NumberGains% > 100 THEN
            CLS
            PRINT "Too many IF gain changes, input truncated."
            EXIT DO
        END IF
    LOOP UNTIL EOF(FileNo%)

                                                        ' Convert gain string to integer gain and time of gain change.
    FOR j = 1 TO NumberGains%
        Gain$ = LTRIM$(MID$(IfGain(j), 10))
        TimeGain$ = RTRIM$(LTRIM$(MID$(IfGain(j), 1, 9)))
        GT&(j, 1) = VAL(LEFT$(Gain$, INSTR(Gain$, " ") - 1))

                                                    ' Strip time from Gain$ and convert to integer seconds since midnight of airplane takeoff.
        GT&(j, 2) = IntegerTIME&(TimeGain$)
        IF GT&(j, 2) < TTo& THEN GT&(j, 2) = GT&(j, 2) + 86400
    NEXT

END SUB
```

```
' Read the information file from disk and reformat the data.

' CALLed by:        MERGE
'
SUB INFOFILE (MN$, MD$, MT&, GainTime&(), IFN$, GFN$, ITL!)

    DO
        ON ERROR GOTO BadFileName
        MergeInfoFile$ = ""
        Valid$ = "Y"

        LOCATE 18, 1: PRINT BlankLine$
        Message 15, "Enter the NAME of the MERGE information file"
        LOCATE 18, 35
        INPUT ; MergeInfoFile$
        MergeInfoFile$ = UCASE$(MergeInfoFile$)
        IF LTRIM$(MergeInfoFile$) = "" THEN
            MN$ = "": MD$ = "": MT& = 0: IFN$ = "": GFN$ = "": ITL! = 0
            EXIT SUB
        END IF
        FileNo% = 2 + FREEFILE
        OPEN MergeInfoFile$ FOR INPUT AS #FileNo%
        IF ERR <> 0 THEN
            MergeInfoFile$ = ""
            Valid$ = "N"
        END IF

    LOOP WHILE Valid$ = "N"

    ON ERROR GOTO 0

    LINE INPUT #FileNo%, MN$                    ' Mission name
    LINE INPUT #FileNo%, MD$                    ' Mission date
    LINE INPUT #FileNo%, MT$                    ' Mission takeoff time (UTC)
    LINE INPUT #FileNo%, IFN$                   ' INS data file name
    LINE INPUT #FileNo%, GFN$                   ' GPS data file name
    LINE INPUT #FileNo%, ITL$                   ' INS time lag wrt ROWS data

                                                ' Convert mission starting time from string to long integer.
    MT& = IntegerTIME&(RTRIM$(LTRIM$(MID$(MT$, 1, 10))))

                                                ' Read IF gain table; save in 2-D array with first column
                                                ' containing the gains and the second column the times.
    CALL IFGains(FileNo%, MT&, GainTime&())

    CLOSE #FileNo%

    IFN$ = MID$(IFN$, 1, INSTR(IFN$, " ") - 1)
    GFN$ = MID$(GFN$, 1, INSTR(GFN$, " ") - 1)
    ITL! = VAL(ITL$)

END SUB
```

```
                                                                                    FUNCTION IntegerTIME&
' CALLed by:        IFGains; INFOFILE;          MERGE

FUNCTION IntegerTIME& (T$)

    Hr& = VAL(LTRIM$(MID$(T$, 1, INSTR(T$, ":") - 1)))
    Min& = VAL(MID$(T$, INSTR(T$, ":") + 1, 2))
    Sec& = 0
    L% = LEN(T$)
    IF L% > 6 THEN
        Sec& = VAL(MID$(T$, L% - 1))
    END IF

    IntegerTIME& = 3600 * Hr& + 60 * Min& + Sec&

END FUNCTION

'*****************************************************************************************************************
                                                                                    FUNCTION KeyCode%
'   General keyboard scan code interpreter. For ALT keys add 1000.
'
' CALLed by:        PlotRadReturn;      Waterfall

FUNCTION KeyCode% (k$)                          ' [11/7/91]
    IF LEN(k$) = 1 THEN
        KeyCode% = ASC(UCASE$(k$))
    ELSEIF LEN(k$) = 2 THEN
        KeyCode% = ASC(RIGHT$(k$, 1)) + 1000
    END IF
END FUNCTION

'*****************************************************************************************************************
                                                                                    SUB Keystroke
' CALLed by:        StepBackwards

SUB Keystroke (C%)

        k$ = INKEY$

        IF LEN(k$) = 1 THEN
            C2% = ASC(UCASE$(k$))
            IF C2% = 70 OR C2% = 66 OR C2% = 27 THEN
                C% = C2%
            END IF
        ELSEIF LEN(k$) = 2 THEN                         ' ALT keystroke
            C2% = ASC(RIGHT$(k$, 1))
            IF C2% = 48 OR C2% = 33 THEN
                C% = C2%
            END IF
        END IF

END SUB
```

```
' Set the color levels for the waterfall display.

' CALLed by:      ColorLevels
'
SUB Levels (LevelChoice%)

   SELECT CASE LevelChoice%

      CASE 1!

         Level%(0) = 8: Level%(1) = 14: Level%(2) = 16: Level%(3) = 18
         Level%(4) = 21: Level%(5) = 24: Level%(6) = 26: Level%(7) = 29
         Level%(8) = 32: Level%(9) = 37: Level%(10) = 42: Level%(11) = 48
         Level%(12) = 56: Level%(13) = 63: Level%(14) = 70

      CASE 2!

         Level%(0) = 9: Level%(1) = 11: Level%(2) = 13: Level%(3) = 15
         Level%(4) = 17: Level%(5) = 19: Level%(6) = 21: Level%(7) = 23
         Level%(8) = 25: Level%(9) = 27: Level%(10) = 29: Level%(11) = 31
         Level%(12) = 33: Level%(13) = 35: Level%(14) = 50

      CASE 3!

         Level%(0) = 10: Level%(1) = 11: Level%(2) = 12: Level%(3) = 13
         Level%(4) = 14: Level%(5) = 15: Level%(6) = 16: Level%(7) = 17
         Level%(8) = 18: Level%(9) = 20: Level%(10) = 22: Level%(11) = 24
         Level%(12) = 26: Level%(13) = 40: Level%(14) = 60

      CASE 4!

         Level%(0) = 10: Level%(1) = 14: Level%(2) = 16: Level%(3) = 18
         Level%(4) = 20: Level%(5) = 22: Level%(6) = 24: Level%(7) = 26
         Level%(8) = 28: Level%(9) = 30: Level%(10) = 33: Level%(11) = 35
         Level%(12) = 45: Level%(13) = 55: Level%(14) = 80

      CASE 5!

         Level%(0) = 15: Level%(1) = 30: Level%(2) = 50: Level%(3) = 60
         Level%(4) = 70: Level%(5) = 80: Level%(6) = 85: Level%(7) = 90
         Level%(8) = 95: Level%(9) = 100: Level%(10) = 110: Level%(11) = 120
         Level%(12) = 130: Level%(13) = 140: Level%(14) = 220

      CASE 6!

         Level%(0) = 10: Level%(1) = 12: Level%(2) = 14: Level%(3) = 16
         Level%(4) = 18: Level%(5) = 20: Level%(6) = 23: Level%(7) = 27
         Level%(8) = 30: Level%(9) = 35: Level%(10) = 40: Level%(11) = 45
         Level%(12) = 50: Level%(13) = 60: Level%(14) = 75

   END SELECT

END SUB
```

```
'   Merge GPS, INS, and IF gain change data located in pc files with an EXABYTE ROWS file and save the merged
'   file on a second EXABYTE. Certain errors in the raw data relating to the pc real-time clock and the antenna
'   shaft angle encoder are adjusted.

' CALLed by:        ExbFile
'
SUB MERGE

    DIM GainTime(100, 2) AS LONG

    StartTime = TIMER

    INFOFILE MissionName$, MissionDate$, MissionTime&, GainTime&(), InsFileName$, GpsFileName$, InsTimeLag!

    IF InsFileName$ <> "" THEN
        InsFile% = FREEFILE
        OPEN InsFileName$ FOR BINARY AS #InsFile%
        InsData$ = INPUT$(44, #InsFile%)
        InsTime& = CVL(MID$(InsData$, 1, 4)) / TicksPerSec + VAL(InsTimeLag$)
    ELSE
        InsTime& = -1
    END IF

    IF GpsFileName$ <> "" THEN
        GpsFile% = FREEFILE
        OPEN GpsFileName$ FOR INPUT AS #GpsFile%
        LINE INPUT #GpsFile%, GpsData$
        GpsTime& = TimeGPS&(GpsData$)
    ELSE
        GpsTime& = -1
    END IF

    CALL STARTINGTIME(StartTime&, MissionTime&, InsTime&, GpsTime&)

    IF InsFileName$ <> "" THEN SEEK #InsFile%, 1
    IF GpsFileName$ <> "" THEN SEEK #GpsFile%, 1

                                                    'Construct the new header for the merge file.
    MergeHeader$ = NewHEADER$

                                                    'Construct string for mission name and date information.
    NewNameDate$ = NameDATE$(MissionName$, MissionDate$)

    FileSource$ = "EXB"

    ExbReset (WriteDriveId%)

    CALLS SETTGT(ReadDriveId%, 0)

    CLS
    PresentFile
    RowsTime2& = 0

                                                    ' Step through ROWS file until start of first full second.
    DO
        RowsTime1& = RowsTime2&
        ReadRecord
        RowsTime$ = PcClockTime$(MID$(NewRecord, 757, 3))
        IF RowsTime$ = "FF:FF:FF" THEN
            RowsTime$ = RowsTime3$
            MID$(NewRecord, 757, 3) = RowsTime$
```

```
    END IF
    RowsTime3$ = RowsTime$
    RowsTime2& = IntegerTIME&(RowsTime$)

                                                ' Assume mission starting time is always entered as
                                                ' earlier than the time of the first radar waveform.
    IF RowsTime2& < StartTime& THEN RowsTime2& = RowsTime2& + 86400
LOOP UNTIL RowsTime2& = RowsTime1& + 1

Dt% = 0
RowsT2# = RowsTime2&

InsTime# = InsTime&
Gain% = 1                                       ' Gain array index
END$ = "  "
RecordsMerged& = 0

                                                ' Primary loop to merge successive ROWS records with the ancillary data.
DO

    MergeINS NewRecord, MergeHeader$, StartTime&, RowsT2#, InsTime#, InsFile%, InsTimeLag$
    MergeGPS NewRecord, StartTime&, RowsTime1&, RowsTime2&, GpsTime&, GpsFile%

                                                ' Merge IF gain with ROWS record.
    DO WHILE GainTime&(Gain% + 1, 2) <> 0 AND RowsTime2& >= GainTime&(Gain% + 1, 2)
        Gain% = Gain% + 1
    LOOP
    MID$(NewRecord, 766, 1) = MKI$(GainTime&(Gain%, 1))

                                                ' Put mission name and date in ROWS record.
    MID$(NewRecord, 1007, 18) = NewNameDate$

                                                ' Put new modified header in ROWS record.
    MID$(NewRecord, 1, 50) = MergeHeader$

                                                ' Correct the antenna shaft angle and put corrected value in the ROWS record.
    MID$(NewRecord, 767, 2) = ShaftAngle$

                                                ' Write the merged record to the target tape.
    CALLS SETTGT(WriteDriveId%, 0)
    DriveReady
    WriteRecord "EXB", -1
    RecordsMerged& = RecordsMerged& + 1
    LOCATE 15, 30
    PRINT "RECORDS MERGED: "; RecordsMerged&

                                ' If previous record had no INS data (header ID was -9999) then reset INS header ID bytes to 1011.
    IF MID$(MergeHeader$, 45, 2) = NoData$ THEN
        MID$(MergeHeader$, 45, 2) = MKI$(1011)
    END IF

    RowsTime1& = RowsTime2&

                                                ' Read next record from the source tape.
    CALLS SETTGT(ReadDriveId%, 0)
    DriveReady
    ReadRecord
    CALLS XSENSE(SenseStatus%(0), IERR%)

    IF SenseStatus(2) = 0 THEN
```

```
        RowsTime$ = PcClockTime$(MID$(NewRecord, 757, 3))

        IF RowsTime$ = "FF:FF:FF" THEN
            RowsTime$ = RowsTime3$
            MID$(NewRecord, 757, 3) = RowsTime$
        END IF

        RowsTime3$ = RowsTime$

        RowsTime2& = IntegerTIME&(RowsTime$)
        IF RowsTime2& < StartTime& THEN RowsTime2& = RowsTime2& + 86400

        SELECT CASE RowsTime2& - RowsTime1&
            CASE IS = 0
                RowsT2# = RowsTime2& + Dt% / 100      ' Assumes 100 pulses/sec
                Dt% = Dt% + 1
            CASE IS = 1
                RowsT2# = RowsTime2&
                Dt% = 1
        END SELECT

    ELSEIF (SenseStatus(2) AND 128) OR (SenseStatus(2) AND 8) THEN

                                                  ' EOF or "blank" bytes encountered during last read.

        LOCATE 11, 22
        PRINT "Writing last record."
        CALLS SETTGT(WriteDriveId%, 0)
        WriteRecord "EXB", -1
        LOCATE 19, 30
        PRINT "Writing EOF to tape."
        CALLS XEOF(IERR%)
        LOCATE 20, 31
        PRINT "END OF TAPE COPY."
        END$ = "END"

    ELSE                                          ' Some type of error has occured.

        LOCATE 12, 25
        PRINT "SYSTEM ERROR: "; SenseStatus(2); " HAS OCCURED."
        END$ = "END"

    END IF

LOOP WHILE END$ <> "END"

Et = TIMER - StartTime
Hr = FIX(Et / 3600!)
Min = FIX((Et - 3600! * FIX(Et / 3600!)) / 60!)

LOCATE 17, 25
PRINT "Tape copy time is ";
PRINT USING "##_:"; Hr; Min;
PRINT USING "##"; FIX(Et - 3600! * Hr - 60! * Min)

CLOSE
IDLE

END SUB
```

```
'  Merge GPS data on DOS disk file with ROWS data in an EXABYTE file.
'
' CALLed by:        MERGE
'
SUB MergeGPS (MR$, ST&, RT1&, RT2&, GpsTime&, GpsFile%)

                                        ' Read GPS data and synchronize time with ROWS

   DO WHILE GpsTime& < RT2&
      IF EOF(GpsFile%) THEN EXIT DO
      LINE INPUT #GpsFile%, GpsData$
      GpsTime& = TimeGPS&(GpsData$)
      IF GpsTime& < ST& THEN GpsTime& = GpsTime& + 86400
   LOOP

                                        ' Put GPS latitude and longitude data into ROWS record.
   MID$(MR$, 771, 8) = NoDataLong$ + NoDataLong$
   IF GpsTime& = (RT2& - RT1&) * RT2& THEN
      MID$(MR$, 771, 8) = GPSLatLong(GpsData$)
   END IF

END SUB
```

****


```
'  Merge 44 bytes of "INS" data with a ROWS record. If the time difference between the INS data and the ROWS
'  data is greater than two pc "ticks" then a "NO DATA" (-9999) is entered in the ROWS record.
'
' CALLed by:        MERGE
'
SUB MergeINS (MR$, MH$, ST&, RT2#, InsTime#, InsFile%, InTL$)

   STATIC InsData AS STRING

                                        ' Read INS data and synchronize time with ROWS time.

   DO WHILE (InsTime# < RT2# - 2 / TicksPerSec)
      IF EOF(InsFile%) THEN EXIT DO
      InsData$ = INPUT$(44, #InsFile%)
      Ticks = CVL(MID$(InsData$, 1, 4))
      InsTime# = Ticks / TicksPerSec + VAL(InTL$)

                                        ' Past midnight?
      IF InsTime# < ST& THEN InsTime# = InsTime# + 86400!
   LOOP

                                        ' Put INS data in present ROWS record.
   MID$(MR$, 779, 44) = InsData$
   IF ABS(InsTime# - RT2#) > 2 / TicksPerSec THEN
      MID$(MH$, 45, 4) = NoData$ + MKI$(80)
      MID$(MR$, 779, 80) = STRING$(80, 0)
   END IF

END SUB
```

SUB Message

```
'   Print a message to the screen.

' CALLed by:     ASciIPitchRoll;     Blank;              ColorSelection;
                 CopyExbToDos;       DosFile;            DosPreProcessing;
                 ExbFile;            ExbPreProcessing;   ExbReset
                 ExbStatus           ExbTapeLog;         FileN
                 FileOpen;           IDLE;               INFOFILE;
                 NoFile;             OpenExistingFile;   OpenNewFile;
                 RecordN;            Rewind;             ValidROWSFile;
                 Version;            Waterfall

'
SUB Message (Row%, Note$)

    LOCATE Row%, 1
    PRINT BlankLine$
    LOCATE Row%, 42 - LEN(Note$) / 2
    COLOR 4, 7
    PRINT Note$;
    COLOR 7, 0

END SUB
```

****

SUB MINMAX

```
'   Find the minimum and maximum elements and element indicies of a one dimensional array of size N.
'
' CALLed by:     StartingTime

SUB MINMAX (A&(), N, Amin&, Imin, Amax&, Imax)

    REDIM A(N)

    Amax& = A&(1): Imax = 1: Amin& = A&(1): Imin = 1
    i = 1

    DO
        i = i + 1
        IF A&(i) > A&(i - 1) THEN
            Amax& = A&(i)
            Imax = i
        ELSE
            Amin& = A&(i)
            Imin = i
        END IF
    LOOP UNTIL i = N

END SUB
```

```
'    Insert mission name and date in merged file.

' CALLed by:        MERGE
'
FUNCTION NameDATE$ (N$, D$)

    A$ = SPACE$(10)
    B$ = SPACE$(8)

                                                    ' Insert the mission name (10 character field)
    MID$(A$, 1, 10) = RTRIM$(MID$(N$, 1))

                                                    ' Insert the mission date as --\--\--.
    B$ = RTRIM$(MID$(D$, 1, 8))
    MID$(B$, 1, 8) = "00\0" + RIGHT$(B$, 4)

    IF LEN(B$) = 8 THEN
        MID$(B$, 1, 8) = B$
    ELSEIF LEN(B$) = 6 THEN
        MID$(B$, 1, 1) = MID$(B$, 1, 1)
        MID$(B$, 1, 1) = MID$(B$, 3, 1)
    ELSEIF LEN(B$) = 7 THEN
        MID$(B$, 1 - INSTR(B$, "\"), 3) = MID$(B$, 1, 3)
    ELSE
        MID$(B$, 1, 8) = "-9999999"                 'No mission date
    END IF

    NameDATE$ = A$ + D$

END FUNCTION
```

```
'  Create a header for the merged records. Header types and lengths are those used during the ERS-1
'  underflight mission to Newfoundland during November, 1991.
'
' CALLed by:      MERGE

FUNCTION NewHEADER$

    DIM NewHeaderString AS STRING * HeaderLength
    DIM Head(2, (HeaderLength% - 2) / 4) AS INTEGER

    Head%(1, 1) = 0:        Head%(2, 1) = 50        ' Header
    Head%(1, 2) = 1:        Head%(2, 2) = 700       ' Radar waveform
    Head%(1, 3) = 2:        Head%(2, 3) = 4         ' pc "tick" time
    Head%(1, 4) = 3:        Head%(2, 4) = 2         ' "main bang" to first radar point time delay
    Head%(1, 5) = 10020:    Head%(2, 5) = 5         ' pc real-time clock with FF:FF:FF corrected
    Head%(1, 6) = 6:        Head%(2, 6) = 2         ' Aircraft roll angle
    Head%(1, 7) = 5:        Head%(2, 7) = 2         ' Aircraft pitch angle
    Head%(1, 8) = 1001:     Head%(2, 8) = 1         ' IFGain
    Head%(1, 9) = 10008:    Head%(2, 9) = 2         ' "corrected" antenna shaft angle
    Head%(1, 10) = 1009:    Head%(2, 10) = 4        ' GPS Latitude
    Head%(1, 11) = 1010:    Head%(2, 11) = 4        ' GPS Longitude
    Head%(1, 12) = 1011:    Head%(2, 12) = 80       ' INS data string

    FOR j = 0 TO 1
        FOR k = j + 2 ^ j TO HeaderLength - 2 STEP 4
            MID$(NewHeaderString$, k, 2) = MKI$(Head%(j + 1, (k + 3 - 2 * j) / 4))
        NEXT k
    NEXT j

                                                    'Teminate header with -1 (ie., hex FFFF)
    MID$(NewHeaderString$, HeaderLength - 1, 2) = MKI$(-1)

    NewHEADER = NewHeaderString$

END FUNCTION

'*************************************************************************************************************
                                                                                                  SUB NoFile
' CALLed by:      DosFile

SUB NoFile

    Message 17, " NO OPEN FILE."
    Message 18, "Use Function Key 2 to open a file."
    Message 20, "PRESS ANY KEY TO CONTINUE."
    x$ = "": DO UNTIL x$ <> "": x$ = INPUT$(1): LOOP

END SUB
```

```
                                                                                    SUB OpenExistingFile
' CALLed by:        DosFile

SUB OpenExistingFile

    FileOpen Close1$
    IF Close1$ = "N" THEN EXIT SUB

    DO
        LOCATE 18, 1: PRINT BlankLine$
        Message 17, "Enter the NAME of the file you want to OPEN"
        EnterFileName Active.FileName, Valid$
    LOOP WHILE Valid$ = "N"

    OPEN RTRIM$(Active.FileName) FOR BINARY AS #1

    SEEK #1, LOF(1) - 1023
    ReadRecord
    DecodeRecord
    Active.TEnd$ = ClockTime$
    Active.TotalRecords = LOF(1) / 1024

    SEEK #1, 1
    ReadRecord
    DecodeRecord
    Active.Record = 2
    Active.ByteLoc = ByteLoc
    Active.TStart$ = ClockTime$

    Dos = Active

END SUB

'*****************************************************************************************************************
                                                                                    SUB OpenNewFile
' CALLed by:        AscIIPitchRoll;       CopyExbToDos

SUB OpenNewFile (FileName$)

    ValidName$ = ""

    DO

        FileNo% = FREEFILE
        EnterFileName FileName$, Valid$
        IF LTRIM$(FileName$) = "" THEN EXIT SUB
        OPEN RTRIM$(FileName$) FOR BINARY AS #FileNo%

        IF LOF(FileNo%) <> 0 THEN

            Message 17, "File " + RTRIM$(FileName$) + " already exists"
            Message 18, "Enter a new filename"
            Message 22, "PRESS ANY KEY TO CONTINUE"
            x$ = "": DO: x$ = INPUT$(1): LOOP UNTIL x$ <> ""
            ValidName$ = "N"

        END IF

        CLOSE #FileNo%
    LOOP WHILE ValidName$ = "N"
END SUB
```

SUB ParameterDisplay

```
' Display experiment parameters and variables during screen plotting of the radar return.
'
' CALLed by:        StepBackwards;      StepForwards
'
SUB ParameterDisplay

    LOCATE 12, 3
    PRINT "Clock time: "; :                          COLOR 11
    PRINT ClockTime$; :                              COLOR 15
    PRINT "     Pulse #: "; :                        COLOR 11
    PRINT USING "**#####"; Active.Record - 1; :      COLOR 15
    PRINT "  Pointing angle:  "; :                   COLOR 11
    PRINT USING "###.#"; Dta.ShaftAngle!:            COLOR 15
    LOCATE 13, 3
    PRINT "Tick time : "; :                          COLOR 11
    PRINT Dta.TickTime; :                            COLOR 15
    LOCATE 13, 49
    PRINT "Pitch angle:  "; :                        COLOR 11
    PRINT USING "###.##"; Dta.Pitch:                 COLOR 15
    LOCATE 14, 49
    PRINT " Roll angle:  "; :                        COLOR 11
    PRINT USING "###.##"; Dta.Roll:                  COLOR 15
    LOCATE 16, 3
    PRINT "Trigger delay "; :                        COLOR 11
    PRINT USING "##### \     \"; Dta.TriggerDelay; :          COLOR 15:
    PRINT " meters."

                                                     ' Decode and display GPS latitude and longitude, if present
    IF CVI(MID$(Dta.Header, 37, 2)) = 1009 OR CVI(MID$(Dta.Header, 41, 2)) = 1010 THEN
        GPSDisplay MID$(NewRecord, 771, 8)
    END IF

END SUB

'*****************************************************************************************************************
```

FUNCTION PcClockTime$

```
' Decode the bytes that contain the output from the pc real-time clock. The three bytes that contain the data
' are encoded in binary coded decimal format with the lowest byte having hours, the middle byte having
' minutes, and the highest byte having seconds.

' CALLed by:        DecodeRecord;       MERGE

FUNCTION PcClockTime$ (PcClTime$)

    DIM T AS STRING * 8

    T$ = "00:00:00"

    Hrs$ = HEX$(CVI(MID$(PcClTime$, 1, 1) + CHR$(0)))
    Min$ = HEX$(CVI(MID$(PcClTime$, 2, 1) + CHR$(0)))
    Sec$ = HEX$(CVI(MID$(PcClTime$, 3, 1) + CHR$(0)))

    MID$(T$, 3 - LEN(Hrs$), LEN(Hrs$)) = Hrs$
    MID$(T$, 6 - LEN(Min$), LEN(Min$)) = Min$
    MID$(T$, 9 - LEN(Sec$), LEN(Sec$)) = Sec$

    PcClockTime$ = T$

END FUNCTION
```

```
                                                                                    FUNCTION PcTime$
' Decode the Pc computer "tick" time based on 18.20648 ticks per second. PcT& = CVL(MID$(NewRecord$, 751, 4))
' is passed to this function. (What determines when the tick count is zeros?)

' CALLed by:        DecodeRecord;       ExbTapeLog
'
FUNCTION PcTime$ (PcT&)

    DIM T AS STRING * 11
    CONST MaxTicks& = 24 * TicksPerSec * 3600

    T$ = "00:00:00.00"

    SELECT CASE PcT&

        CASE IS > MaxTicks&

            T$ = "BAD TIME   "

        CASE ELSE

            Hr# = PcT& / (TicksPerSec * 3600)
            Min# = (Hr# - INT(Hr#)) * 60!
            Sec# = (Min# - INT(Min#)) * 60!

            Hrs$ = LTRIM$(STR$(INT(Hr#)))
            Min$ = LTRIM$(STR$(INT(Min#)))

            Sec$ = LTRIM$(STR$(INT(Sec#)))
            HundredthSecs$ = LTRIM$(STR$(INT(Sec# * 100) - 100 * INT(Sec#)))

            MID$(T$, 3 - LEN(Hrs$), LEN(Hrs$)) = Hrs$
            MID$(T$, 6 - LEN(Min$), LEN(Min$)) = Min$
            MID$(T$, 9 - LEN(Sec$), LEN(Sec$)) = Sec$
            MID$(T$, 12 - LEN(HundredthSecs$), LEN(HundredthSecs$)) = HundredthSecs$

    END SELECT

    PcTime$ = T$

END FUNCTION

*****************************************************************************************************************
                                                                                    FUNCTION PitchRoll!
' Convert the pitch and roll in degrees for a 14 bit shaft angle encoder. The 120 degrees is subtracted because the
' shaft angle encoder was incorrectly wired for the ERS1 mission in November of 1991.

' CALLed by:        DecodeRecord
'
FUNCTION PitchRoll! (A$)

    PitchRoll! = 360! * CVL(A$ + CHR$(0) + CHR$(0)) / (2 ^ 14) - 120!

END FUNCTION
```

```
' Plot the radar waveform and display auxiliary data below the waveform. The user can step forward one record at
' a time, step backward one record at a time, or step continuously forward.
'
' CALLed by:      DosPreProcessing;  ExbPreProcessing

SUB PlotRadReturn                                   ' [12/10/91]

    KEY OFF

    BegEndByte WaveformStart%, WaveformStop%

    StartByte% = WaveformStart% + HeaderLength%
    StopByte% = WaveformStop% + HeaderLength%

    SCREEN 12
    VIEW (1, 1)-(600, 128)
    WINDOW (StartByte%, 256)-(StopByte%, 1)
    VIEW PRINT 12 TO 30

    Help2
    DO: LOOP WHILE INKEY$ <> ""
    Code% = 0

    DO
        Code2% = KeyCode(INKEY$)

        SELECT CASE Code2%

                                                    ' Select valid keys for interrupt. 1000 is added to the ALT keycodes.

            CASE 1048, 1033, 70, 66, 27
                Code% = Code2%

        END SELECT

        IF FileSource$ = "EXB" THEN
            CALLS XSENSE(SenseStatus(0), IERR%)
            IF SenseStatus(2) <> 0 THEN
                CheckExbStatus SenseStatus(2)
                Help2
            END IF
        END IF

        SELECT CASE Code%

            CASE 70, 1033                           ' [F] forward one record
                                                    ' [ALT-F] continuous forward

                StepForwards Code%, StartByte%, StopByte%

            CASE 66, 1048                           ' [B] backwards 1 record
                                                    ' [ALT-B] continuous backwards

                                                    ' Don't step the EXABYTE 8200 backwards

                IF FileSource$ = "DOS" THEN
                    StepBackwards Code%, StartByte%, StopByte%
                ELSE
                    Code% = 0
                END IF
```

```
        CASE 27                                 ' [ESC] exit display
            EXIT DO
        CASE ELSE                               ' invalid key
    END SELECT

    Help2

    IF FileSource$ = "DOS" AND Active.Record > Active.TotalRecords THEN
        VIEW PRINT
        LOCATE 26, 28
        PRINT "END OF FILE"
        DO
            Code% = KeyCode(INKEY$)
        LOOP UNTIL Code% = 66 OR Code% = 1048 OR Code% = 27
        LOCATE 26, 28: PRINT "          "
    END IF

LOOP

SCREEN 0
KEY ON

END SUB
```

```
' CALLed by:      AsciIPitchRoll;          CopyExbToDos;     DosFile;
                  DosPreProcessing;  ExbFile; ExbPreProcessing;
                  FileN;                   FirstMenu;        MERGE;
                  RecordN

SUB PresentFile

    RowOffSet% = 4
    ColOffSet% = 16

    SELECT CASE FileSource$

        CASE "DOS"

            Active = Dos

        CASE "EXB"

            Active = Exb

        CASE ELSE

            ResetActiveStats
            ResetDta

    END SELECT

    RecordNumber& = Active.Record - 1
    COLOR 3, 0: LOCATE RowOffSet%, ColOffSet% - 1
    PRINT CHR$(201); STRING$(52, CHR$(205)); CHR$(187)

    FOR i = 0 TO 4
        LOCATE RowOffSet% + i + 1, ColOffSet% - 1: PRINT CHR$(186)
        LOCATE RowOffSet% + i + 1, ColOffSet% + 52: PRINT CHR$(186)
    NEXT
    LOCATE RowOffSet% + 6, ColOffSet% - 1
    PRINT CHR$(200); STRING$(52, CHR$(205)); CHR$(188)
    COLOR 7, 0:     LOCATE RowOffSet% + 1, ColOffSet% + 1
    PRINT "FILE:"
    COLOR 4, 7:     LOCATE RowOffSet% + 1, ColOffSet% + 7
    PRINT RTRIM$(Active.FileName)
    COLOR 7, 0:     LOCATE RowOffSet% + 2, ColOffSet% + 1
    PRINT "TOTAL RECORDS: ";
    COLOR 4, 7:     PRINT Active.TotalRecords
    COLOR 7, 0:     LOCATE RowOffSet% + 3, ColOffSet% + 1
    PRINT "STARTING TIME: ";
    COLOR 4, 7:     PRINT " " + Active.TStart + " ";
    COLOR 7, 0:     PRINT " ENDING TIME: ";
    COLOR 4, 7:     PRINT " " + Active.TEnd + " "
    COLOR 7, 0:     LOCATE RowOffSet% + 5, ColOffSet% + 1
    PRINT "PRESENT RECORD #: ";
    COLOR 4, 7:     PRINT RecordNumber&;
    COLOR 7, 0:     PRINT "  TIME: ";
    COLOR 4, 7:     PRINT " " + ClockTime$
    COLOR 7, 0
END SUB
```

```
                                                                                    SUB ReadRecord
' CALLed by:      CopyExbToDos;    DosPreProcessing;   ExbPreProcessing;
                  ExbTapeLog;      FileN;                    MERGE;
                  OpenExistingFile; RecordN;                 StepBackwards;
                  StepForwards;    Waterfall

SUB ReadRecord

   SELECT CASE FileSource$

      CASE "DOS"
         NewRecord = INPUT$(1024, 1)

      CASE "EXB"
         CALLS XRCHR(NewRecord, 1, IERR%)

   END SELECT
   Active.Record = Active.Record + 1

END SUB

'*****************************************************************************************************************
                                                                                    SUB RecordN
' Move record pointer to record number N. For EXABYTE files the pointer is positioned
' at the end of record N while for DOS files the pointer is at the beginning of record N.
'
' CALLed by:      AscIIPitchRoll;     CopyExbToDos;     DosPreProcessing;
'                 ExbPreProcessing
'
SUB RecordN

   S& = 0!

   SELECT CASE FileSource$

   CASE "DOS"

      DO

         CLS
         PresentFile
         Message 17, "Enter the first record number for processing."
         EnterRecord S&
         IF S& = Active.Record - 1 THEN
            EXIT SUB
         END IF

         IF S& > LOF(1) / 1024 THEN
            PresentFile
            Text$ = "Your starting record number is beyond the end of the file."
            Message 17, Text$
            Message 18, "Enter a new number."
            BEEP
            SLEEP 1
         END IF

      LOOP WHILE S& > LOF(1) / 1024
      Active.Record = S&
      SEEK #1, (S& - 1) * 1024 + 1

   CASE "EXB"
```

```
RecordsMoved& = 0
Message 17, "Enter the first record number for processing."
EnterRecord S&
IF S& = Active.Record - 1 THEN
    EXIT SUB
END IF

IF S& = 1 AND Active.FileNo = 1 THEN
    Rewind
    ResetExbStats
    Active = Exb
    ReadRecord
    DecodeRecord
    Active.TStart = ClockTime$
    EXIT SUB
END IF

IF Active.Record = 0 THEN
    ReadRecord
    DecodeRecord
    Active.TStart = ClockTime$
END IF

Jump& = S& - Active.Record
MoveRecords% = SGN(Jump&) * 32000
IF (Jump& < 0) AND (S& < 500) AND Active.Record > 1000 THEN
                                        ' If it is necessary to move backwards to near the beginning of the file,
                                        ' it is quicker to go to the first record and then step forward.
    CALLS XSKIPF(-1, IERR%)
    CALLS XSKIPF(1, IERR%)
    Jump& = S& - 1
END IF

CALLS XSENSE(SenseStatus(0), IERR%)

DO WHILE (ABS(Jump&) > 0) AND (SenseStatus(2) < 128)
    IF ABS(Jump&) <= 32000 THEN MoveRecords% = Jump&
    CALLS XSPACE(MoveRecords%, IERR%)
    RecordsMoved& = RecordsMoved& + MoveRecords%
    Jump& = Jump& - MoveRecords%
    DecodeRecord
    Update
LOOP

CALLS XSENSE(SenseStatus(0), IERR%)

IF SenseStatus(0) AND 128 THEN
                                        ' Test if trying to jump ahead more records than are in the file.
    NonRecords& = SenseStatus(4) * (256! ^ 2) + SenseStatus(5) * 256! + SenseStatus(6)
    RecordsMoved& = RecordsMoved& - NonRecords&
END IF
IF (SenseStatus(2) AND 128) THEN                ' If EOF.
    Active.TotalRecords = Active.Record + RecordsMoved&
    Message 17, "EOF reached."
    CALLS XSKIPF(-1, IERR%)
    CALLS XSPACE(-1, IERR%)
    Active.Record = Active.Record - 1
END IF

CALLS XSENSE(SenseStatus(0), IERR%)
```

```
      IF (SenseStatus(2) AND 8) THEN                    ' If EOD.
         Message 17, "EOD reached. Read head moved past last EOF."
         Text$ = "Tape will be positioned at end of last file."
         Message 18, Text$

         IF Active.Record = 0 THEN

            CALLS XSKIPF(-1, IERR%)
            CALLS XSPACE(-1, IERR%)
            RecordsMoved& = 0

         ELSE

            Message 19, "NOTE: Last file does not have an EOF."
            CALLS XSPACE(-1, IERR%)
            RecordsMoved& = 0

         END IF

      END IF

      Active.Record = Active.Record + RecordsMoved&
      ReadRecord
      DecodeRecord
      Update

      IF Active.Record = Active.TotalRecords THEN
         Active.TEnd = ClockTime$
      ELSEIF Active.Record = 1 THEN
         Active.TStart = ClockTime$
      END IF

      Active.ByteLoc = (Active.Record - 1) * 1024 + 1
      Exb = Active

   END SELECT

END SUB
```

```
                                                                                    SUB ResetActiveStats
' CALLed by:        MAIN;   FirstMenu;          PresentFile

SUB ResetActiveStats
    Active.FileName = SPACE$(40)
    Active.FileNo = 0
    Active.TStart = "       "
    Active.TEnd = "       "
    Active.Record = 0
    Active.TotalRecords = 0
    Active.ByteLoc = 0
END SUB

*****************************************************************************************************************
                                                                                    SUB ResetDosStats
' CALLed by:        MAIN;   DosFile;   EnterFileName;     FileOpen

SUB ResetDosStats
    CLOSE #1
    Dos.FileName = SPACE$(40)
    Dos.FileNo = 0
    Dos.TStart = "       "
    Dos.TEnd = "       "
    Dos.Record = 0
    Dos.TotalRecords = 0
    Dos.ByteLoc = 0
END SUB

*****************************************************************************************************************
                                                                                    SUB ResetDta
' CALLed by:        MAIN;   FirstMenu;          PresentFile

SUB ResetDta
    Dta.Header = SPACE$(HeaderLength%)
    Dta.Radar = SPACE$(700)
    Dta.AccurateTime = SPACE$(11)
    Dta.TickTime = SPACE$(11)
    Dta.TrueTime = SPACE$(8)
    Dta.TriggerDelay = 0
    Dta.Pitch = 0
    Dta.Roll = 0
    Dta.ShaftAngle = 0
END SUB

*****************************************************************************************************************
                                                                                    SUB ResetExbStats
' CALLed by:        MAIN;   ExbFile;   FileN;     RecordN

SUB ResetExbStats
    Exb.FileNo = 1
    Exb.TStart = "       "
    Exb.TEnd = "       "
    Exb.TotalRecords = 0
    Exb.FileName = RTRIM$("Exabyte file # ") + "1"
    Exb.Record = 1
    Exb.ByteLoc = 1
END SUB
```

```
                                                                                                    SUB Rewind
' CALLed by:        ExbTapeLog;        RecordN

SUB Rewind              ' Rewind EXABYTE tape to LBOT

    Message 17, "REWINDING TAPE."
    CALLS XREWND(IERR%)
    Message 17, "COMPLETED."

END SUB

'*****************************************************************************************************************
                                                                                                    FUNCTION RotationAngle!
' Decode the rotation angle of the antenna shaft as implemented after September 1991.

' CALLed by:        DecodeRecord;        ShaftAngle
'
'    INPUT:  binary coded decimal degrees to 0.1 degrees
'    OUTPUT: decimal degrees
'
FUNCTION RotationAngle! (RotAng$)                                                   ' [12/17/91]

    LeftBits% = CVI(MID$(RotAng$, 1, 1) + CHR$(0))
    RightBits% = CVI(MID$(RotAng$, 2, 1) + CHR$(0))
    Ang1! = 100! * ((RightBits% AND 48) \ 16) + 10! * (RightBits% AND 15)
    Ang2! = .1 * (LeftBits% AND 15) + 1! * ((LeftBits% AND 255) \ 16)

    RotationAngle! = Ang1! + Ang2!

END FUNCTION

'*****************************************************************************************************************
                                                                                                    SUB ShaftAngle$
'    "Correct" the raw antenna shaft angle values for missing bits. The corrected angle is accurate to about +/- 0.4 deg.
'
' CALLed by:        DecodeRecord;        MERGE

FUNCTION ShaftAngle$

    STATIC Angle1 AS SINGLE, Angle2 AS SINGLE

    Angle2 = Angle1
    Angle$ = MID$(NewRecord$, 767, 2)
    Angle1 = RotationAngle!(Angle$)

                                                    ' Test and correct for missing 1 deg. bit.
    IF (Angle2 - Angle1 < 0) OR (Angle2 - Angle1 >= 1) THEN          ' NOTE: This test doesn't work correctly
                                                                     ' when ' the angle goes across 360 degrees.
        Angle1 = Angle1 + 1!
    END IF

    a4% = INT((INT(Angle1)) / 100)
    a3% = INT((INT(Angle1) - a4% * 100) / 10)
    a2% = INT(Angle1) - a4% * 100 - a3% * 10
    a1% = INT((Angle1 - INT(Angle1)) * 10)
    BcdAngle% = a1% + (2 ^ 4) * a2% + (2 ^ 8) * a3% + (2 ^ 12) * a4%
    BcdAngle% = BcdAngle% + (CVI(Angle$) AND &HC000)

    ShaftAngle$ = MKI$(BcdAngle%)

END FUNCTION
```

SUB STARTINGTIME

```
'    Find the earliest time between the mission starting time, the first INS time, and the first GPS time.
'
' CALLed by:        MERGE
'
SUB STARTINGTIME (StartTime&, MT&, IT&, GT&)

    Tn = 3
    DIM Times(Tn) AS LONG

    Times&(1) = MT&: Times&(2) = IT&: Times&(3) = GT&

    ' Find the earliest time and use as the mission starting time. Algorithm assumes less than 2 hours (7200 secs) between the earliest
    ' and latest starting times for acquiring GPS and INS data and the operator entered "Mission" starting time.
    DO
        MINMAX Times&(), Tn, MinTime&, MinElement, MaxTime&, MaxElement
        IF MaxTime& > MinTime& + 7200 THEN
            Times&(MinElement) = Times&(MinElement) + 86400
        END IF
    LOOP UNTIL MaxTime& <= MinTime& + 7200

    StartTime& = Times&(MinElement)
    ERASE Times&

END SUB
```

****

SUB StepBackwards

```
'   Steps a DOS file backwards one record at a time. NOTE: With the present EXABYTE drive, it isn't
'   practical to step backward through a file one record at a time.
'
' CALLed by:        PlotRadarReturn

SUB StepBackwards (C%, StrBy%, StpBy%)

    Active.Record = Active.Record - 2

                                                    ' Test for beginning of file.
    IF Active.Record > 0 THEN
        SEEK 1, ByteLoc&
        ReadRecord
        DecodeRecord
        ParameterDisplay
        WaveFormPlot StrBy%, StpBy%
    ELSE                                            ' At beginning of file.
        Active.Record = 1
        SEEK 1, ByteLoc&
        ParameterDisplay
        WaveFormPlot 1, 1
        VIEW PRINT
        LOCATE 26, 28
        PRINT "BEGINNING OF FILE"
        DO
            Keystroke C%
        LOOP UNTIL C% = 70 OR C% = 33 OR C% = 27
        LOCATE 26, 28: PRINT "                 "
    END IF

    IF C% = 66 THEN C% = 0

END SUB
```

```
                                                                                SUB StepForwards
' CALLed by:        PlotRadarReturn

SUB StepForwards (C%, StrBy%, StpBy%)

    ReadRecord
    DecodeRecord
    ParameterDisplay
    WaveFormPlot StrBy%, StpBy%
    IF C% = 70 THEN C% = 0

END SUB

*****************************************************************************************************************
                                                                                FUNCTION TimeGPS&
' Read the time in ASCII from the GPS file and convert to a long integer. Eight
' seconds are subtracted from the time to bring it into correspondence with UTC.
'
' CALLed by:        MERGE; MergeGPS

FUNCTION TimeGPS& (D$)

    Tb% = INSTR(D$, ",")
    Te% = INSTR(Tb% + 1, D$, ",")
    Gps$ = MID$(D$, Tb% + 1, Te% - Tb% - 1)

    IF Te% - Tb% = 9 THEN

        Hr& = VAL(MID$(Gps$, 1, 2)) * 3600
        Min& = VAL(MID$(Gps$, 3, 2)) * 60
        Sec& = VAL(MID$(Gps$, 5)) - 8

        TimeGPS& = Hr& + Min& + Sec&

    ELSE                                    ' If GPS field is less than 8 characters put appropriate code here.

    END IF

END FUNCTION


*****************************************************************************************************************
                                                                                SUB Update
' CALLed by:        AscIIPitchRoll;      CopyExbToDos;      ExbPreProcessing;
                    RecordN

SUB Update

    COLOR 7, 0: LOCATE RowOffSet% + 5, ColOffSet% + 1
    PRINT "PRESENT RECORD #: ";
    COLOR 4, 7: PRINT Active.Record - 1;
    COLOR 7, 0: PRINT "  TIME: ";
    COLOR 4, 7: PRINT " " + ClockTime$;
    COLOR 7, 0: PRINT "   "                 ' Blanks some previous characters

END SUB
```

SUB ValidROWSFile

```
' CALLed by:        EnterFileName

SUB ValidROWSFile (Vn$, FileNo%)

    SELECT CASE LOF(FileNo%) / 1024

        CASE IS <> INT(LOF(FileNo%) / 1024)

            Message 17, "Probable non-ROWS or corrupted data file"
            Vn$ = "N"
            SLEEP 2

        CASE 0

            Message 17, RTRIM$(Active.FileName)
            Message 19, "does not exist"
            KILL Active.FileName
            Vn$ = "N"

        CASE ELSE

            Active.Record = 1
            Active.ByteLoc = 1
            Vn$ = "Y"

    END SELECT

END SUB
```

****

SUB Version

```
'    Display the PREROWS program version.          (9/23/92)
'
' Called by:        FirstMenu

SUB Version

    CLS
    Message 12, "PREROWS version 2 (10/13/92)"
    Message 14, "Use with ERS-1 mission and later"
    IDLE

END SUB
```

```
' Color coded waterfall display of ROWS pulse data.
'
' Convert a byte to a 16 bit integer by appending an ASCII 0 to the righthand byte of the word.

' CALLed by:      DosPreProcessing;  ExbPreProcessing
'
SUB Waterfall

    DIM FileByteLength AS LONG
    STATIC NumOfRecords&

    KEY OFF
    StartRecord& = Active.Record
    SELECT CASE FileSource$

        CASE "DOS"

            FileByteLength = Active.TotalRecords * 1024

        CASE "EXB"

            IF NumOfRecords& <= Active.Record + 1 THEN
                CLS
                Text$ = "Enter the largest valid record number for processing"
                Message 15, Text$
                INPUT ; NumOfRecords&
            END IF

            FileByteLength = NumOfRecords& * 1024

    END SELECT

    IF Active.Record >= (FileByteLength / 1024 - 1) THEN
        CLS
        Message 17, "NOTHING TO DO"
        SLEEP 2
        EXIT SUB
    END IF

    BegEndByte WaveformStart%, WaveformStop%
    ColorLevels

    StartByte% = WaveformStart% + HeaderLength%
    StopByte% = WaveformStop% + HeaderLength%

    ViewX1% = 30: ViewY1% = 40
    ViewX2% = 619: ViewY2% = 459
    WindowX1% = 0: WindowY1% = StartByte%
    WindowX2% = ViewX2%: WindowY2% = StartByte% + ViewY2%

    SCREEN 12

    VIEW (ViewX1%, ViewY1%)-(ViewX2%, ViewY2%)
    WINDOW (WindowX1%, WindowY1%)-(WindowX2%, WindowY2%)

    IF (FileByteLength - Active.ByteLoc + 1!) / 1024 >= WindowX2% THEN
        Wdth% = WindowX2%
    ELSE
        Wdth% = (FileByteLength - Active.ByteLoc + 1) / 1024
    END IF
```

```
    IF (StopByte% - StartByte%) > 479 THEN StopByte% = StartByte% + 479

    ColCoord% = ViewX1% / 8! - 3!
    RowCoord% = 25! - (1! * (StopByte% - StartByte% - ViewY1%) / 18!)

    StartRecord& = Active.Record
    ReadRecord
    DecodeRecord
    Active.ByteLoc = ByteLoc

    LOCATE RowCoord%, ColCoord%

    PRINT USING "\      \"; ClockTime$

    k = 0

    DO WHILE Active.Record < Wdth% + StartRecord& AND Active.Record <= FileByteLength - 1023
        k = k + 1
        IF KeyCode%(INKEY$) = 27 THEN EXIT DO
        ReadRecord
        DecodeRecord
        FOR j = StartByte% TO StopByte%
            PSET (k, j), ColorLevel(CVI(MID$(NewRecord, j, 1) + CHR$(0)))
        NEXT j
    LOOP

    LOCATE RowCoord%, Wdth% / (8! * WindowX2% / 620!) - 5
    PRINT USING "\      \"; ClockTime$

    DO: k$ = INKEY$: LOOP WHILE k$ <> ""
    DO: k$ = INKEY$: LOOP WHILE k$ = ""

    SCREEN 0
    Active.ByteLoc = ByteLoc

END SUB

'*****************************************************************************************************************
                                                                                                    SUB WaveFormPlot
' CALLed by:        StepBackwards;      StepForwards

SUB WaveFormPlot (StartByte%, StopByte%)

    DIM WaveForm(700) AS INTEGER
    REDIM WaveForm(StartByte% TO StopByte%) AS INTEGER

    CLS 1
    WaveForm%(StartByte%) = CVI(MID$(Dta.Radar, StartByte%, 1) + CHR$(0))

    FOR i = StartByte% + 1 TO StopByte%
        WaveForm%(i) = CVI(MID$(Dta.Radar, i - HeaderLength%, 1) + CHR$(0))
        LINE (i - 1, WaveForm%(i - 1))-(i, WaveForm%(i)), 10
    NEXT

END SUB
```

```
' CALLed by:        CopyExbToDos;      MERGE

SUB WriteRecord (FileSource$, FileNumber%)

    SELECT CASE FileSource$

    CASE "DOS"

        PUT FileNumber%, , NewRecord

    CASE "EXB"

        CALLS XWRCHR(NewRecord, 1, IERR%)
        DriveReady

    END SELECT

END SUB
```

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	January 1993	Technical Memorandum

4. TITLE AND SUBTITLE
Radar Ocean Wave Spectrometer (ROWS) Preprocessing Program (PREROWS2.EXE)
User's Manual and Program Description

5. FUNDING NUMBERS
972

6. AUTHOR(S)
C.R. Vaughn

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Laboratory for Hydrospheric Processes
Goddard Space Flight Center
Wallops Flight Facility
Wallops Island, Virginia

8. PERFORMING ORGANIZATION
93B00031

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Washington, D.C. 20546–0001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
TM–104579

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified–Unlimited
Subject Category 48

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This Technical Memorandum is a user's manual with additional program documentation for the computer program PREROWS2.EXE. PREROWS2 works with data collected by an ocean wave spectrometer that uses radar (ROWS) as an active remote sensor. The original ROWS data acquisition subsystem was replaced with a PC in 1990. PREROWS2.EXE is a compiled QuickBasic 4.5 program that unpacks the recorded data, displays various variables, and provides for copying blocks of data from the original 8mm tape to a PC file.

14. SUBJECT TERMS
Computer Program, User's Manual, ROWS, Radar, Oceanography

15. NUMBER OF PAGES
87

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	Unlimited