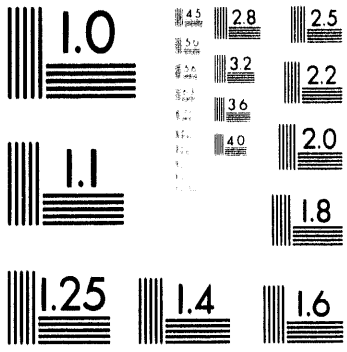

1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0

1 of 5

SAND93-2064
Unlimited Release
Printed September 1993

Distribution
Category UC-706

Data Collection System Volume 1: Overview and Operators Manual

Richard B. Caudell, Marvin E. Bauder,
William B. Boyer, Ralph E. French, Robert J. Isidoro,
Peter C. Kaestner, W. George Perkins

Instrumentation Development Department 9321
Sandia National Laboratories
Albuquerque, NM 87185

ABSTRACT

Sandia National Laboratories (SNL) Instrumentation Development Department was tasked by the Defense Nuclear Agency (DNA) to record data on Tektronix RTD720 Digitizers on the HUNTERS TROPHY field test conducted at the Nevada Test Site (NTS) on September 18, 1992. This report contains a overview and description of the computer hardware and software that was used to acquire, reduce, and display the data. The document is divided into two volumes: an overview and operators manual (Volume 1) and a maintenance manual (Volume 2).

CONTENTS

CONTENTS (Continued)

FIGURES

TABLES

ACRONYMS

ACL	Access Control List
ATC	Advanced Technology Center
CHN	Channel
DCL	Digital Command Language
DEC	Digital Equipment Corporation
DNA	Defense Nuclear Agency
DSP	digital signal processing
FMS	Forms Management System
GKS	Graphical Kernel System
ICF	instrument control file
KSC-1	Kaman Sciences Corporation
LPARL	Lockheed Palo Alto Research Laboratory
NTS	Nevada Test Site
SAIC	Science Applications International Corporation
SNL	Sandia National Laboratories
SPL	Software Project Leader

1.0 OVERVIEW OF RTD720 SOFTWARE

1.1 Introduction

The Sandia National Laboratories (SNL) Field Instrumentation Department has been tasked by the Defense Nuclear Agency (DNA) to record data on Tektronix RTD720 Digitizers on the HUNTERS TROPHY field test conducted at the Nevada Test Site (NTS) on September 18, 1992. This report contains a description of the computer hardware and software that was used to acquire, reduce, and display the data. The rest of this chapter contains an overview of the hardware and software in the Sandia RTD720 recording system. The remainder of the document presents detailed descriptions of the hardware and software. It is assumed that the reader and user are very familiar with Digital Equipment Corporation (DEC) VAX computers, the VMS operating system including DECnet, and the capabilities of the RTD720 digitizer. References are provided to specific VAX VMS and RTD720 manuals.

The data was recorded by SNL in parallel with primary data recorded by DNA contractors Science Applications International Corporation (SAIC), Kaman Sciences Corporation (KSC-1), and Lockheed Palo Alto Research Laboratory (LPARL). The SNL recorders were located in NTS Area 12, Building 909, and the contractor recorders were located in various recording alcoves in the HUNTERS TROPHY tunnel (downhole). Analog data recorded by SNL was transmitted from downhole to Building 12/909 using fiber-optic technology. SNL is also tasked to provide comparisons between the primary data recorded by DNA contractors and the secondary data recorded by SNL. Details of the comparison procedure are addressed in a separate document.

Each RTD720 digitizer can be configured with one, two, or four independently programmable data channels. Each data channel can be used to record several signals, the first of which could be a laser calibration pulse supplied by the fiber optics developers. We have provided software not only to record and display the data but also to provide for instrumentation setup.

SNL involvement in a DNA instrumentation modernization project began with the DIAMOND FORTUNE event. SNL commitment to the project has increased through the HUNTERS TROPHY field test. DNA's ultimate goal is to develop and field an uphole recording capability similar to SNL's.

1.2 Computer-Instrument Network

Computers and instruments associated with RTD720 testing are located in Building 12/909. The master computer in the RTD720 network is a DEC VAX 4000. A backup VAX (having less capability than the VAX 4000) was used for data acquisition when the VAX 4000 was not functioning properly during early dry runs for HUNTERS TROPHY.

The VAX 4000 is equipped with two ethernet interfaces. One ethernet interface connects to the Field Instrumentation Department NTS VAX network and the other connects to a data network, including several smaller computers or workstations. For HUNTERS TROPHY, we used several Sandia-owned

microVAX-II computers that were temporarily available. For subsequent tests, the microVAX-II computers will be replaced by SNL-purchased, DNA-owned workstations. Each microVAX-II is equipped with an ethernet interface and a DEC IEQ11 Dual IEEE-488 interface. All ethernet interfaces use the DECnet communication protocol (Digital Equipment Corp., 1988). Each microVAX-II uses the two IEEE-488 buses on the IEQ11 interface to communicate with the RTD720 digitizers.

The VAX 4000 has the DECnet node name "GEAR10." GEAR10 is an acronym composed of the following elements:

1. General purpose interface bus
2. Ethernet
3. Analog
4. Recording node
5. 10 - Serial number

Each microVAX-II has node name RMVm0, where m0 is a serial number (e.g., 10). For HUNTERS TROPHY, each contractor was assigned a separate node (RMVm0). This was done so that contractors could make dry runs by hand in their own area without having access to the data of other contractors. Node assignment was made as follows:

a. RMV10 - KSC-1
b. RMV20 - LPARL
c. RMV30 - SAIC

Figures 1-1 through 1-3 are channel diagrams for each RMV node. Eight RTD720s were allotted to KSC-1 experiments with 16 data channels. Within these data channels were 32 subchannels (i.e., separately recorded signals). Four RTD720s were assigned to LPARL data with 4 data channels and a total of 28 subchannels. SAIC data was assigned 14 RTD720s using 14 channels with a total of 68 subchannels.

The RTD720 control network was connected to the Sandia NTS VAX network for HUNTERS TROPHY. Many assets of the Sandia network were used to support the RTD720 recording project (e.g., INGRES data base, optical disk archiving, ethernet printers, etc).

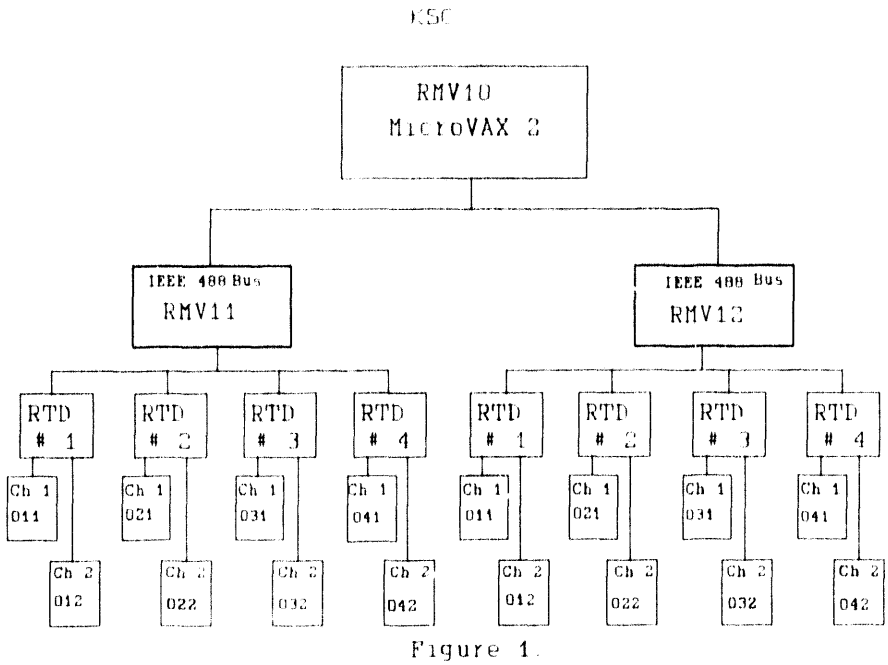


Figure 1-1. KSC-1 Data Channel Diagram

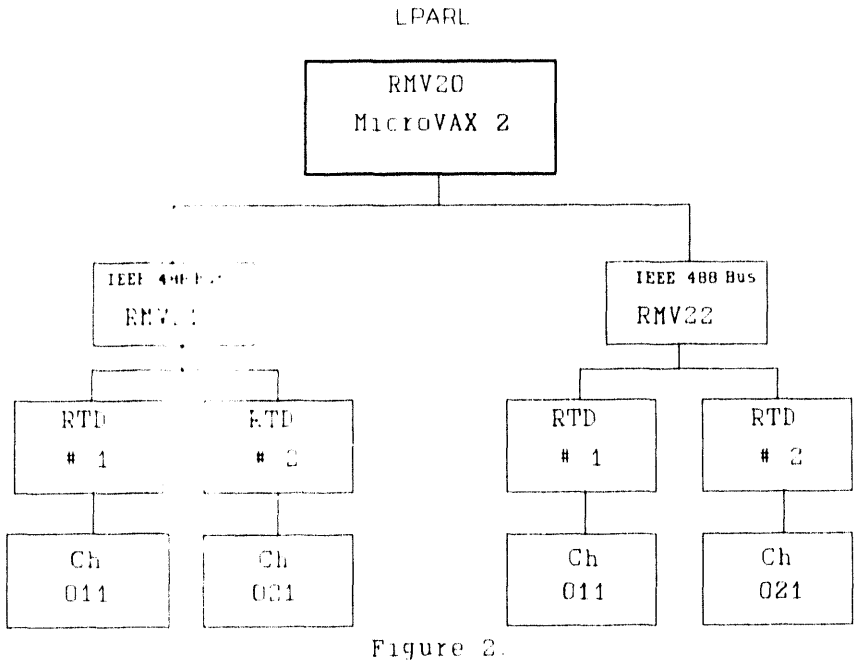


Figure 1-2. LPARL Data Channel Diagram

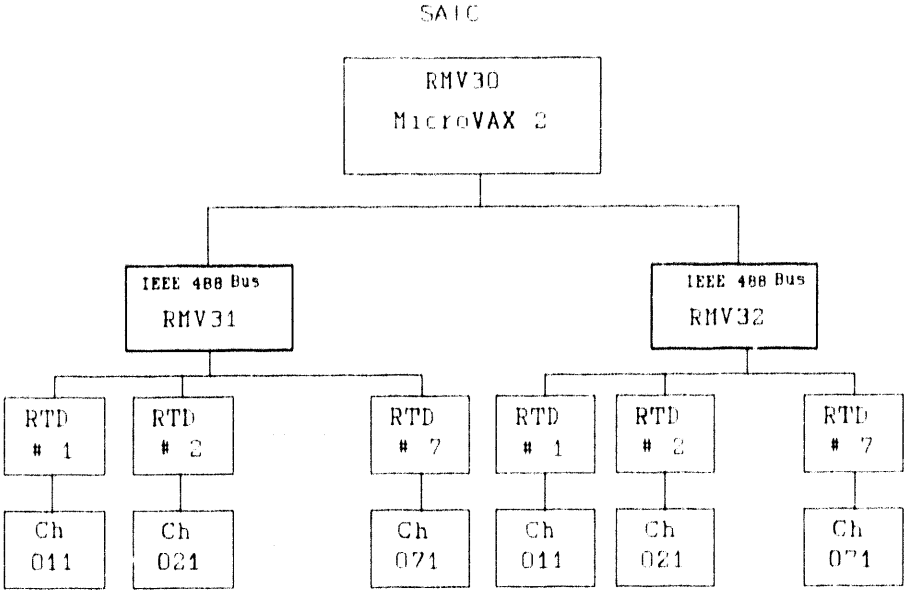


Figure 3

Figure 1-3. SAIC Data Channel Diagram

1.3 Instrumentation Data Base and Instrument Control Files

Data about each instrument, channel, and signal are contained in a data base that is maintained using an SNL-written INGRES application. The INGRES application is used to create ICFs to be used by other software to control access to instruments. In particular, the RTD720 instrument control files contain fields for all possible settings of each channel, as well as descriptions of all signals that may be multiplexed into each channel on every RTD720. The RTD720 control files are stored in a special GEAR10 directory designated DD:[TABLES]. Each control file contains information about all RTD720s that are connected via a single IEEE-488 interface to a given RMVm0. The control file associated with the *nth* IEEE-488 bus on RMVm0 is named RMVm*n*R20.TBL. There is one such file for each active IEEE-488 bus on any RMVm0. Each IEQ11 interface is equipped with two IEEE-488 buses with device designations IXA0: (bus 1) and IXA1: (bus 2). For the HUNTERS TROPHY configuration, the ICFs were named as follows:

Contractor	Bus	Instrument Control File
KSC-1	RMV11	RMV11R20.TBL
	RMV12	RMV12R20.TBL
LPARL	RMV21	RMV21R20.TBL
	RMV22	RMV22R20.TBL
SAIC	RMV31	RMV31R20.TBL
	RMV32	RMV32R20.TBL

1.4 CONTROL-Account Software

There are two types of nodes in the computer architecture for testing RTD720s. In this architecture the GEAR10 node is the "Master" node and there is a certain set of procedures that may be performed when logged into CONTROL on GEAR10. RMVm0 nodes in this architecture are "Slave" nodes and may not perform those procedures granted to the GEAR10 node. They are allowed to perform other procedures appropriate to "Slave" nodes. In either case, the CONTROL-account procedure begins when a user logs onto the node with username CONTROL. The CONTROL account on any node at the NTS is a captive account that allows a restricted group of users to perform a very limited set of activities specified in a menu. After username and password validation, the procedure presents a valid user with a menu of options appropriate to the type of node (Master or Slave).

1.4.1 Master Node Menu

If the user is logged into the "Master" node (GEAR10) then the menu is as follows:

(1) Run RTD720 Dry-Run Scheduler
(2) Run RTD-diagnostics
(3) Get new instrument-control files from data base node
(L) Logbook
(M) Mail
(E) Exit

If the user selects "Run RTD720 Dry-Run Scheduler" from the menu, the command procedure invokes the RTD720 scheduler, a software ensemble that includes both FORTRAN-coded executable images and DEC Digital Command Language (DCL) command procedures. The RTD720 scheduler and the procedures invoked are discussed below in Section 1.6 "Automated Dry-Run Data-Acquisition Software."

1.4.2 Slave Node Menu

If the user is logged into a "Slave" node (RMVm0) then the menu is as follows:

(1) Run RTDTEST
(2) Show users
(3) Get new instrument control files from data base node
(4) Push modified instrument-control files to data base node
(E) EXIT

If the user selects "Run RTDTEST" from the menu, the command procedure invokes the RTDTEST image. This option is described in Section 1.8 "SETUP (RTDTEST) SOFTWARE." The CONTROL housekeeping includes all necessary control-file copies to the instrumentation data base node.

It should be noted that the list of users authorized to use the GEAR10 CONTROL command procedure is not necessarily identical to the list authorized to use the RMVm0 CONTROL command procedure or any other VAX on the NTS VAX network.

1.5 Software Package Summary

Figure 1.4 illustrates the software package applied to the HUNTERS TROPHY event that will be described in the following sections. The activities associated with the GEAR10 computer are enclosed in the dotted lines.

1.5.1 The Login Procedure

When a user logs into GEAR10, the login procedure described in Section 1.4 will first check to see if ICFs in the data base (upper left corner of Figure 1.4) have a later creation date than the ICFs in DD:[TABLES] on GEAR10. ICFs on GEAR10 will be updated automatically if a later issue exists in the data base.

1.5.2 Run RTD Scheduler

Assuming that the user selects "Run RTD Scheduler" to be described in Section 1.6, the Scheduler will look into DD:[TABLES] for ICFs. For each ICF it finds, it will create three command procedures to be submitted at the appropriate time to batch. As illustrated in the upper center of Figure 1.4, one procedure will be created for INITIALIZE, which loads each device on the bus with parameters from the ICF. Another procedure will be created for REALIZE and ANALYZE (directly under INITIALIZE in Figure 1.4). REALIZE will pull data from RTD720 devices creating a BIG file from which ANALYZE generates channel files for plotting by PRELEWD. Finally, the third command procedure for PRELEWD will not be submitted to batch until all the REALIZE jobs (for all ICFs) are completed.

1.5.3 The Initialize Function

The initialize function is described in Section 1.8.1. As illustrated in the upper right corner of Figure 1.4, INITIALIZE communicates with the RTD720 devices on an RMVm0 node through the RTD720 DRIVER in performing the task of loading ICF parameters into each RTD720.

1.5.4 The Realize Function

Section 1.8.2 describes how the REALIZE and ANALYZE programs pull data from RTD720 devices and create plotting files for PRELEWD. As shown in Figure 1.4, REALIZE also communicates with RTD720 devices through the RTD720 DRIVER running on the RMVm0 node.

1.5.5 The Plotting Function

Section 1.8.2 also describes how the PRELEWD program creates plots from CHN type files. This is illustrated in the lower left corner of Figure 1.4. CHN type files may also be used by the digital signal processing (DSP) program that creates ENG type files from which the data can be subjected to a variety of signal processing techniques.

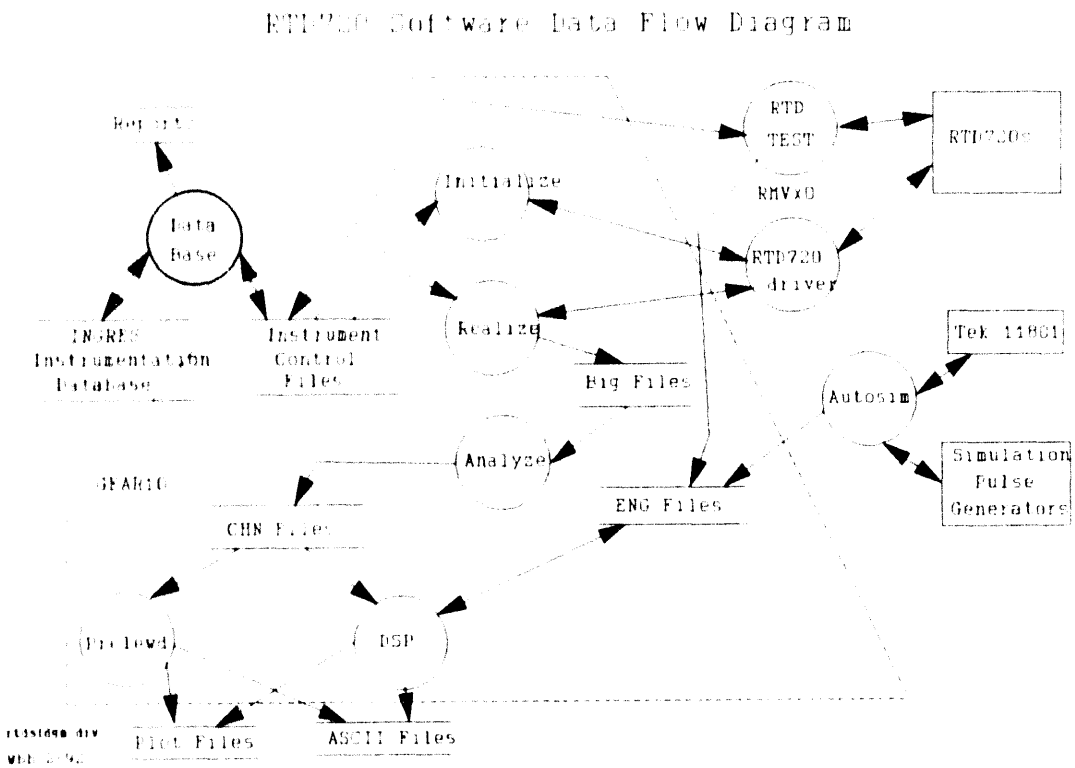


Figure 1-4. RTD 720 Software Data Flow Diagram

1.6 Automated Dry-Run Data-Acquisition Software

The automated dry-run data-acquisition software is designed to acquire, reduce, and display data automatically from all RTD720 channels in rapid sequence. The software includes cooperating processes (discussed below) that run on the GEAR10 and the RMVm0 nodes. The software is designed to provide a scaled data plot for each signal defined by the experimenters. In addition, the software generates data files that can be examined by using DSP software (Lee, 1990).

1.6.1 RTD720 Scheduler

Each of the RMVm0 nodes shown in Figures 1-1 through 1-3 is connected to the GEAR10 computer through a data ethernet. Two IEEE-488 buses are used on each RMVm0, and several RTD720 digitizers are connected to each IEEE-488 bus. DECnet and the VAX VMS operating system allow the ethernet hardware to be accessed essentially in parallel by a large number of processes. The purpose of the RTD720 scheduler software is, thus, to initiate and keep track of an independent automated, data-acquisition process for all RTD720s connected to each IEEE-488 bus.

Because there is one uniquely named RTD720 instrument control file for each IEEE-488 bus, the RTD720 scheduler software simply searches the directory GEAR10::DD:[TABLES] for these files (with names of the form "RMVmnR20.TBL"). As each instrument control file is found, the RTD720 scheduler creates a command procedure that is specific for instrument setup and data recovery from the *nth* IEEE-488 bus on RMVm0. The scheduler first sets the default directory to GEAR10::DD:[SOURCE.RMVm*n*]. The scheduler then invokes the created command procedure, which in turn sequentially invokes INITIALIZE (instrument setup software), REALIZE (data acquisition software), ANALYZE (data analysis software), and PRELEWD (data plotting software) for the corresponding RMVm0 and IEEE-488 bus. The RTD720 scheduler creates and invokes a command procedure for each unique instrument control file it finds in directory DD:[TABLES]. All command procedures run concurrently, and each writes its output to the appropriate directory.

When the RTD720 scheduler detects the final exit from the created command procedures, it then displays completion status and archives the data. Archiving consists of copying the data files and log files from the directories in which they were created on GEAR10 to a similar area on N14VAX. At the end of a day's activities, the program NIGHTMARE will copy the files on N14VAX from their temporary storage area to a permanent storage area. Finally, the Scheduler exits to the GEAR10 CONTROL command procedure, which again displays its menu. At this point, the operator could select "EXIT" or some other option.

1.7 Semi-Automated Data Acquisition

During "Signal" dry runs, the RTD_SCHEDULER command file will normally be called upon to automate the data acquisition and processing functions. When tasks such as laser calibration, cable compensation, and other diagnostic functions necessary in checking out system software and hardware are to be performed, the RTD-DIAGNOSTICS Package is used. This is done by selecting the "Run

RTD-digitizer diagnostics" option from the CONTROL-account menu. These activities are classified as "hand" dry runs and it is optional whether or not the results are sent to archives (N14VAX).

1.8 Setup, Acquisition, and Plotting Programs

As described in the preceding sections, three functions are performed by the software when running either the Scheduler or the Diagnostics. The setup function of installing the ICF parameters into the RTD720 devices is done by the INITIALIZE program. The acquisition function of pulling data from RTD720 devices and creating separate channel files for plotting is done by the REALIZE and ANALYZE programs. The plotting function is done by the PRELEWD program.

1.8.1 Instrument-Setup Program INITIALIZE

INITIALIZE is a program that sequentially initializes all RTD720s on all IEEE-488 buses. Running on GEAR10, INITIALIZE gets instrument setup parameters from instrument control files in the directory GEAR10::DD:[TABLES]. The program then uses cooperating procedures running on the RMVm0s to download the instrument control file settings to each RTD720. A separate invocation of INITIALIZE is started for each IEEE-488 bus so that multiple copies of INITIALIZE run concurrently on GEAR10. This use of VAX/VMS multitasking capability should allow setup for all instruments to proceed rapidly.

An essential function of INITIALIZE will be to leave each RTD720 in an appropriate arm enablement state. According to Tektronix personnel (Hawkey, Undated), the command ARM INTERNAL puts an RTD720 into a state in which it can accept the command ACQUIRE STATE:HLDNXT, which actually arms the RTD720. Once the digitizer is armed, it can then record data upon receipt of an appropriate trigger. Normally, however, we will use external arm and trigger signals. To achieve external arming, the RTD720 must receive the command ARM EXTERNAL. Once this command has been received, the RTD720 will be armed by a TTL external signal at a rear-panel connector. Once the digitizer is armed, it can be triggered by a specified signal trigger applied to the rear-panel external trigger input connector. The exact timing of the arm and trigger signals is the subject of ongoing discussion.

1.8.2 Data-Acquisition Program REALIZE

The main program that collects data into usable files is called REALIZE. Although the RTD720 scheduler runs in the CONTROL account directory on GEAR10, it causes each invocation of REALIZE to run in a directory that is deduced from the control-file name. Thus, each invocation of REALIZE writes its output into this directory. Each REALIZE image knows, from the name of this directory and from parameters and qualifiers on its command line, which instrument control file and output directory on GEAR10 it is to work with. REALIZE first opens its output file (called a BIG file) and prepares to write data to the appropriate disk directory on GEAR10. REALIZE then calls a subprogram that opens communications with the appropriate RMVm0. Another subprogram then awaits and records triggers that will occur on the RTD720s. This subprogram reports to REALIZE

which RTD720s were triggered. REALIZE then loops through the triggered channels, calling a subroutine that gets raw data, one RTD720 at a time. As data are recovered, REALIZE writes the data to the BIG file. Once all triggered devices have been read, REALIZE calls a subroutine that closes communication with the RMVm0. Finally, REALIZE closes the BIG file and sends a completion signal to the RTD720 scheduler.

The modules that are invoked from REALIZE are subroutines that run on GEAR10 as part of the REALIZE executable image. These subroutines in turn communicate using the DECnet protocol (Digital Equipment Corp., 1988) with a data-retrieval executable image that runs on the appropriate RMVm0. The GEAR10 resident module informs the RMVm0 resident module where to find control-file information and where to write the raw data (both on GEAR10). The RMVm0-resident modules are designed to ensure that the RTD720 setups are correct (i.e., in agreement with the control-file values), to poll RTD720s for triggers, to collect raw data from the RTD720s over the IEEE-488 interface, to store the data in GEAR10 arrays specified by REALIZE, and to close communication with the RMVm0. When REALIZE exits, the command procedure created by the RTD720 scheduler then invokes the processing program ANALYZE, which breaks the BIG data file down into a series of experiment files called CHN files. When ANALYZE exits, the command procedure invokes the plotting program PRELEWD on GEAR10. The output of PRELEWD is a set of paper plots and ASCII files (when requested) that can be supplied to experimenters. In addition, the binary channel data files can undergo digital signal processing and trend analysis.

1.8.3 Data Files and Directories

REALIZE creates a BIG file containing the data from one or more RTD data channels and ANALYZE creates a CHN file for each subchannel described in the data base. In addition, each of the processes — INITIALIZE, REALIZE, ANALYZE, and PRELEWD — creates log files that should be reviewed when problems are encountered. These data files along with the log files are stored in directories on the DD: disk for signal dry runs as follows:

Data Bus	Data File Directory
RMV11	DD:[SOURCE.RMV11]
RMV12	DD:[SOURCE.RMV12]
RMV21	DD:[SOURCE.RMV21]
RMV22	DD:[SOURCE.RMV22]
RMV31	DD:[SOURCE.RMV31]
RMV32	DD:[SOURCE.RMV32]

For hand dry runs, files are stored in subdirectories of data file directories based upon run type as follows:

Run Type	Subdirectory
Dry Run	.DIAG
Laser Calibration	.LASER
Cable Compensation	.CABLE

As an example, if a cable compensation hand dry run was performed on bus 1 of RMV10 then the data would be found in DD:[SOURCE.RMV11.CABLE].

1.9 Setup (RTDTEST) Software

RTDTEST is a CONTROL-account option that is intended to allow users to perform setups and hand dry runs on RTD720s. A development version of RTDTEST was written to run on a RMVm0 for the DISTANT ZENITH field test (Caudell, 1991).

The RMVm0s shown in Figures 1-1 through 1-3 are accessible from the NTS VAX network via terminal servers. Any authorized user can log on to the RMVm0 CONTROL account from any terminal connected to the terminal-server network. The functions of the instrument control software RTDTEST allow the user to modify RTD720 setups, modify ICFs, and indicate to the INGRES software to update the data base. RTDTEST allows interactive control to acquire, analyze, and display (plot) dry run data at the terminal. Additional functions of RTDTEST allow the user to send commands to an RTD720, obtain status information, and calibrate the RTD720.
For example, when an authorized user wishes to perform setup type tasks such as making minor changes to the ICFs and monitor the results by plotting data pulled from an RTD720 device with the revised ICF, the CONTROL-account menu option, RTDTEST is the appropriate choice.

1.10 Fielding Results

The most significant result from RTD720 testing during HUNTERS TROPHY fielding was the dramatic reduction in processing time achieved compared to the DIAMOND FORTUNE event. The data volume was approximately five times as great as the DIAMOND FORTUNE data volume, but processing took about one fifth as long. The problem of incompatibilities between the GEAR10 and the ethernet was corrected early in HUNTERS TROPHY fielding, and the results indicated that there was no need to change computer architecture as had previously been proposed (Isidoro, 1992).

Additional capabilities were added to the software to

- Make some changes to the Instrument Control Files in RTDTEST;
- Acquire, process, and plot data from an RMV node independent of GEAR10;

- Permit laser calibration and cable compensation;
- Separate the plotting capability from the data acquisition and processing option in the diagnostic package and the scheduler;
- Halt the scheduler to allow the operator to make corrections in the event all triggers have not been received; and
- Support multiple triggers on the RTD720.

A problem occurred during cable compensation that unfortunately was not detected until after the HUNTERS TROPHY shot on September 18, 1992. The KSC-1 results on buses RMV11 and RMV12 showed proper pulses on all RTD channel 1's but only baselines on all channel 2's. No such discrepancy had ever been observed during signal dry runs. It was later determined that the reason for the discrepancy was that the software that pulls data from RTD devices always pulls both channels of data when there are two channels associated with the device. No problem occurred during signal dry runs or during the shot because we always asked the processing program to process all channels. During cable compensation, however, we were interested in only one of the two channels and the processing program always provided the first half of the data file no matter which channel was asked for. In other words, the BIG file contained channel 1 data when either channel 1 or channel 2 data was requested. The problem has been corrected.

2.0 RTD VAX NETWORK

2.1 Introduction

Instrumentation Development Department 9321 uses DEC VAX computers to support full-scale nuclear testing at NTS.

The SNL NTS VAX Network provides for automating activities ranging from the setup of data collection instrumentation to the final playback and analysis of the test data. The computer systems are used in interactive and batch modes ranging from single-program execution to intercomputer, multiple-task communications. A complete description of the SNL VAX Network can be found in a Sandia Report, number SAND91-2916, printed in March 1992.

The RTD VAX Network was designed to use DEC VLCs and a DEC VAX 4000. For HUNTERS TROPHY, DEC microVAX-IIs were used. The microVAX-IIs use an IEQ11 Q bus-to-GPIB bus converter, which differs from the DEC VLCs that use an IEZ11 SCSI-to-GPIB bus converter. The DEC machines can be used interchangeably with only minor code changes to reflect differences in calling the GPIB driver.

The selection process for the hardware considered the eventual system that would be used. For the HUNTERS TROPHY event, DEC microVAX-IIs were used as instrument controller nodes, instead of the VLCs discussed in Section 2.2, describing the selection process.

Talaris printers were selected for these systems. The code is written to use QUIC commands from the Talaris library. If Talaris 800 or 1200 printers are used, the library module QC is required as an addition. If Talaris 1590 printers are used, the QC module for the library is delivered with the software. For additional speed, it is recommended that a QUIC hardware chip be installed in the Talaris 1590.

2.2 Selection Process

When selecting the equipment, several things were considered, including cost, ease of code portability between SNL and DNA, interconnections to the SNL VAX network, data portability between the agencies, and maintenance cost. The VAX architecture, specifically the 4000 line of computers, satisfied these considerations. The VMS operating system with its virtual memory capability and other features, such as enhanced FORTRAN, DECnet, and system utilities, greatly reduced software development cost. The VAX VMS architecture allowed for portions of the software in current use to port to the new computers with minimal modifications.

In selecting the network hardware, we decided to continue the thick-wire ethernet application that is used throughout the SNL VAX Network. This backbone is compatible with multivendor,

multiprotocol networking. Thick-wire ethernet is also more durable and is less susceptible to network errors than thin-wire applications.

The software and hardware were chosen to reduce the time required to perform a dry run or actual test playback and to ease the interface for the operators and engineers.

2.3 Hardware

The VAX 4000-300 was chosen for its Q-bus capabilities, DSSI architecture ability, and low cost. The VAX 4000 requires no special power or environmental needs. The system configuration and specifications are listed below:

- DV-43LT1-B9, VAX 4000-300 rackmount system
- 64 Mbytes memory
- 2-DSSI adapters
- 1-RF35E-AA, 852-Mbyte ISE disk
- 1-TF85E-JA, 2.6-Gbyte cartridge tape system

The four VAX VLC workstations supplied a low-cost solution for the GPIB interface to the RTD720s. The specifications are:

- PV31A-AA, 4000 VLC VAXstation
- 8 Mbytes memory
- MS40-BA, 8 Mbytes additional memory
- VCR16, 16 inch color 1024 x 726 monitor
- SZ03B-CA RZ25 426 Mbyte tabletop SCSI disk drive

The IEZ11 is an SCSI-to-GPIB bus converter. It allows for up to 14 IEEE-488 bus instrument connections. The device connects to the SCSI port that also contains the SCSI disk. The maximum number of connections to an SCSI bus is seven. The VAX 4000-300 computer and the RF35 disk take two of these connections. The transfer rate of the IEZ11 is up to 500 kbytes per second.

2.4 Software

The operating system for the five computers is VAX VMS 5.5. Additional software packages, including the Graphical Kernel System (GKS), the Forms Management System (FMS), IEX drivers, and INGRES, are required before the application programs will run.

The GKS is a package supplied by Advanced Technology Center (ATC). The package is licensed in two manners: The VAX 4000 has a development license and the VAX VLCs have an execution license.

The address and phone number for ATC is:

Advanced Technology Center
22982 Mill Creek Drive
Laguna Hills, CA 92653

Phone (714) 583-9119
Fax (714) 583-9213

The FMS software is a DEC product and consists of a single file to allow for execution of application code. A license would be required to allow for development and recompilation of code that includes any FMS commands.

The IEX driver is a DEC product and requires a license. The VLCs are licensed for the product. This package provides the interface to the IEZ11 interface.

INGRES is the data base package that is used to create the instrument control files. This package is licensed to the VAX 4000-300. for HUNTERS TROPHY, INGRES was run on the SNL node N12DBM. INGRES is purchased through a third-party vendor. The address and phone number for the INGRES producer is:

ASK Computer Systems, Inc.
INGRES Products Division
1080 Marina Village Parkway
Alameda, CA 94501-1041
Phone (415) 769-1400

To provide development capabilities, additional licenses for FORTRAN and FMS are necessary. It is recommended that these licenses be purchased for the VAX 4000-300 if development is desired.

2.5 System Parameters

To allow for better system performance and the successful execution of some applications, it is necessary that the following system parameters be set:

```
min_SYSMWCNT    = 1287      ! from 1257 for GKS
min_GBLPAGES    = 114415    ! from 10700 for GKS
min_LRPCOUNT    = 40        ! from 8
min_LRPCOUNTV   = 100       ! from 60
min_MAXBUF      = 33000     ! for users accounts
min_CHANNELCNT  = 200       ! from 127 for INGRES
```

The LRPCOUNTS were increased to help the DECnet traffic. The MAXBUF parameter was increased because of the application codes.

2.6 Networking

The network consists of a VAX 4000-300 and four VAX VLC workstations. A diagram of the network is included in Figure 2-1. Thick-net ethernet was chosen as the transmission media for durability. The VLC workstations have a thick-net 15-pin AUI connector. The VAX 4000-300 has a switchable port for either thin or thick net. Thick net was selected on this machine for compatibility. The VAX 4000-300 can also be equipped with dual ethernet ports by utilizing a DESQA. Dual ethernet ports would separate the RTD720 data traffic from the communications to a production network. To increase the performance for networking, it is recommended that the following parameters be increased in NCP by issuing the following commands:

- $ MCR NCP
- define exec incoming timer 60
- define exec outgoing timer 90
- define exec max links 35
- define exec pipeline quota 12000
- set line ISA-0 state off
- define line ISA-0 receive buffers 30
- set line ISA-0 state on
- $

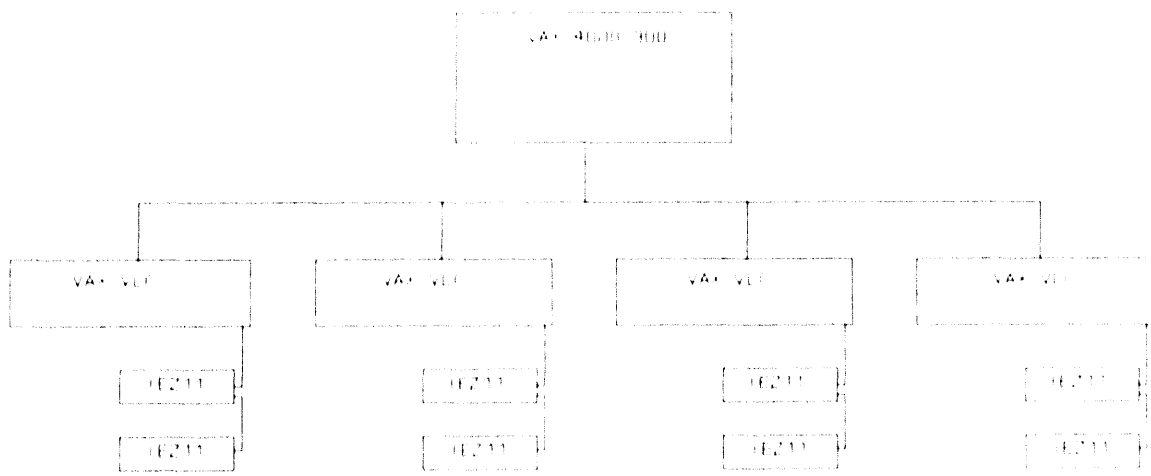


Figure 2-1. RTD Network

2.7 Security Concerns

Security was considered in the original architecture design for the software and hardware. The Software Project Leader (SPL) and the Network Manager agreed on the design to allow for the separation of the agency's data and operational separation in accessing the instrument buses. The network, as shown in Figure 2-1, allows for bus isolation with user authentication on the VLCs to prevent any agency from accessing the IEEE-480 buses of other agencies. The standard group protections and Access Control List (ACL) features of VMS provide for the remainder of the data isolation.

The SZ03B-CA RZ25 426-Mbyte tabletop SCSI disk drive purchased for the VAX VLCs was chosen for ease of removal if the systems were required to change classification modes. Simply disconnecting the power cord and the SCSI interface would allow for the removal of the drive for sanitization procedures.

The RF35E-AA 852-Mbyte disk used in the VAX 4000-300 can be removed with one thumbscrew located on the front of the drive. By selecting these disks, Winchester technology could be applied while considering the problems associated with classification issues.

The computer network was operated in a classified (SRD) mode for the HUNTERS TROPHY event and for approximately two months afterward. The conversion to and from classified operation was performed in accordance with the DOE-approved network security plan.

3.0 OVERVIEW OF RTD720 SCHEDULER AND DIAGNOSTICS

3.1 Introduction

The automated dry-run data-acquisition software is designed to acquire, reduce, and display data automatically from all RTD720 channels in rapid sequence. The software includes cooperating processes (discussed below) that run on the VAX 4000 and the microVAX II computers. The software is designed to provide a scaled data plot for each signal defined by the experimenters.

In addition, the software generates data files than can be subjected to examination using DSP software.

3.2 RTD720 Scheduler

The purpose of the RTD720 scheduler software is to initiate and keep track of an independent automated data-acquisition process for all RTD720s connected to each IEEE-488 bus.

Because there is one uniquely named RTD720 instrument control file for each IEEE-488 bus, the RTD720 scheduler software simply searches the directory GEAR10::DD:[TABLES] for these files (with names of the form RMVm*n*R20.TBL). As each such instrument control file is found, the RTD720 scheduler creates a command procedure that is specific for instrument setup and data recovery from the *nth* IEEE-488 bus on microVAX-II RMVm0. The scheduler first sets the default directory to GEAR10::DD:[SOURCE.RMVm*n*]. The scheduler then invokes the created command procedure, which in turn sequentially invokes INITIALIZE (instrument setup software), REALIZE (data acquisition software), ANALYZE (data analysis software), and PRELEWD (data plotting software) for the corresponding microVAX II and IEEE-488 bus. The RTD720 scheduler creates and invokes a command procedure for each unique instrument control file it finds in directory DD:[TABLES]. All command procedures run concurrently, and each writes its output to the appropriate directory.

Each task created provides the scheduler with status messages which are displayed at the control terminal. This is done through message files created by the scheduler. A separate message file is created for each job that is submitted. Each file is displayed on a separate portion of the screen. Each submitted job represents the RTD720 devices associated with an ICF. The programs INITIALIZE, REALIZE, ANALYZE, and PRELEWD write messages to this message file, each with its own "COMM_SYM" so the sender can be identified. The scheduler scans through these messages as the jobs are running, pausing after each loop. It retrieves the latest message in each file and displays it at the appropriate screen location. Table 3-1 describes the messages that may be displayed in each program.

Table 3-1. Status Messages, Program of Origin, and Descriptions of Messages.

COMM_SYM MESSAGE	Program	Description
STARTING	INIT	The command line is being parsed and initialize has started.
SETTING UP DEVICE n	INIT	Attempting to initialize device n.
ERROR EXIT	INIT	A fatal error has been found and an exit has occurred. See INITIALIZE log file for details.
NORMAL EXIT	INIT	Terminating normally.
STARTING	REAL	The command line is being parsed and REALIZE has started.
OPENING LINK	REAL	Attempting to open task-to-task link.
LINK OPENED SUCCESSFULLY	REAL	Task-to-task link opened successfully.
STARTING ACQUISITION	REAL	Starting data acquisition across link.
DOING DEVICE n	REAL	Starting data acquisition for device n.
ACQUISITION TERMINATED	REAL	Ending data acquisition.
ERROR EXIT	REAL	A fatal error has been found and an exit has occurred. See REALIZE log file for details.
LINK CLOSED	REAL	The task-to-task link is closed.
RTD DATA AVAILABLE	REAL	Device n data has been transferred.

Table 3-1. Status Messages, Program of Origin, and Descriptions of Messages (Concluded).

COMM_SYM MESSAGE	Program	Description
STARTING ANALYSIS	ANLZ	Program ANALYZE begins.
NORMAL EXIT	ANLZ	End of program ANALYZE.
ERROR EXIT	ANLZ	The error exit from ANALYZE is displayed whenever bad status is encountered from: 1. BIG_DIR which obtains a list of channels in the BIG file. OR 2. READ_BIG_HDR which reads the header from the BIG file for a channel OR 3. READ_BIG which reads a buffer of data from the BIG file. Examine the ANALYZE log file to determine which event caused the error.
PRELEWD STARTING	PREL	The command line is being parsed and PRELEWD has started.
PRELEWD NORMAL EXIT	PREL	Normal termination.
PRELEWD:EXIT ERROR	PREL	A fatal error has occurred. See PRELEWD log for further information.
"plot file name"	PREL	Attempting to generate a plot for the named file.

When the RTD720 scheduler detects the final exit from the created command procedures, it then displays completion status and performs any necessary data archival functions. Finally, it exits to the GEAR10 CONTROL command procedure, which displays its menu. At this point, the operator could select "EXIT" or some other option.

The RTD_SCHEDULER creates a log file in DD:[SOURCE] identifying the directories from which the SCHEDULER is deleting files, and the plotting node and device selected by the operator. The log file is named SCHEDULER.LOG.

3.3 Semi-Automated Data Acquisition

During "signal" dry runs, the RTD_SCHEDULER command file will normally be called upon to automate the data acquisition and processing functions. When tasks such as laser calibration, cable compensation, and other diagnostic functions necessary in checking out system software and hardware are to be performed, the RTD DIAGNOSTICS package is used. These activities are classified as "hand" dry runs and it is optional whether or not the results are sent to archives (N14VAX). The menu for the diagnostics package is as follows:

(1) Initialize one or more RTD720s
(2) Obtain and Analyze RTD720 data from one or more RTD720s
(3) Plot RTD720 data from one or more RTD720s
(4) Run Development Scheduler
(E) Exit

For options 1, 2, and 3, a debug option is available to the software designers. This option permits the debugging of programs while running from the COM file. It helps to isolate errors when determining the location of the bug, in the COM file or in the program. It also provides an option to skip programs when only the interface of the COM file and a specific program is necessary.

On-line help is available for options 1, 2, and 3. To obtain help, the user enters a question mark (?) in response to a question for which help is needed. An explanation is then provided followed by a repeat of the question.

3.3.1 Option 1 - Initialize

When the initialize option is selected, the diagnostics command procedure (RUN_RTD_DIAGS.COM) invokes the command procedure RUN_INITIALIZE_RTD720.COM. After confirming that the user is running from the GEAR10 node, the program searches through the ICFs in the DD:[TABLES] directory and presents the user with a list of files. Each file represents a bus containing one or more RTD720 devices. The user is asked to respond to the following items:

1. Select one of the files (or buses) for initializing devices. The user may also elect to initialize all devices on all buses (ICFs) by entering a carriage return only (no number).

2. If the user has not chosen to initialize all devices on all buses, the next question will be to determine which RTD720 on the bus is to be initialized. If the user simply enters a carriage return (no number), all devices on the bus will be initialized.

3. Next, the user will be asked to select a data source. The options are:

   a. 0 ==> normal or diagnostics (default)
   b. 1 ==> laser calibration
   c. 2 ==> cable compensation

4. What type of dry run is it? The options are hand (H) or signal (S). If the hand dry run (H) is selected, all files created will be stored in a subdirectory of the applicable default directory as follows:

   normal or diagnostics ==> default_dir.diag
   laser calibration ==> default_dir.laser
   cable compensation ==> default_dir.cable

5. If the user has requested cable compensation and has also selected a single device to initialize, the program will then request a channel number on the device to be initialized.

6. If the user has selected signal (S) for run type, all files will be deleted from the applicable directory before proceeding.

7. Before attempting to initialize any devices, this procedure will display the options that the user has selected and ask for the operator's approval before continuing. The operator can review his or her selections at this point and reenter parameters if desired.

8. A log file will be created in the applicable subdirectory of DD:[SOURCE.RMVXX] containing the parameters entered by the operator. The file is named DIAG_INITIALIZE.LOG

3.3.2 Option 2 - Realize/Analyze

When the realize/analyze option is selected, the diagnostics command procedure RUN_RTD_DIAGS.COM invokes the command procedure RUN_REALIZE_RTD720.COM. As in the initialize option, the realize command procedure starts out by confirming that the user is running from the GEAR10 node. The program then searches through the ICFs in the DD:[TABLES] directory and presents the user with a list of files. Each file represents a bus containing one or more RTD720 devices. The user is requested to respond to the following items:

1. Select one of the files (or buses) for data acquisition and analysis. In the realize/analyze option, however, the user cannot elect to acquire and analyze data on all buses. This can be done only by the RTD_SCHEDULER.

2. Next, select a data source. The options are:

   a. 0 ==> normal or diagnostics (default)
   b. 1 ==> laser calibration
   c. 2 ==> cable compensation

3. What type of dry run is it? The options are hand (H) or signal (S). If the hand dry run (H) is selected, all files created will be stored in a subdirectory of the applicable default directory as follows:

   normal or diagnostics ==> default_dir.diag
   laser calibration ==> default_dir.laser
   cable compensation ==> default_dir.cable

4. If the run type is signal (S), all files created prior to the current date in the default directory will be deleted before new data is acquired. If the run type is hand (H), the user will be asked if existing files in the directory should be cleaned up.

5. The user is then asked: "Do you want to REALIZE new data?" At this point, the user may choose to skip REALIZE (data acquisition), and ANALYZE previously acquired data.

6. If the user has chosen to REALIZE new data, the next question is "From what RTDs do you want data? [ALL]" If the user wishes to retrieve data from all RTDs on the bus, the response should be a carriage return, otherwise the program expects a numerical entry of the desired RTD device. If the user has requested cable compensation and has also selected a single device to collect data, the program will then request

   a. A channel number on the device to collect data,

   b. The applicable subchannel number, and

   c. A character string tag (up to 8 characters) identifying the applicable pulse width.

   If the user had requested laser calibration and a single device, the program will ask

   a. From which channels do you want data?

   b. From what subchannels do you want data?

7. The next question is: "Do you want REALIZE to look for triggers?" If the devices have not yet been triggered, the user would probably want the program to look for a trigger before attempting to acquire data. On the other hand, the operator may know that the device(s) have already been triggered and he or she simply wants to extract stored data.

8. Before attempting to get data, this procedure will display the options that the user has selected and ask for the operator's approval before continuing. The operator can review the selections at this point and reenter parameters if desired.

9. A log file will be created in the applicable subdirectory of DD:[SOURCE.RMVXX] containing the parameters entered by the operator. The file is named DIAG_REALIZE.LOG.

10. After a BIG file has been acquired by REALIZE, the user is given a choice of whether or not to create channel plotting files using the ANALYZE program.

11. Finally, if this has been a signal dry run, all files created during the current run will be transferred to archives on the N14VAX node. If this has been a hand dry run, the user will be given a choice of whether or not to archive files.

3.3.3 Option 3 - Plotting

The plotting option was added to the list of diagnostics during HUNTERS TROPHY. Plotting had formerly been part of the data acquisition and processing package (RUN_REALIZE_RTD720.COM), but it was determined that it was not always necessary to create plots when acquiring data, and plotting was generally a very time-consuming process.

When the plotting option is selected, the diagnostics command procedure RUN_RTD_DIAGS.COM invokes the command procedure RUN_PRELEWD_RTD720.COM. As in the other options, the plotting command procedure starts out by confirming that the user is running from the GEAR10 node. The program searches through the ICFs in the DD:[TABLES] directory and presents the user with a list of files. Each file represents a bus containing one or more RTD720 devices. The user is requested to respond to the following items:

1. Select one of the files (or buses) for plotting. Plotting will be limited to those channel files in the directory associated with the RTD720 bus selected.

2. Select a data source. The options are:

   a. 0 ==> normal or diagnostics (default)
   b. 1 ==> laser calibration
   c. 2 ==> cable compensation

The data source is necessary in the plotting option to locate the applicable directory for plotting.

3. What type of dry run is it? The options are hand (H) or signal (S). If the hand dry run (H) is selected, all files stored in the applicable subdirectory will be plotted as follows:

   normal or diagnostics ==> default_dir.diag
   laser calibration ==> default_dir.laser
   cable compensation ==> default_dir.cable

4. Select a plotting node (the default is N12VAX) and a plotting queue for the plotting program, PRELEWD, to send plotting files.

5. The user has an option in the number of files plotted. The user may either select a single device (RTD) or plot all files in the directory.

After all plots have been submitted to the selected node and queue, all PRELEWD log files will be copied to the default directory in the archive node.

3.3.4 Option 4 - Development Scheduler

This option would not normally be used by an operator. It was placed here to take advantage of the environment afforded by the Control Account in comparing the performance of several different computers in a standard task of creating plot files from all the channel files in all the directories associated with ICFs DD:[TABLES]. The purpose of these performance comparisons was to determine if an alternate architecture, in which the programs INITIALIZE, REALIZE, ANALYZE, and PRELEWD would be run at the RMV level on VAX 4000 VLC workstations (replacing the present microVAXs), would significantly reduce processing time.

Because it was believed that PRELEWD (the plotting program) was largely responsible for the amount of time required to process RTD720 data in a previous event (DIAMOND FORTUNE), the performance of each architecture was measured by determining time required to process a set of 36 channel files contained in 6 different directories each corresponding to a different RTD data bus:

1. The GEAR10 VAX 4000 computer in the current architecture submits all 6 jobs to run in parallel; therefore, its performance was measured by the time it took to process the entire set of 36 files (6 per directory) when all jobs were submitted in parallel.

2. The VAX workstation performance was measured by determining the time required to process 2 jobs running in parallel, because this is the way it would be done in the proposed new architecture.

The results of this study showed that the present architecture was superior to the proposed architecture because the time required (about 3 minutes) was less than the 4 1/2 minutes required by the VAX workstation in the new architecture. The poor performance of the system during DIAMOND FORTUNE was attributed to incompatibilities between the GEAR10 computer and the ethernet that were corrected for the HUNTERS TROPHY event.

4.0 REALIZE AND INITIALIZE USER INFORMATION

4.1 Introduction

REALIZE and INITIALIZE are two programs in the NTS Instrumentation System suite of codes. They are very similar in construction, and both call the RTD720 communications subprograms described in Volume 2, Section 3. INITIALIZE reads the ICF and sends the setup information to the communications subprograms, which in turn transfer the information to the slave node to be sent on the GPIB to the RTD720. REALIZE reads the ICF and, if required, checks for triggers. It then requests data from the communications subprograms for each RTD, which it formats into a BIG file. For information on the BIG file see Volume 2, Appendix C. Because INITIALIZE and REALIZE are similar, they will be described together. Unless a specific program is indicated, all information pertains to both.

4.2 Definitions

The following defines some useful terms:

(1) directory_name refers to the NODE::DISK:[DIRECTORY] description that is either an implied or stated part of every file name.

(2) file_spec refers to the entire file description NODE::DISK:[DIRECTORY]FILE_NAME.EXTENSION;VERSION, or as much of it as is required.

(3) {...} indicates that what is enclosed in { } is optional.

4.3 Running REALIZE and INITIALIZE

Source code and executables for both programs exist on all major nodes (GEAR10, ABQVAX, and N12VAX), and executables exist on the RMV nodes.

These programs are usually run under the captive account CONTROL, which has the necessary privileges. CONTROL defines all necessary commands and then initiates execution with a command line formed from the user's answers to questions.

A user not running as CONTROL must define the VMS-like commands, INITIALIZE or REALIZE, using the following commands:

$ DEFINE REALIZE$LIBRARY (directory_name containing .CLD files)

$ SET COMMAND REALIZE$LIBRARY:INITIALIZE_COMMAND_DEF.CLD

$ SET COMMAND REALIZE$LIBRARY:REALIZE_COMMAND_DEF.CLD

These definitions should be in LOGIN.COM on all nodes intended for REALIZE or INITIALIZE. They make use of the DEC Utility Command Definition described in the VMS Programming Manual Volume 2B. REALIZE_COMMAND_DEF.CLD is the definitive authority on REALIZE's parameters and qualifiers, while INITIALIZE_COMMAND_DEF.CLD is the definitive authority on INITIALIZE's parameters and qualifiers. The CLD files are ASCII.

4.4 Command Line Examples

The following are examples of command lines that might arise in actual usage. Normally a user will be running as CONTROL from a command file, although both programs can be executed directly. These examples should cover most user situations. The specific meaning of each qualifier in INITIALIZE and REALIZE is covered in Sections 4.5 and 4.6, respectively.

(1) $ INITIALIZE/DEVICE=4/TABLE=DD:[TABLES]RMV11R20.TBL

This command line requests that the RTD with device address 4 on port 1 of node RMV10, be initialized with information from the ICF DD:[TABLES]RMV11R20.TBL.

(2) $ REALIZE/TABLE=DD:[TABLES]RMV21R20.TBL

This line requests recovery of data from all RTDs available from port 1 of node RMV20, with the BIG file created in the current directory, and no wait for triggers.

(3) $ REALIZE/DEVICE=1/TABLE=DD:[TABLES]RMV12R20.TBL/ -
SEND=ABC::DD:[RTDSTUFF]/USE=1

This command requests recovery of data from that RTD with device address 1 on port 2 of node RMV10. The BIG file created in the current directory, and sent to the DD:[RDTSTUFF] directory on node ABC after being closed. The data are to be recovered with no wait for triggers. The data recovered is for laser cals, and the laser cal subchannel description (/USE=1) is put in the BIG file. The user must have write permission on that node in ABC::DD:[RTDSTUFF].

(4) $ REALIZE/DEVICE=2/CHANNEL=2 -
/TABLE=ABC::DD:[TABLES]RMV32R20.TBL -
/SEND=USER:[HOME]/TRIGGER

This command line requests data only from channel 2 from the RTD with device address 2 on port 2 of node RMV30, with the BIG file created in the directory USER:[HOME] on the current node. REALIZE will not recover data until after a trigger (/TRIGGER) has been received.

4.5 Command Qualifiers for INITIALIZE

The verb defined by INITIALIZE_COMMAND_DEF.CLD has one required and several optional qualifiers.

QUALIFIER	DEFAULT
Required Qualifier:	
(1) /TABLE=filespec.	NA
Qualifiers that control REALIZE logic:	
(2) /DEVICE=n	all devices
(3) /USE=n, (n=0,1,or 2)	/USE=0
Qualifiers not intended for the casual user:	
(4) /DEBUG	not present
(5) /COMM_SYM=*string*	not present

Taking these in order:

(1) /TABLE=file_spec

The /TABLE=file_spec qualifier MUST be supplied. The filespec is the ICF to be used. If it is not supplied, or if the ICF specified does not exist, execution is terminated.

(2) /DEVICE=n

The *n* is a GPIB device address on a specific computer port. /DEVICE without a value is not allowed.

(3) /USE=n (n=0,1, or 2)
/USE=0 (default)

USE directs which set of structures is to appear in the BIG file.
USE=0 requests normal event data structures,
USE=1 indicates that laser cal data structures are requested, and
USE=2 indicates that cable compensation structures are requested.

The ICF contains information for each of the above conditions. INITIALIZE must be told which set to use.

(4) /DEBUG

If used, this qualifier MUST follow the verb INITIALIZE. It causes an alternate image to be invoked which has been linked "/DEBUG" with DEC's debugger utility. DEBUG may not be negated, just omitted. Generally, it is intended for software developers' use only.

(5) /COMM_SYM=*string*

This special qualifier is used ONLY when INITIALIZE is invoked while using the CONTROL account and RTD_SCHEDULER.COM. The casual INITIALIZE user should not supply this qualifier. *string*.DAT is the name of a file into which messages are written, and from which the RTD_SCHEDULER.COM reads those messages for display on the screen.

4.6 Command Qualifiers for REALIZE

The verb defined by REALIZE_COMMAND_DEF.CLD has one required and several optional qualifiers.

QUALIFIER	DEFAULT
Required Qualifier:	
(1) /TABLE=filespec,	NA
Qualifier dealing with BIG file creation or disposition:	
(2) /SEND[=directory_name],	/NOSEND
Qualifiers that control REALIZE logic:	
(3) /CHANNEL=n	all ICF channels

QUALIFIER	DEFAULT
(4) /DEVICE=n	all ICF devices
(5) /USE=n, (n=0,1, or 2)	/USE=0
(6) /TRIGGER	/NOTRIGGER

Qualifiers not intended for the casual user:

(7) /DEBUG	not present
(8) /COMM_SYM=*string*	not present

Taking these in order:

(1) /TABLE=file_spec

The /TABLE=file_spec qualifier MUST be supplied. The filespec is the ICF to be used. If it is not supplied, or if the ICF specified does not exist, execution is terminated.

(2) /SEND(=directory_name)
/NOSEND (default)

The /SEND qualifier tells REALIZE what to do with BIG files. BIG files are created in the CURRENT DIRECTORY. If /SEND specifies a directory_name, then BIG files are copied to that directory_name. This is done by using LIB$SPAWN from within REALIZE. If /SEND does not specify a directory_name, it has the same action as /NOSEND, and the BIG files are created in the current directory and not moved.

WARNING

Using the CONTROL account gives the user all necessary permissions and privileges. If you are NOT running under CONTROL, and if BIG files are to be sent to another directory_name, YOU MUST HAVE WRITE PERMISSION in that directory_name. You must always have write permission in the current directory.

(3) /CHANNEL=n (n=1,2,3, or 4)

RTD720 devices have either 1, 2, or 4 channels. If you only want the data from a specific channel, it is named here. /CHANNEL without a value is not allowed.

(4) /DEVICE=n

The device specification is a device address on a specific computer port. /DEVICE without a value is not allowed.

(5) /USE=n (n=0,1, or 2)
/USE=0 (default)

USE directs which set of structures is to appear in the BIG file.
USE=0 request normal event data structures,
USE=1 indicates that laser cal data structures are requested, and
USE=2 indicates that cable compensation structures are requested.

The ICF contains information for each of the above conditions. REALIZE must be told which set to use and send in the BIG file.

(6) /TRIGGER
/NOTRIGGER (default)

The /TRIGGER qualifier indicates that REALIZE asks the function interface to look for trigger signals. The negated values tell it not to expect triggers, just get the data.

(7) /DEBUG

If used, this qualifier MUST follow REALIZE. It causes an alternate image to be invoked, one that was linked /DEBUG with DEC's debugger utility. DEBUG may not be negated, just omitted. It is generally intended for software developers' use only.

(8) /COMM_SYM=*string*

This special qualifier is used ONLY when REALIZE is invoked while using the CONTROL account and RTD_SCHEDULER.COM. The casual REALIZE user should not supply this qualifier. *string*.DAT is the name of a file into which messages are written, and from which the RTD_SCHEDULER.COM reads those messages for display on the screen.

4.7 Sequence of Operations

A simplified overview of the operations of REALIZE and INITIALIZE follows:

(1) Check the computing environment. Get the USERNAME, the current node, the directory, and the date and time. Open the LOG file and write the name of the execution file and its creation date.

(2) Parse the command line after writing it to the LOG file. If an ICF is not specified, then terminate with an appropriate message in the LOG file. Set the defaults into all parameters not specified on the command line.

(3) Open the ICF and read all non-RTD-specific records. Start filling the record that will be sent to the communications subprograms.

(4) Find all devices available in the ICF and for each device find all possible channels. Make an intersection of these lists with device/channel requests from the command line.

(5) Read the RTD-specific records. Close the ICF.

(6) Open the node-to-node link.

(7) For REALIZE only, open the BIG file.

(8) For REALIZE only, and only if requested, determine the trigger status of all desired modules via the communications subprograms.

(9) If INITIALIZE, then, for each device, request the device to be set up by passing the ICF data for that device to the communications subprograms.

If REALIZE, then, for each device, request its data and write it, by channel, to the BIG file.

(10) When finished with all devices, close the node-to-node link.

(11) Close all open files and exit.

4.8 General Comments

Each program creates a .LOG file. This file contains relevant information, such as the command line, the name and creation date of the executable, and all unusual happenings noted by the software. This file will be created in the current directory. The .LOG file should be the first place to look if you suspect an error. It is formatted to allow its use with SLOG (Summary LOG, Appendix G). Errors that occur in the communications subprograms described in Volume 2, Section 3 are documented in their .LOG file but are noted in the appropriate REALIZE or INITIALIZE LOG file.

REALIZE creates a BIG file that is passed on to ANALYZE and contains records from the ICF and data recovered from the RTD. This file is organized by channel for all devices requested.

5.0 ANALYZE USERS MANUAL

5.1 Introduction

During the Diamond Fortune test, the NTS Instrumentation Development Department 9321 developed a computer architecture to acquire and process RTD720 high-speed digital data. Within this architecture, the REALIZE program acquires data from RTD720 devices and stores it together with the appropriate header information in a file called BIG. The ANALYZE program then separates the contents of BIG into individual channel (or subchannel) files with the appropriate header information. These channel (CHN) files can then be plotted using the PRELEWD program.

5.2 Input Parameters

Most input parameters to ANALYZE are obtained from the channel information provided in the CHNBIG header. The user must supply the name of the BIG file on the command line. There are two parameters that may be entered as an argument on the command line:

> /CHANNEL=n or /CHANNEL="n1,n2,n3,etc." or /CHANNEL="n1-nn" to select one or more specific channels in a group to be processed in this run. Note: CHANNEL may be shortened to CH.
>
> /SUBCHAN=n or /SUBCHAN="n1,n2,n3,etc." or /SUBCHAN="n1-nn" to select one or more specific subchannels on one RTD channel processed in this run. Note: SUBCHAN may be shortened to SUB.

5.3 Input File

ANALYZE requires an RTDBIG file as input.

5.4 Output Files

If the input from the BIG file is a single channel without subchannels, then there is one output file name CHN*snnrmmm*.DAT. If the input is a channel with two or more subchannels, then there is a CHN*snnrmmm-kk*.DAT file for the combined raw data and for each subchannel. The combined raw data will be identified as subchannel "00."

In the examples above:

snn = source code
r = recorder code
mmm = channel number
kk = subchannel number

ANALYZE also creates a file called ANALYZE.LOG that contains a description of the discrepancies observed during the run on each of the data channels contained in the BIG file.

5.5. Examples

An example of the simplest case for running ANALYZE:

```
$ ANALYZE RTDBIG1011.DAT
```

ANALYZE will obtain the source and recorder codes from the BIG file name.

The next three commands are examples of channel selection from within the total group in a BIG file. The first example is for one channel, the second for several selected channels, and the last for a range of channels.

```
$ ANALYZE/CHANNEL=2 RTDBIG1011.DAT
$ ANALYZE/CHANNEL="3,7,12,17" RTDBIG1011.DAT
$ ANALYZE/CHANNEL="3-14" RTDBIG1011.DAT
```

The following command shows how subchannels may be selected from one specified channel.

```
$ ANALYZE/CHANNEL=2/SUB=1,3 RTDBIGG1011.DAT
```

5.6 Sequence of Operation

The ANALYZE program performs its function of creating channel and subchannel files from the BIG file created by the REALIZE program in the following sequence:

1. Initialize program variables.
2. Open a log file ANALYZE.LOG for informative messages and diagnostics.
3. Parse the command line for BIG file name and other processing parameters.
4. Extract source and collection point parameters from the BIG file name to use in creating CHN file names
5. Open the BIG file using the OPEN_BIG utility.
6. Write header information into the log file.
7. Get a directory of data channels in the BIG file using the BIG_DIR utility.

8. For each data channel in the directory, perform the following sequence of operations (unless specific channels have been selected on the command line):

   a. Get the header information for the channel using the READ_BIG_HEADER utility.

   b. Read in a buffer of information from the BIG file for the channel using the READ_BIG utility.

   c. Call the PROC_RTD720 subroutine that creates the desired channel and sub-channel files based upon data base information contained in the header.

9. Notify the RTD Scheduler that ANALYZE has completed its processing of the BIG file.

10. Close the BIG file using the CLOSE_BIG utility.

5.7 PROC_RTD720 Sequence of Operation

When the PROC_RTD720 subroutine is called, it performs the following sequence of operations for each prescribed data channel:

1. Convert the input data from a byte array to a 16-bit integer array.

2. Obtain a list of applicable subchannels from data base information in the header.

3. Compare the number of subchannel names so obtained with the number of subchannels specified in the data base.

   a. If these numbers do not agree, only create the "00" subchannel, which contains all the data for the channel.

4. For each subchannel defined in the data base, the following steps are performed to create files based upon the boundaries defined by EXPMT_LEFT and EXPMT_RIGHT parameters in the data base.

   a. Open the designated subchannel file.

   b. Write the applicable portion of the channel information from the BIG file into the subchannel file.

c. Write the appropriate header information at the beginning of the subchannel file.

d. Close the subchannel file.

5.8 Diagnostics

ANALYZE will generate a file called ANALYZE.LOG. For each data channel within the BIG file, ANALYZE will display the messages:

(1) PROCESSING RTD CHANNEL NO. xxx
(2) PROCESSING COMPLETE ON CHANNEL xxx NO SAMPLES PROCESSED: xxxxx

Between these two messages discrepancies may be displayed that are applicable to the data channel encountered during processing.

The following is a listing of all diagnostics and a description of the meaning:

DIAGNOSTIC	DISCUSSION
1. NO DATA FOR THIS CHANNEL	A byte count of zero was obtained when an attempt was made to read from the RTDBIG file for this memory channel.
2. COULD NOT FIND ALL RTD720 SUBCHANNELS	The number of subchannels in one section of the data base does not agree with the actual subchannel numbers from another part of the data base. The ANALYZE program, therefore, cannot properly identify output CHN files. The data base administrator should be notified immediately if this discrepancy occurs.
3. SELECTED SUBCHANNEL NO. XX DOES NOT EXIST.	The subchannel selected on the command line could not be found in the list of applicable subchannels from the data base.
4. TRACE_LEFT IS ZERO IN TABLES CHANGED TO ONE BY ANALYZE - SUB_CHAN XX	The convention is that the first data point in an array will have an index of 1. Zero, therefore, would be out of bounds in this convention.

DIAGNOSTIC	DISCUSSION
5. NOTE! EXPMT_RIGHT IN TABLES = XX TOTAL SAMPLES = YY - SUB_CHAN ZZ	The count associated with experiments in the data base is greater than the number of points in the BIG file.
6. ***** DATA IN SUB-CHAN XX AT BAND EDGE	Data values of either "0" or "255" were observed in the input data file indicating that the actual value could have been out of range.

6.0 USING PRELEWD, WITH COMMENTS ON GRAFPAK

6.1 Introduction

PRELEWD is a first-look plotting program for use with the NTS Instrumentation System suite of codes. It reads channel (CHN) files and generates a hardcopy plot for each channel as a part of the run sequence. Because it operates in BATCH mode during the run sequence, it is very fault tolerant. PRELEWD can plot on a limited set of graphics terminals, and is therefore widely used by our customers. It does not have any analysis capability, but there are a limited number of plot formatting instructions available through the ICF. The underlying graphics package is GRAFPAK-GKS.

6.2 Definitions

Some useful terms are listed below:

(1) The sequence xxxyzzz-*nn* is the historical source-collection-channel-subchannel sequence by which the provenance of the data is known. Where

a) xxx is the source code, i.e., R11, for the RTD720 with bus address 1 on RMV 10 port 1;

b) y is the collection code, i.e., S, for source/collection devices like the RTD720; M, for Mass Memory; L, for HDDR; T, for HDDR tape recovery;

c) zzz is the octal channel number, i.e., 007 or 177 for either the HDDR or MM, and the decimal channel number for S devices like the RTD720s;

d) *nn* is the subchannel number for all non-SANDUS devices. "00" implies data for the entire channel, i.e., a raw data file, *nn* > 0 implies all or some part of that channel's data.

(2) CHN file refers to a file with a name of the form CHNxxxyzzz-*nn*.DAT. For additional information on CHN files,see CHANNEL DATA FILE FORMAT, Appendix D.

(3) PLT file refers to the generated plot file PLTxxxyzzz-*nn*.DAT, where the xxxyzzz-*nn* comes from the first (and possibly the only) data file in the plot file. PLT files are generated by GRAFPAK in the QUIC format

(4) SAD file refers to the generated ASCII file SADxxxyzzz-*nn*.DAT, where the xxxyzzz-*nn* comes from the CHN file used. SAD files are only created when using the SRAD or QRAD qualifier.

(5) Directory_name refers to the NODE::DISK:[DIRECTORY] description that is either an implied or stated part of every file name.

(6) File_spec refers to the entire file description, NODE::DISK:[DIRECTORY]FILE_NAME.EXTENSION;VERSION, or as much of it as is required.

(7) [...] indicates that what is enclosed in [ ] is optional.

6.3 GRAFPAK-GKS Comments

GRAFPAK is the name for the shared library of subroutines that creates the graphics. It is a product licensed from ATC* in California, and may not be distributed outside the 9320 network without a valid license. It conforms to the GKS standard. (A useful reference is *Computer Graphics Programming - GKS - The Graphics Standard* by Enderle, Kansy, and Pfaff published by Springer-Verlag, 1987. Later editions exist.) PRELEWD generates plots from CHN files, and uses GRAFPAK to accomplish this. GRAFPAK can be used with either FORTRAN or C. Both *GRAFPAK-GKS A Fortran Reference Manual*, and *GRAFPAK-GKS C Reference Manual* are available from ATC.

GRAFPAK must exist on any computer on the 9320 network where PRELEWD runs. All logicals necessary to use GRAFPAK are defined system wide.

6.4 Running PRELEWD

PRELEWD.EXE should already be distributed to all 9320 network computers. It is often executed as a part of the run-time sequence, in which case, all definitions, privileges, and permissions are taken care of by the user account CONTROL. To run PRELEWD, the user must define the VMS-like command PRELEWD by entering the DCL command:

```
$ SET COMMAND LD:[PRELEWD]PRELEWD_COMMAND_DEF.CLD
```

---

* Advanced Technology Center, 22982 Mill Creek Drive, Laguna Hills, CA 92653, (714) 583-9119- FAX (714) 583-9213

This definition should be placed in LOGIN.COM on all nodes intended for PRELEWD. This uses the DEC Utility, Command Definition described in the VMS Programming Manual Volume 2B. PRELEWD_COMMAND_DEF.CLD is the definitive authority on PRELEWD's parameters and qualifiers. It is an editable ASCII file.

Having defined the verb PRELEWD, PRELEWD can now be executed by entering:

$ PRELEWD[/qualifier(s)] P1

P1 is described in Section 6.5 and the qualifiers are described in Sections 6.6 and 6.6.1.

6.4.1 PRELEWD Command Line Examples

The following are some examples of command lines that might arise in actual usage. These should cover most user situations. The specific meaning of each qualifier is described in Section 6.6.

(1) $ PRELEWD/DEVICE=2 CHNR11S011-01

This command line plots the file CHNR11S011-01.DAT on device 2, which is a Tektronix 4208.

(2) $ PRELEWD/DEVICE=1/QUEUE CHNR22S021-00

This line creates a hardcopy plot of the raw data file CHNR22S021-00 and prints it using the default queue on the current node.

(3) $ PRELEWD/DEVICE=1/SEND=B::USER:[HOME]/QUEUE=XYZ -
C::OTHER:[HOME]CHNR12S011-03.DAT;3

This command uses C::OTHER:[HOME]CHNR12S011-03.DAT;3 to create a PLT file, which is an RTD720 CHN file located in a directory on node C. It sends the PLT file to B::USER:[HOME] for plotting on queue XYZ on node B.

(4) $ PRELEWD/DEVICE=4/SRAD DD:[SOURCE.RMV32.DIAG]CHNR32S* -

This line creates screen plots on a VT340, and SRAD files for all the CHNR32S* files in the directory DD:[SOURCE.RMV32.DIAG].

(5) $ PRELEWD/DEVICE=1/QRAD/NOPLOT/NOCALIBRATE "" -
COLLECTION.LIS

This command line creates short SRAD files with Y-axis scaling in COUNTS, without creating plot files, for each file listed in the ASCII file COLLECTION.LIS.

6.6 PRELEWD Parameters

There is one and only one REQUIRED parameter, p1, which is a file_spec, and may have wildcards.

> WILDCARDS ARE DANGEROUS!!!
>
> If wildcards are used, be very careful that all files selected will be CHN files. PRELEWD gets severe indigestion from files not in the CHN format.

P1 specifies the data file(s) and, if missing on the command line, the following prompt will be issued:

"Filename (wild cards OK) please"

PRELEWD adds a .DAT extension if it is missing. If a version number is specified, it will be used. If there are several versions of a CHN file, and no version is supplied, PRELEWD follows the DCL convention of using the latest version number. All the input files are opened as READONLY.

If the plots are NOT directed to a terminal, a PLT file is created for every *n* input files, where *n* defaults to 4 or is obtained from the /GROUP qualifier. The PLT file is created in QUIC format. The location of the PLT file will be as explained below under the /SEND qualifier.

A second parameter, p2, is expected if and only if p1 is the empty string, "" (two double quotes with NO intervening characters). See paragraph 6.4, example 5. P2 is optional and, if present, is the name of a file containing a list of CHN files. Wildcards are not allowed. The files may come from different nodes, disks, and directories, and each piece of the file_spec holds until a new, corresponding piece is encountered. The files must be one file_spec per line, with not more than 512 files total.

6.6 PRELEWD Command Qualifiers

The verb defined by PRELEWD_COMMAND_DEF.CLD has the sixteen optional qualifiers listed below, each of which is followed by its default value. A complete description of each qualifier follows in Section 6.6.1.

	QUALIFIER	DEFAULT
Qualifiers not intended for the casual user:		
(1)	/DEBUG	not present
(2)	/MONITOR=*string*	not present
(3)	/COMM_SYM=*string*	not present
Qualifiers for file creation or disposition:		
(4)	/QUEUE{=queue_name}	/NOQUEUE
(5)	/SEND=directory_name	/NOSEND
(6)	/SRAD{=directory_name}	/NOSRAD
(7)	/QRAD{=directory_name} (SRAD and QRAD are mutually exclusive)	/NOQRAD
(8)	/NOPLOT (This qualifier only allowed with /SRAD or /QRAD)	/PLOT
(9)	/GROUP{=n}	/GROUP=4
Qualifiers that modify the plot's appearance:		
(10)	/NOGRID	/GRID
(11)	/NOLEGEND	/LEGEND
(12)	/SINGLE (This qualifier not allowed with /SRAD or /QRAD)	/NOSINGLE
(13)	/MUX	/NOMUX

	QUALIFIER	DEFAULT
(14)	/DEVICE=n (If interactive, the user will be prompted for this information)	/NODEVICE

Qualifiers that control PRELEWD logic:

(15)	/NOCALIBRATE	/CALIBRATE
(16)	/SORT	/NOSORT

6.6.1 PRELEWD Command Qualifier Definitions

(1) /DEBUG

If used, this qualifier MUST follow PRELEWD. It causes an alternate image to be invoked that has been linked /DEBUG with DEC's debugger utility. DEBUG may not be negated, just omitted. Generally, it is intended for software developers' use only.

(2) /MONITOR=*string*

This special qualifier is used ONLY when PRELEWD is invoked by SUPERMON. *String* contains information necessary to establish a communications link from PRELEWD to SUPERMON, and allows PRELEWD to inform SUPERMON of its progress. It is not necessary for a PRELEWD user to supply this qualifier; in fact, it may cause an error.

(3) /COMM_SYM=*string*

This special qualifier is used ONLY when PRELEWD is invoked while using the CONTROL account and RTD_SCHEDULER.COM. The casual PRELEWD user should not supply this qualifier. *String*.DAT is the name of a file into which messages are written, and from which the RTD_SCHEDULER.COM reads those messages for display on the screen.

(4) /QUEUE[=queue_name], (default queue_name=SYS$PRINT)
/NOQUEUE (default)

The queue_name, if supplied, must be a laser printer queue to which output plot files can be directed. If /QUEUE is specified, but the queue_name is omitted, the default queue_name is SYS$PRINT. The laser printer must be able to accept QUIC commands.

For additional information, see Section 6.6.2 and the /SEND qualifier.

(5) /SEND=directory_name
/NOSEND (default)

The /SEND qualifier tells PRELEWD what to do with PLT files. PLT files are always created in the CURRENT DIRECTORY unless the qualifier specifies a directory_name on the CURRENT NODE. In that case, PLT files are created there. If the specified directory_name is on another node, then the PLT files are created in the CURRENT DIRECTORY and, as each PLT file is generated, it is copied to the new node using the LD:[TOOLS]NET_EXECUTE.COM procedure. If /QUEUE was specified, then the COPY is accompanied by the appropriate PRINT command to be executed on the destination node. This is done by SPAWN of a command procedure with the same name as the plot file, but with a .COM extension.

WARNING

Using the CONTROL account gives the user all necessary privileges and permissions. If you are not running under CONTROL, and PLT files are to be sent to another directory_name, YOU MUST HAVE WRITE PERMISSION in that directory_name. You must always have write permission in the current directory.

If both the /QUEUE qualifier and the /SEND qualifier are missing or negated, the plot file(s) will NOT be printed, and will remain in the current directory. If the PLT files are sent to another node, a copy remains in the current node even though they are deleted from the destination node after printing.

(6) /SRAD[=directory_name]
/NOSRAD (default)

OR

(7) /QRAD[=directory_name]
/NOQRAD (default)

If present, and not negated, either qualifier causes SAD files to be created in directory_name, or in the current directory if directory_name is omitted. The format

of an ASCII SAD file is understood by the program UFO on the 9310 computer system. The data in the files is in the same units as the plot. Not all CHN channel types can currently be put into this format, but SANDUS, 7912, and RTD720 channels can. T48 files and CAL files from 7912s may also be made into SAD files.

/SRAD causes the creation of SAD files with ALL the data from the CHN file, that is, before data thinning, while /QRAD causes the creation of SAD files containing plot information after data thinning. The amount of space used by a SAD file is significantly greater than the original CHN file. The /QRAD qualifier creates smaller SAD files than the /SRAD qualifier but even these can be large. /QRAD was created for use with realtime files from a SANDUS. It would be a useful qualifier for very large RTD720 data files.

(8) /NOPLOT
/PLOT (default)

The /NOPLOT qualifier is allowed only with the /SRAD (or /QRAD) qualifier, and inhibits the generation of PLT files. This will not save much time, but it does conserve disk space. If /NOPLOT is present, then /NOSEND, /NOQUEUE, and /DEVICE=1 are implied.

(9) /GROUP{=n}
/GROUP=4 (default)

The /GROUP qualifier need not be present, and if present need not have a value. The default value is 4. It is the number of input files per PLT file. *n* is bounded between 1 and 20. The *xxxyzzz-nn* part of the PLT file is taken from the first data file in that plot file, regardless of the size of /GROUP.

(10) /NOGRID
/GRID (default)

This qualifier deletes the grid that normally appears on a plot.

(11) /NOLEGEND
/LEGEND (default)

This qualifier deletes the legend that normally appears on the plot.

(12) /SINGLE
/NOSINGLE (default)

This qualifier causes 7912 traces to appear as a single line, not as the two lines that normally appear. The qualifier is optional, and is negatable. It is not allowed with the /SRAD (or /QRAD) qualifier described above. The algorithm is: y(plotted)=[y(upper) + y(lower)]/2.0

(13) /MUX
/NOMUX (default)

A MUX channel contains up to 32 subchannels, and the default is to plot ALL subchannels on one page(screen). This qualifier, if present and not negated, causes each subchannel to be plotted on ONE page/screen. It takes significantly more time to plot SANDUS ANALOG MUX channels this way. This type of channel has not been used for several tests.

(14) /DEVICE{=n}
/NODEVICE (default)

GKS must know what the plotting surface is going to be. This qualifier allows the user to select the output device on the command line. If not selected here, the program will ask the question interactively. This is useful when PRELEWD is called from a command procedure. The qualifier is optional and may be negated. If present, the default value is 1 (laser). Any value supplied should be between 1 and 5 inclusive, and the user is NOT asked any questions. The *n* should be chosen from the information below describing the response to request for an output device. PRELEWD will display ATC's GRAFPAK logo. If run interactively, and if the

/DEVICE qualifier is not present or is negated, PRELEWD will ask the user once for one piece of information:

"Please enter the plot device desired:"
(1) QMS in landscape mode
(2) TK4208 terminal
(3) VT330
(4) VT340
(5) X-WINDOWS
(Respond with a number from 1-5, default=1):

If PRELEWD is run in BATCH mode or as a DETACHED process, /DEVICE=1 is chosen by default.

When plotting to a terminal, PRELEWD will pause between plots. It will continue after the user strikes a RETURN key.

(15) /NOCALIBRATE

The Y-axis scaling is determined by the ICF value of PLOT_OPTION. If the /NOCALIBRATE qualifier is specified, no calibration information will be used and while the format of the plot will be normal, the Y-axis will be forced to COUNTS.

(16) /SORT
/NOSORT (default)

The /SORT qualifier is valid only if the data files are specified in parameter p1, and only if there are more that 20 data files. If specified, the data files are ordered for plotting first by experimenter name and then by experiment identification. This is a political qualifier.

6.6.2 PRELEWD'S Print Command

PRELEWD directs each PLT file to the requested queue as it is generated with the following command:

$ PRINT/DELETE/QUEUE=queue_name/PASSALL/NOFEED/NOFLAG -
/NOTRAILER/SETUP=QC/NOBURST directory_name:PLT*xxxyzzz-nn*.DAT

Special features of this print command include the following items:

(1) The file directory_name:PLT*xxxyzzz-nn*.DAT is deleted after printing. PLT files are created in the current directory on the execution node regardless of where they are to be sent. If they are sent on to another node, the copy at the final node is the one deleted, the original remains. In other words, the /DELETE applies only to the node from which printing occurs. This feature is less useful with the advent of network printing where a queue name is all that is required.

(2) The /PASSALL qualifier is REQUIRED by GRAFPAK, regardless of who initiates the printing. This is the most common mistake made by the casual PRELEWD user

(3) /SETUP=QC works for Talaris 1590-T printers that contain firmware to change QUIC commands to XCEL commands. A modification to SYS$LIBRARY:TALDEVCTL.OLB to add a text module named QC is required if the queue is not a Talaris 1590-T, or if the 1590-T is not properly configured.

(4) The command eliminates miscellaneous extra pages insofar as possible.

6.7 PRELEWD General Comments

In addition to any PLT files and SAD files that may be created, a PRELEWD.LOG file is created for each run. This file contains information relevant to that run, the command line, the creation date of the version of PRELEWD being executed, the specific files examined, and number of points plotted for each. If anything unusual happens, it is recorded in PRELEWD.LOG. This file will be created in the current directory. PRELEWD.LOG is formatted to allow its use with SLOG (Summary LOG, Appendix I).

ABNORMAL TERMINATION

Should PRELEWD fail, or should you use CONTROL Y to exit PRELEWD, the terminal can be left in an indeterminate state, possibly requiring it to be RESET.

6.7.1 Common Error Messages

Several common error messages and possible solutions are listed below:

(1) "error while opening unit outlog, probably no write permission in directory" OR
"It is very probable you do not have write permission in the current directory" OR
"Unable to open SRAD file, do you have write permission in the data directory?
No SRAD file creation"

Unable to create a file, due to no write permission in the current directory.
Usually the user is trying to run PRELEWD from the DD directory tree.

(2) "Supplied indirect file list, file not found?"

The parameter P1 was "^", and either P2 was missing, or the file_spec was improper. See Section 6.5.

(3) "No time cals for this channel because there were too few zero crossings. The number found was nn"

Tektronix 7912 calibrations are based on the baseline crossings of a sine wave, and at least "min-cycles" full cycles are required for a time calibration. PROCESS calculates the number

of crossings that did occur, and there were too few. The data are abnormal or (usually) the screen intensity was too low. There are several error messages associated with 7912 calibration, but this is most frequently seen.

(4) "Prelewd found no acceptable files"

P1 was improperly typed, i.e. CNHR31*. not CHNR31*.

6.7.2 Error Message Levels

There are three levels of error messages, INFORMATIVE, NON-FATAL ERROR, and FATAL ERROR.

An example of a message is (here, an echo of the command line as entered):

```
          (A)                (B)              (C)
** ** INFORMATIVE from  COMMAND  For your information
  PRELEWD/DEV=2 UK:[PKAESTNER.TEST.HT_DATA]CHNR12S031-01      54
                                (D)                           (E)
```

where the parts are:

(A) Level of the error message;

(B) Name of the subprogram which originated the message;

(C) One of several canned greetings from the message subprogram;

(D) Text of the message, usually the heart of the problem;

(E) An integer, which may or may not be meaningful. A maintenance programmer should look at the source code.

A variant of this format is:

NEW DATA FILE INFORMATIVE from HANDLE A FILE Current data file is
UK:[PKAESTNER.TEST.HT_DATA]CHNR12S031-01.DAT;1 0

which gives the file_spec for the current CHN file. A list of the CHN files found is written to the log file earlier. This message gives the user a progress report.

7.0 RTD TEST OPERATOR INFORMATION

7.1 Introduction

This section describes the operators manual for the software system (RTDTEST) that manages Tektronix RTD720 Digitizers (RTDs) using Digital Equipment Corporation (DEC) computers and workstations.

7.2 RTDTEST Functions

RTDTEST is an interactive menu driver system. The user can only access this system by signing onto CONTROL and selecting the RTDTEST option. The MAIN menu will appear giving the user several options. Most menus (forms) give the user the opportunity to QUIT and or move back to the PREVIOUS menu. All user responses are validated and any incorrect response is indicated at the bottom of menu with a message. Most messages show the correct responses allowed. There is no limit to how many times an invalid response can be made by the user.

The functions of the instrument control software allow the user to modify RTD settings, modify ICFs, and flag the INGRES software to update the data base with modified ICFs. The user can interactively acquire, analyze, and display (plot) dry run data at the terminal. Additional functions allow the user to send commands to an RTD, obtain status information, and calibrate the RTD720.

7.2.1 Set Up the RTD from the ICF

This option allows the operator to set up a RTD for Dry Run/Shot Configuration based on information residing in the INGRES data base. It accomplishes this by reading the ICF and sending the appropriate commands (ASCII strings) to an RTD. (Refer to 7.2.2 and 7.2.3.)

7.2.2 Modify the ICF

The ICF can be modified by changing device (RTD) settings according to the options for each function listed below.

Acquire:

a) Internal,
b) Length,
c) Mode,
d) Clock,
e) Number of records, and
f) State

Trigger:

a) Source,
b) Coupling,
c) Slope,
d) Type Level,
e) Level,
f) Type position,
g) Position, and
h) Mode

Arming can be set to:

a) External, or
b) Internal

Channel settings can be modified according to the options for each function listed below.

General settings:

a) Range,
b) Offset,
c) Type offset,
d) Coupling,
e) Band-width, and
f) Channel off/on

Subchannel:

a) Left bound, and
b) Right bound

Note: This for the first ten (1-10) subrecords in the INGRES data base. The left and right bound is used only by PRELEWD for plotting, and does not affect the RTD.

The ICF is used as the means to transport these changes to the INGRES data base if required. If not transported, the changes are used only for the current RTDTEST session. (See 7.2.1 and 7.2.3.)

7.2.3 Update INGRES Data Base

At the end of the current RTDTEST session and if an ICF was modified, the operator is prompted as to whether the modified ICF is to be sent to the INGRES data base for updating. If the resonse is YES, the ICFs are copied from their present directory to an INGRES directory.

7.2.4 Acquire Data from an RTD

If an RTD has acquired data previously (acquisition cycle has completed - an event code equal to 465) the operator can pull the data from the RTD and plot it on the terminal being used. (Refer to 7.2.5.)

7.2.5 Plot the Acquired Data to the Terminal

A two-dimensional plot of X,Y data is available if an RTD has acquired data and that data has been written to a temporary file. The option in 7.2.4 has to be performed first. A graphic terminal is required for this function.

7.3 Interface with CONTROL

RTDTEST is accessed through software called CONTROL. A complete description of CONTROL and its interface is discussed in the VAX-available document "RTDTPDWN.HT," R. J. Isidoro of SNL, Department 9321.

REFERENCES

Lee, J. W., 1990. "Digital Signal Processing Workstation User Guide," Unpublished report, Sandia National Laboratories.

Hawkey, M., Undated. Tektronix, Inc., Beaverton, OR, private communication to R. B. Caudell.

DEC (Digital Equipment Corporation), 1988. "Guide to DECnet-VAX Networking," Digital Equipment Corporation VAX System Management, Volume 5, Chapter 2, April 1988.

Caudell, R. B., 1991. "Tektronix RTD720 Digitizer Evaluation during DISTANT ZENITH," memo to W. B. Boyer, Division 9321, November 23.

Isidoro, R. J., 1992. "Preliminary Results of RTD720 Testing on VAX Workstation (VLC 4000) and Other Nodes", memo to distribution, August 19.

SAND93-2064
Unlimited Release
Printed September 1993

Distribution
Category UC-706

Data Collection System Volume 2: Maintenance Manual

Richard B. Caudell, Roger D. Aden,
Robert J. Isidoro, Peter C. Kaestner

Instrumentation Development Department 9321
Sandia National Laboratories
Albuquerque, NM 87185

ABSTRACT

Sandia National Laboratories (SNL) Instrumentation Development Department was tasked by the Defense Nuclear Agency (DNA) to record data on Tektronix RTD720 Digitizers on the HUNTERS TROPHY field test conducted at the Nevada Test Site (NTS) on September 18, 1992. This report contains a overview and description of the computer hardware and software that was used to acquire, reduce, and display the data. The document is divided into two volumes: an operators manual (Volume 1) and a maintenance manual (Volume 2).

CONTENTS

CONTENTS (Continued)

CONTENTS (Continued)

CONTENTS (Continued)

ACRONYMS

ACL	Access Control List
ATC	Advanced Technology Center
CHN	Channel
DCL	Digital Command Language
DEC	Digital Equipment Corporation
DNA	Defense Nuclear Agency
DSP	digital signal processing
FMS	Forms Management System
GKS	Graphical Kernel System
ICF	instrument control file
KSC-1	Kaman Sciences Corporation
LPARL	Lockheed Palo Alto Research Laboratory
NTS	Nevada Test Site
RTD	Tektronix RTD720 Digitizers
SAIC	Science Applications International Corporation
SNL	Sandia National Laboratories
SPL	Software Project Leader

1.0 CONTROL-ACCOUNT SOFTWARE

1.1. Introduction

Normal data acquisition, processing, and display for the Sandia National Laboratories (SNL) Field Instrumentation Departments are done via a limited-access account designated CONTROL. Identical copies of the CONTROL-account software are in directory LD:[CONTROL] on all operational computer nodes used by the SNL Field Instrumentation Departments. The use of a limited-access account permits authorized individuals, such as equipment operators, to use relatively sophisticated software with exceptionally high system-access privileges in a protected environment. Operational characteristics of the software are discussed in subsequent sections. Complete listings of the CONTROL procedures are stored in LD:[CONTROL].

1.2 LOGIN.COM Procedure

A user may access the CONTROL-account software by typing in the user name CONTROL when logging into a VAX node. CONTROL has been set up by the system manager so that no password is needed, and its log-in command procedure (LOGIN.COM) begins running immediately. LOGIN.COM first performs some housekeeping functions, such as defining the INQUIRE command simulator and defining its control-Y and control-T actions. LOGIN.COM then asks the user to enter his or her username and password. If the username and password are valid, LOGIN.COM then checks to see if the username is permitted to operate CONTROL on the current node. If the user passes these checks, the process name is defined to include the user name of the operator so that logs of CONTROL usage may be kept.

LOGIN.COM then determines the name of the computer node on which it is running. If the node name contains the string RMV, then the logical name TABLE_DIR is defined as GEAR10::DD:[TABLES.*node*], where *node* stands for the node-name character string (e.g., RMV10). Otherwise, TABLE_DIR is defined as DD:[TABLES]. Software that runs under CONTROL should use TABLE_DIR to refer to the instrument-control-file directory, so the proper directory will automatically be referenced.

If the node name contains any of the strings ACE, RMV, or SAN, then the operator is asked if the software is to check N12DBM (the data base node) for the most recent versions of whichever instrument-control files are in directory TABLE_DIR. If the node name does not contain one of the instrumentation node strings, N12DBM is unconditionally searched for later versions of the instrument-control files contained in TABLE_DIR. The DCL command procedure MOVE_TABLES.COM, discussed below, is used to perform the search for and any necessary movement of tables (Perkins, 1992).

Once the node name has been established, LOGIN.COM displays a menu appropriate to the node. Several menus are displayed in Section 1.6.

After a given menu is displayed, the operator is asked to enter a choice. Choice L allows the operator to enter notes in the CONTROL-account logbook file; choice M invokes the VMS MAIL utility; and choice E performs some housekeeping functions and logs the operator off the computer. Most nodes offer these generic choices. The numbered choices define software that can be invoked by entering the number associated with the choice. For example, if one chooses number 6 from the ACE menu, CONTROL executes the following DCL commands:

```
$ define uub0 uua0
$ @ld:[dhtest]dhtest
$ exit
```

The *define* command performs a logical definition necessary to run DHTEST on an ACE node. The command *@ld:[dhtest]dhtest* invokes DCL command procedure DHTEST.COM that is resident in the directory LD:[DHTEST]. The *DHTEST.COM* command procedure in turn invokes the FORTRAN-coded executable DHTEST program that performs various test and setup procedures related to SANDACE and SANDUS hardware. Finally, the *exit* command returns control to the main body of the CONTROL-account LOGIN.COM command procedure. At this point the menu is displayed again and the operator may select another menu option.

In most cases, the software invoked outside of LD:[CONTROL] has been written and is maintained by someone other than the CONTROL developer in a directory other than LD:[CONTROL]. For historical reasons, however, some command files invoked by LOGIN.COM are located in LD:[CONTROL].

When an operator selects E and the node name contains any of the strings ACE, RMV, or SAN, then any instrument-control files that have been modified during the current session will be copied from TABLE_DIR to an appropriate directory on N12DBM, where the technical information will be extracted into the instrumentation data base for use in creating new versions of the instrument-control files. If the node is not an ACE, an RMV, or a SANDUS node, or after the instrument-control file copies are completed, the operator is logged off the computer.

1.3 MOVE_TABLES.COM Procedure

Tables are copied to or from instrumentation or recording nodes by using a DCL command procedure called MOVE_TABLES.COM, which is discussed at length by Perkins (1992).

1.4 Hand Dry-Run Software

Software that is specifically designed to set up or exercise instrumentation or recording hardware with significant operator control can be lumped into the "hand dry-run software" category. The category includes ACETEST, AUTOCAL, AUTOSIM, DHTEST, high-density data recorder diagnostics, mass-memory diagnostics, RTD diagnostics, RTDTEST, and UHTEST. Most of the diagnostic or hand dry-run programs are invoked via command procedures running outside LD:[CONTROL]. However, a few programs are still invoked by command procedures in LD:[CONTROL].

1.5 Signal Dry-Run Software

Software that is specifically designed to acquire, process, and plot signal dry-run or shot data can be lumped into the "signal dry-run software" category. At present, this category includes DATALOG, SUPERMON, and RTD_SCHEDULER. SUPERMON and DATALOG are invoked via command procedures running outside LD:[CONTROL], but RTD_SCHEDULER is invoked by a command procedure in LD:[CONTROL].

Even though SUPERMON is invoked by a command procedure that runs outside LD:[CONTROL], the SUPERMON FORTRAN code submits batch jobs that use DCL command procedures resident in LD:[CONTROL].

1.6 Control Menus

The following three sections are control menus (screens) that can be accessed from CONTROL-account software. Each section duplicates the screen display for the given control menu.

1.6.1 ACEm0 Nodes

You are logged into the CONTROL account on node ACE10. CONTROL LOGIN.COM was created 8-APR-1992 16:16:53.00. Your choices from node ACE10 are

(1) Run ACETEST
(2) Plot channels
(3) Show users
(4) Run 7912 AUTOCAL (local)
(5) Dry run report
(6) Run DHTEST
(7) Run DHTEST for a different system
(8) Run AUTOSIM (local)
(9) Get new instrument-control files from data base node
(10) Push modified instrument-control files to data base node
(L) Logbook
(M) Mail
(E) Exit

1.6.2 SANDUS Nodes

You are logged into the CONTROL account on node SAN504. CONTROL LOGIN.COM was created 8-APR-1992 16:16:53.00. Your choices from node SAN504 are

(1) Fetch, process, and plot data
(2) Run DHTEST
(3) Run DHTEST for a different system
(4) Show users
(5) Get new instrument-control files from data base node
(6) Push instrument-control and master files to archival and backup nodes
(7) Dry run report
(L) Logbook
(M) Mail
(E) Exit

1.6.3 GEAR10

You are logged into the CONTROL account on node GEAR10. CONTROL LOGIN.COM was created 8-APR-1992 16:16:53.00. Your choices from node GEAR10 are

(1) Run RTD720 dry-run scheduler
(2) Run RTD-digitizer diagnostics
(3) Get new instrument-control files from data base node
(L) Logbook
(M) Mail
(E) Exit

1.6.4 N12VAX

You are logged into the CONTROL account on node N12VAX. CONTROL LOGIN.COM was created 8-APR-1992 16:16:53.00. Your choices from node N12VAX are

(1) Run SUPERMON
(2) Run DHTEST
(3) Run 7912 AUTOCAL (ethernet)
(4) Run AUTOSIM
(5) Get new instrument-control files from data base node
(L) Logbook
(M) Mail
(E) Exit

1.6.5 HDDRa Nodes

You are logged into the CONTROL account on node HDDRA. CONTROL LOGIN.COM was created 8-APR-1992 16:16:53.00. Your choices from node HDDRA are

(1) Run 7912 AUTOCAL (remote bitstream)
(2) Run HDDR diagnostics
(3) Get new instrument-control files from data base node
(4) Run ACETEST
(5) Run uphole hardware test program UHTEST
(6) Archive today's AUTOCAL and LASER-cal files
(L) Logbook
(M) Mail
(E) Exit

1.6.6 MM12a Nodes

You are logged into the CONTROL account on node MM12A. CONTROL LOGIN.COM was created 8-APR-1992 16:16:53.00. Your choices from node MM12A are

(1) Run mass-memory diagnostics
(2) Run uphole hardware test program UHTEST
(3) Get new instrument-control files from data base node
(L) Logbook
(M) Mail
(E) Exit

1.7 LD:[CONTROL] Directory

Table 1.1 lists the directory of files in LD:[CONTROL].

Table 1.1 Directory of LD:[CONTROL] files.

File Name	File Name
ABORT_JOB.COM;35	DBG720.COM;17
LD:[SUPERMON]	DRYRUN_BACKGROUND.COM;122
ACEDEV_DBG720_INIT.COM;2	LD:[SUPERMON]
ACE_PROC.COM;28	DRYRUN_DSP.COM;46
LD:[ACETEST]	LD:[SUPERMON]
ARCHIVE_MM.COM;6	DRYRUN_FOREGROUND.COM;109
LD:[SUPERMON]	LD:[SUPERMON]
CONTROL.LOG;1	DSPUSERDEF.COM;32
COPY_FILE.COM;4	NET_EXECUTE.COM;56
LD:[DSP]	PROCESS_PLOT.COM;12
EDTINI.EDT;4	LD:[SUPERMON]
HDDR_DIAGS.COM;16	REM_EXECUTE.COM;28
LD:[FETCH]	RMV_DBG720_INIT.COM;2
INQUIRY.EXE;30	RTD720.COM;4
INQUIRY.FOR;60	RTD_SCHEDULER.COM;82
LOAD_8330.COM;8	LD:[REALIZE]
LD:[8330DECOM]	RUN_ENABLIZE_RTD720.COM;13
LOGIN.COM;277	LD:[REALIZE]
LOGNET.COM;8	RUN_INITIALIZE_RTD720.COM;15
LOGPERK.COM;7	LD:[REALIZE]

Table 1.1. Directory of LD:[CONTROL] files.

File Name
MM_DIAGS.COM;12
MOVE_TABLES.COM;65
RUN_REAL.COM;10
RUN_REALIZE_RTD720.COM;41
LD:[REALIZE]
RUN_RTD_DIAGS.COM;7
LD:[REALIZE]
SANDUS_PROC.COM;33
TEST_717.EXE;9
TEST_717.FOR;12
TEST_8330.EXE;7
TEST_8330.FOR;10
TEST_SRCH.EXE;5
TEST_SRCH.FOR;8
TEST_ZT.EXE;3
TEST_ZT.FOR;4

2.0 REALIZE AND INITIALIZE MAINTENANCE INFORMATION

2.1 Introduction

REALIZE and INITIALIZE are two programs in the Nevada Test Site (NTS) Instrumentation System suite of codes. They are very similar, using many of the same modules. INITIALIZE reads the instrument control file (ICF) and sets up RTD720s as requested. REALIZE reads the ICF, fetches data from RTDs, and writes it to a BIG file. Because REALIZE and INITIALIZE are so similar, they will be documented together and, unless a specific one is indicated, all information pertains to both.

The NTS Instrumentation System is a sequence of programs that work across an extensive network of special and general-purpose computers, with special hardware to record and to display short-lived phenomena captured from a hostile environment. The data is received by one of several types of source devices. Source devices such as the SANDUS can record many channels of data, while other source devices can only record one channel.

With the recent advent of newer digitizers that have their own memory (the RTD720 specifically), a new sequence of programs has emerged. REALIZE recovers data from the RTD720s and packs it into BIG files, which ANALYZE reformats into individual CHN files. Neither program can be viewed as a single entity, as they are closely tied to both the hardware and the software of SNL's NTS Instrumentation System. A change in any part of this system may affect REALIZE, because it prepares BIG files for use by ANALYZE, which in turn prepares CHN files for use by PRELEWD. Both programs use ICFs prepared by still another program.

2.2 Overview

This is not a maintenance manual that lists troubleshooting procedures or instructions for adding undocumented features. Maintenance information is often obtained from the source, which contains comments for guidance. This document, instead, discusses REALIZE/INITIALIZE and the routines that perform specific tasks.

This document is directed to persons familiar with the Run Time Library and Digital Equipment Corporation's (DEC) FORTRAN, particularly DEC's version of structures and records. A knowledge of the RTD hardware is not necessary. It assumes that the user has read and is familiar with Section 4.0 of the Operators Manual, REALIZE and INITIALIZE User Information. It is also helpful if the user has also run one or both of the programs.

2.2.1 Programming Constraints

REALIZE was written as an interface to software that sets up and recovers data from RTD720s. INITIALIZE was written later to allow the setup function to be done earlier in time, and independent of the data recovery. Both are significantly easier to understand than FETCH or DECOM, partly because they communicate through a GPIB in ASCII. The only constraints were those of time. They had to be written quickly. Clock time is not important, as the RTD's were uphole and would retain the data for a very reasonable period.

Previous experience with Tektronix 7912s and LeCroy 6880s led to the techniques used for the RTDs. BIG files and ICFs were already available in addition to very good software with which to manipulate both. The command procedures were in place to move the ICFs as needed. The only significant software that had to be written was the device communication software.

2.3 References for REALIZE and INITIALIZE

The following list of references is specific. It does not include references in the VAX/VMS Documentation Set, the VAX/VMS FORTRAN Reference Manual or User's Manual. Please read the INCLUDE files, because they contain significant comments.

2.3.1 Documentation

REALIZE and INITIALIZE User Information - Section 4 of the Operators Manual (Volume 1).

It contains a description of REALIZE and INITIALIZE and of the command line qualifiers available to each.

REALIZE and INITIALIZE Interface Manual - Section 3 of the this volume.

This section describes routines called from REALIZE and INITIALIZE, the node-to-node interface, and routines that interface directly to the RTD720s. Specifically, it describes the communications interface that exists on the REALIZE execution node.

INSTRUMENT CONTROL FILE for Tektronix RTD720 Digitizers - Appendix A

This is documentation on the ICF file for the RTDs. It will aid in understanding both INITIALIZE and REALIZE. Both programs must read the ICF to find out what equipment is available, and how it is to be configured. Both programs must pass selected ICF records to the communications routines.

Instrument Control File (Table) Utility Routines - Appendix B

This documents the package of routines needed to create, read, write, modify ICFs. Understanding ICFs is essential to understanding both INITIALIZE and REALIZE.

BIG File Format - Appendix C

This is documentation on the BIG file. Understanding BIG files will aid in understanding REALIZE.

BIG File Utility Routines - Appendix E

This documents the package of routines that create, read, write, and modify the BIG files.

Maintenance of ICF (Table), BIG, and Channel File Utility Routines - Appendix F

This is the maintenance document for both the TABLE routines and the BIG routines.

2.3.2 Structure Definition Files

Because both REALIZE and INITIALIZE are going to read records from the ICF, and because REALIZE will write a BIG file, it is necessary to define several structures that are local, but consist in part of structures defined for the NTS Instrumentation System.

REALIZE_STRUCTS.DEF contains two structures. It does not reserve any memory. It does include the file REALIZE.PRM. Both these files are referenced with the directory_name symbol REALIZE$INCLUDE, which is defined:

$ DEFINE REALIZE$INCLUDE LD:[REALIZE] (or wherever the files are located).

The structure /ONE_RTD/ is used to pass information to the communications routines. It defines several variables available from a record within the structure. This is to make the variables more accessible.

The structure /CHANNEL_INFO_STRUCT/ is used to help form the BIG file header. It too defines variables available in the records that are part of the structure.

The following three files are referenced with the directory_name symbol INCLUDE$INCLUDE, which is defined:

$ DEFINE INCLUDE$INCLUDE LD:[INCLUDE] (or wherever they are located).

TABLE_STRUCTS.DEF is one of the important files containing structure definitions needed to understand the ICF and the BIG file. It does not reserve any memory.

BIG_STRUCTS.PRM contains, among other parameters, the maximum number of subchannels allowed for RTD720s.

BIG_STRUCTS.DEF is the second of the important files containing structure definitions needed to understand the BIG file. It defines the BIG file header and the header for each type of source equipment. BIG_STRUCTS.DEF requires BIG_STRUCTS.PRM, but it is not called out.

2.3.3 Other INCLUDE Files

There are comments in these files. Generally they are included with the /NOLIST option; however, each main program has them listed. These files are usually maintained in REALIZE$INCLUDE.

REALIZE.PRM contains parameters that define the maximum size of most of the REALIZE arrays. Other necessary parameters have been included here for consistency.

ADMIN.CMN declares many of the variables that are needed in REALIZE. It defines the common blocks:

(a) ADMIN_INT_1 - integer variables
(b) ADMIN_INT_2 - additional integer variables
(c) ADMIN_CHAR_1 - string variables
(d) ADMIN_CHAR_LENGTH_1 - the "real" length of the string variables
(e) ADMIN_LOG_1 - logical variables

2.3.4 Command Definition Files

REALIZE and INITIALIZE have a number of command line qualifiers, but no parameters. All information about the RTD720s comes via the required /TABLE qualifier. The CLD files are usually maintained in REALIZE$INCLUDE.

INITIALIZE_COMMAND_DEF.CLD is the file that defines the command line images and qualifiers specifically for INITIALIZE. It is the ultimate authority on the INITIALIZE command line qualifiers.

REALIZE_COMMAND_DEF.CLD is the file that defines the command line images and qualifiers specifically for REALIZE. It is the ultimate authority on the REALIZE command line.

2.4 Useful Information

The source code is all written in FORTRAN. VAX/VMS Run Time Library routines and System Service Routines are used. There is system level I/O while reading the ICF and writing the BIG files. Each source file contains one program module with the same name as the file. The source is compiled /NOOPT/DEBUG, and the object modules are maintained in the object library REALIZE.OLB, created with the Librarian Utility. Because of the I/O in REALIZE, using the /OPTIMIZE qualifier is wasted, and therefore only one object library is maintained. REALIZE.OLB resides in the directory_name REALIZE$LIBRARY, which has to be defined by the programmer as follows:

$ DEFINE REALIZE$LIBRARY LD:[REALIZE] (or wherever the file is maintained).

Four executable files are normally created from REALIZE.OLB. They are REALIZE.EXE and DEBUG_REALIZE.EXE for REALIZE, and INITIALIZE.EXE and DEBUG_INITIALIZE.EXE for INITIALIZE. The DEBUG_... executables are made with the /DEBUG qualifier on the LINK command.

In addition to the .FOR files that contain the source, there are several .COM files that may be useful. Because of the limited disk space available, most .COM files that create files also do a PURGE. After the object modules have been introduced into REALIZE.OLB, they are deleted.

The important .COM files follow:

(a) REALLIB.COM requires one parameter: the name of module to compile. The resulting object module is put into REALIZE.OLB, and the object file is deleted. The compile command is:

$ fortran/list/nooptimize/debug/extend 'p1'

(b) REALIZE.COM is used to create REALIZE executables. It uses REALLIB and requires one parameter, as described above. After REALLIB.COM is done, or if the required parameter is NONE, it does a LINK to create module REALIZE.EXE, and a LINK/DEBUG to create DEBUG_REALIZE.EXE. REALIZE.COM requires that, in addition to the symbols named above, the symbols UTILITY$LIBRARY and PRELEWD$LIBRARY should be defined as follows:

$ DEFINE PRELEWD$LIBRARY LD:[PRELEWD] (or wherever PRELEWD.OLB is maintained)

$ DEFINE UTILITY$LIBRARY LD:[UTILITY] (or wherever UTILITY.OLB and UTILITY.DBG_OLB are maintained)

(c) INITIALIZE.COM is used to create INITIALIZE.EXE, and DEBUG_INITIALIZE.EXE. It requires the same symbols.

Because there is so much shared code in the two programs, the operator must relink both REALIZE and INITIALIZE when a source module is changed. Assuming modules BLAH.FOR and CRAM.FOR have been changed, the following sequence is suggested:

$ @REALLIB BLAH
$ @REALLIB CRAM
$ @REALIZE NONE and/or
$ @INITIALIZE NONE

2.5 Source Files, Purpose, and Contents

There are a number of source files, each containing one module with the same name as the file.

Production source code, the object library, executables, and include files are usually kept in the directory REALIZE$INCLUDE. Development source code, its object library, and executables are maintained in another directory. A distinct .CLD file must be maintained in each directory.

This section is intended to provide a quick guide to the location of an item. Each module is prefaced with a more elaborate description of its purpose, arguments, modules called, and calling module(s). The major modules also contain a history of their significant changes. Unless noted otherwise, the routines named below are used by both REALIZE and INITIALIZE.

(1) REALIZE.FOR is a main program, and controls REALIZE's overall logic.
(2) INITIALIZE.FOR is a main program, and controls INITIALIZE's overall logic.
(3) COMMAND_LINE.FOR parses the REALIZE command line.
(4) INITIAL_COMMAND_LINE.FOR parses the INITIALIZE command line.
(5) ENVIRONMENT.FOR finds the operating environment, current directory, etc.
(6) FOUND.FOR is an error routine that writes the LOG file. See paragraph 6.8.3.
(7) GET_DEV_CHAN.FOR determines the devices and channels in the ICF, then does an AND with any command line device and/or channel requests.
(8) PARSE_PETE_NUM_LST.FOR decodes numeric lists from command line qualifiers.
(9) READ_RTD_TABLE_RECORDS supervises recovery of all remaining ICF records needed.
(10) READ_RTDCS.FOR reads the subchannel record /RTD_CC_SUB_DESC/ from the ICF.
(11) READ_RTDES.FOR reads the subchannel record /RTD_EXP_SUB_DESC/ from the ICF.
(12) READ_RTDLS.FOR reads the subchannel record /RTD_LC_SUB_DESC/ from the ICF.
(13) READ_RTD_CHANS.FOR reads the record /RTD_CHAN_DESC/ from the ICF.
14) READ_RTD_DEVICES.FOR reads the record /RTD_DEV_DESC/ from the ICF.
(15) WRITE_BIG_FILE.FOR writes the BIG file. (Used only by REALIZE.)

2.6 External Modules

Two external object libraries supply modules not in the above list of source files.

2.6.1 Modules Called from LEWD.OLB

LEWD.OLB is external to a number of other executables. It contains several routines that are similar to UTILITY routines, and should be specified first in the LINK process.

(1) CLI returns the command line which is sent to the .LOG file.

(2) ELIMINATE_BLANKS eliminates leading and trailing blanks from a string, and returns the truncated string and the number of characters in it.

(3) PARSE_NUMBER extracts a number from the command line qualifier.

(4) PARSE_STRING extracts a string from the command line qualifier.

2.6.2 Modules Called from UTILITY.OLB

This object library is really two libraries, UTILITY.OLB and UTILITY.DBG_OLB, where UTILITY.DBG_OLB is compiled with the /NOOPT/DEBUG qualifiers. These libraries consist of a number of useful basic modules written by the staff and available to all.

(1) CHANGE_COMM_SYM creates or modifies the file string.dat, where string is the COMM_SYM qualifier's value. It opens the file string.DAT, and appends the input string to that file. The calling command procedure reads this file, and displays the newest addition on the screen.

(2) CLOSE_BIG closes a BIG file, and is part of the BIG_ROUTINES set of modules.

(3) CLOSE_TBL closes an ICF, and is part of the TABLE_ROUTINES set of modules.

(4) CNV_NUM_LST_TBL converts a numeric string to integer representation.

(5) FILE_DATE obtains the creation date of the calling executable module.

(6) OPEN_BIG opens a BIG file, and is part of the BIG_ROUTINES set of modules.

(7) OPEN_TBL opens an ICF, and is part of the TABLE_ROUTINES set of modules.

(8) PARSE_PRESENCE gets information about a command line parameter.

(9) READ_TBL_REC reads a named ICF record, and is part of the TABLE_ROUTINES set of modules.

(10) TBL_CHAN_DIR gets a list of entries in the ICF, and is part of the TABLE_ROUTINES set of modules.

(11) WRITE_BIG writes a BIG file, and is part of the BIG_ROUTINES set of modules.

2.7 Records and Structures

For a detailed description of DEC's nonstandard implementation of records and structures, see both the VAX FORTRAN Language Reference Manual (Order Number AA-D034E-TE) and the VAX FORTRAN User Manual (Order Number AA-D035E-TE), both dated June 1988. Later editions exist.

If an operator chooses not to delve deeply into records and structures, he or she should think of a structure as a description of a collection of variables (a plan), and a record as the structure's realization in memory. The structure is the architect's plan, while the record is the construction company's building, realizing that other buildings may be built using the same set of plans. The variables described in the structure may be any legal FORTRAN type in any order. In a record, the variables are stored in structure order with no blank space. A record statement assigns a variable name to a structure. A structure description may contain record statements, but the structures named in those record statements must have been previously defined.

The variable name that defines a record statement can be used in much the same way that usual variable names are used, but the name refers to all the elements contained in that record. The structure defined by a record statement may be dimensioned. An individual element in a record is referenced by prefixing its name with the name of each record it is a member of, working outward. Thus the variable name A.B.C(*i*).X is element X of the *i*th record C in record B in record A. There is a difference between a structure and a record, but most authors use the words interchangeably.

2.8 Files Created

FORTRAN logical units 1 and 3 are used in both program modules. Logical unit 2 is used only in REALIZE. For temporary use, logical unit numbers come from the RTL routines LIB$GET_LUN and LIB$FREE_LUN.

TBLLUN (=1) is the logical unit for the ICF file.
BIGLUN (=2) is the logical unit for the BIG file.
IOUTLOG (=3) is the logical unit for the LOG file.

2.8.1 The BIG File

One BIG file is created for all data recovered from the RTDs on a GPIB. BIG file format is described in Appendix E. It contains all necessary records from the ICF and data. ANALYZE only needs a BIG file to create CHN files. BIG files created by REALIZE are named RTDBIGRmn.DAT, where m is the leading digit of the RMV node number, and n is the GPIB bus number.

2.8.2 The LOG File

The file REALIZE.LOG (or INITIALIZE.LOG) is created in the current directory during each execution. This file is opened in ENVIRONMENT, and written in FOUND. FOUND is described below. The LOG file is the first place to begin troubleshooting. It has an execution date/time stamp,

the command line, and the creation date and name of the .EXE file. It tells about any unusual occurrence, and what, if anything, was done about it. Each entry is annotated with the originating module's name. Information about messages in the LOG file is in the Opeartors Manual.

If an error has occurred in the RTD720 communications routines, it is noted with a request to see the LOG file written by those routines.

LOG file messages are sometimes cryptic. While trying to interpret any message, the user should review the module that generated the message for insight into the problem, or item of information.

2.8.3 FOUND, The Error Routine

Module FOUND is the standardized error routine. FOUND is used in all codes, though not always in exactly the same version. A typical call to FOUND looks like this:

```
                ..... from COMMAND_LINE.FOR .....

      do i=1,num_cl_chans
            if(cl_chans(i) .lt. 1 .or.
     1         cl_chans(i) .gt. 4) then
                  ij=ij+1
                  call found(.false.,0,'COMMAND LINE',2,
     1                  16,'Command line channel '
     2                  //'had bad value, setting '
     3                  //'to -1, was ',cl_chans(i),2)
                  cl_chans(i)=-1
                  endif
            enddo

            ..... end code fragment .....
```

Explanation: The /CHANNEL qualifier was used, the number of channels was called out, and num_cl_chans was not zero. The RTD hardware only permits channel numbers in the range of 1 to 4 inclusive. This loop checks each channel number, and counts and redefines all invalid channel numbers. It also writes a message about each bad channel number to the LOG file.

Note the use of the concatenation symbol "//" to extend the sixth argument. The significance of each of the eight arguments is (in order):

(1) .false. a logical and if .true. then argument 2 is meaningful.

(2)	0	the status value returned by a system routine. If this argument is meaningful, LIB$SIGNAL is called, and the message is added to the LOG file.
(3)	'COMMAND LINE'	a string containing the name of the calling module.
(4)	2	an integer in the range 1 to 3 that defines the error level written to the LOG file, and controls some module logic. 1 = = > INFORMATIVE 2 = = > NON-FATAL ERROR 3 = = > FATAL ERROR
(5)	16	an integer in the range 1 to 17, and an index into a set of canned messages. They are defined in the source code. In this case the message is: For your information.
(6)	Command ... was	the specific message about what was found, and what, if anything, was done about it. Notice here how it directs the operator to the next argument.
(7)	cl_chans(i)	an integer that may or may not be meaningful. In this case, it is the offending channel number.
(8)	2	an integer value in the range of 1 to 9 that indicates the programmer's sense of the severity of the message. One is usually fatal, and 9 is usually informative. Here, the user will not get what was requested, because that request was not understood. This value is put into the SLOG tag. See Appendix G.

Because every error is in the log file, and each is tagged with the name of the originating program, all the possible errors are not listed here. The DEC utility program SEARCH is helpful in this task.

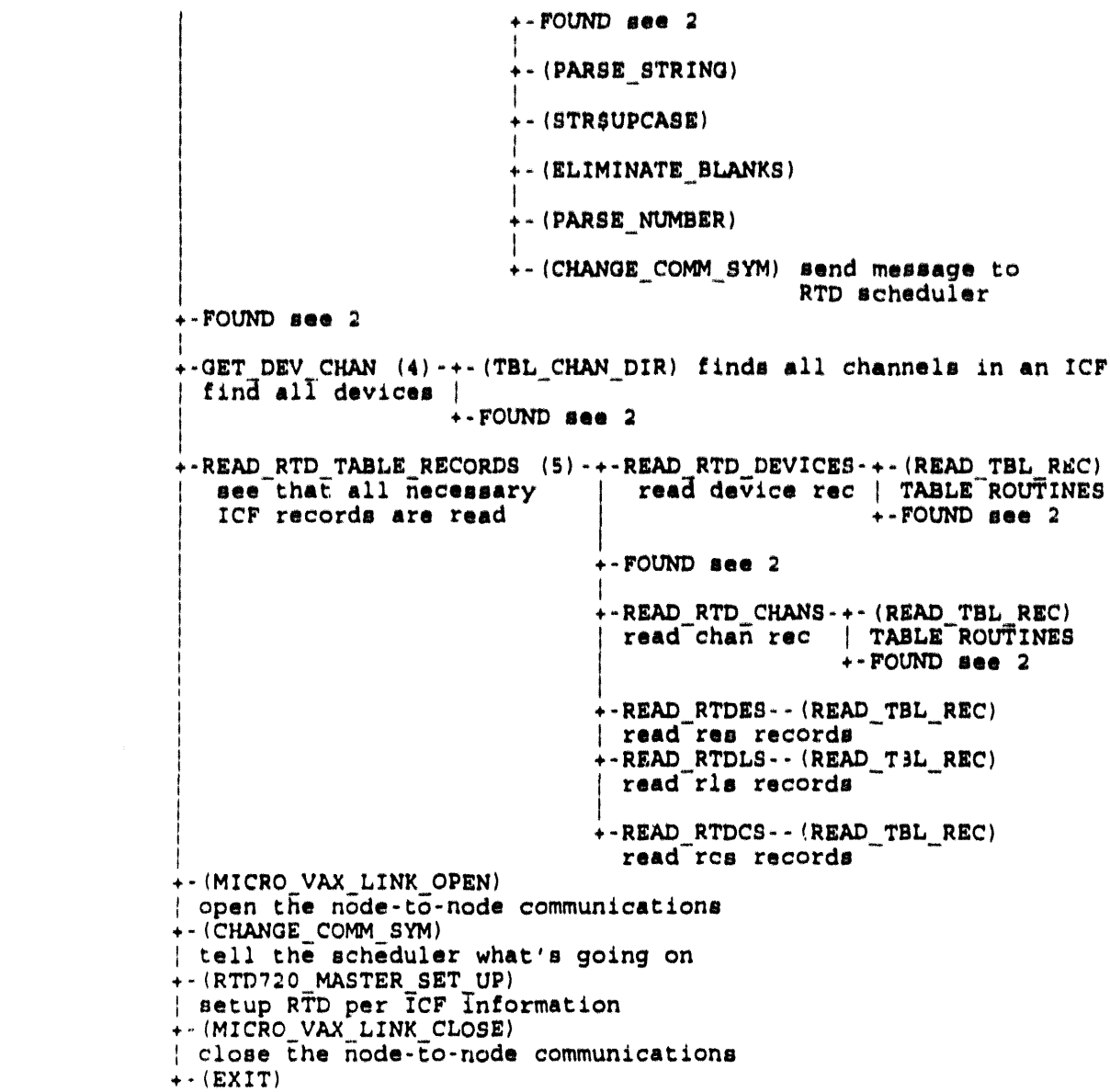
For further information on the LOG file format required by SLOG, see Appendix G.

2.9 The Call Tree

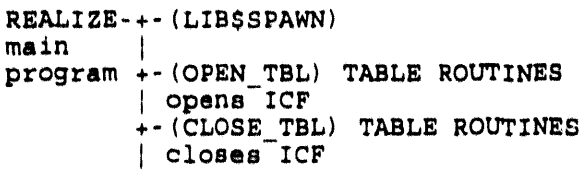
The following call tree is from the program FLINT (version 2.71). At least the first occurrence of each routine has been annotated. Those routines that appear as (name) are external to REALIZE. Those with names followed by (*n*) are referenced later as "see *n*".

This is a primary tree starting at the program INITIALIZE:

```
INITIALIZE-+-(OPEN_TBL) opens an ICF
main prog. |
           +-(CLOSE_TBL) closes an ICF
           |
           +-(READ_TBL_REC) reads an ICF record
           |
           +-ENVIRONMENT (1)-+-(LIB$GETJPI)
           | who,what,when   |
           | where and why   +-(LIB$SYS_TRNLOG)
           |                 |
           |                 +-(LIB$FIND_FILE)
           |                 |
           |                 +-(LIB$FIND_FILE_END)
           |                 |
           |                 +-(FILE_DATE) find executable name
           |                 |             and date
           |                 |
           |                 +-(LIB$DATE_TIME)
           |                 |
           |                 +-FOUND (2)-+-(LIB$SYS_GETMSG)
           |                 | writes    |
           |                 | log file  +-(CHANGE_COMM_SYM) RTD
           |                 |           |                   Scheduler
           |                 |           |                   communications
           |                 |           +-(MICRO_VAX_LINK_CLOSE)
           |                 |           | break the node-to-node
           |                 |           | link.
           |                 |           +-(LIB$SIGNAL)
           |                 |           |
           |                 |           +-(LIB$ENABLE_CTRL)
           |                 |           |
           |                 |           +-(EXIT) auf wiedersehen
           |                 |
           |                 |
           |                 +-(ELIMINATE_BLANKS) trim lead and
           |                                      trail blanks
           |
           +-INITIAL_COMMAND_LINE-+-PARSE_PETE_NUM_LST (3)-+-(CNV_NUM_LST_TBL)
           | crack the command    | extract numbers from   | convert list
           |   line               | list                   +-(PARSE_STRING)
           |                      |                          find string on
           |                      +-(CLI)                    command line
           |                      | get command line, write
           |                      | to LOG file
```



This is a primary tree starting at the program REALIZE:



```
+-(READ_TBL_REC) TABLE ROUTINES
| reads records from ICF
+-(CLOSE_BIG) BIG ROUTINES
| close BIG file
+-(OPEN_BIG) BIG ROUTINES
| opens BIG file
+-ENVIRONMENT see 1
| who, what, when, where, and why
+-COMMAND_LINE-+-PARSE_PETE_NUM_LST see 3
| crack the    | get numbers from string of numbers
| command line +-(CLI)
|              | get and write command line to LOG file
|              +-FOUND see 2
|              | the LOG file write routine
|              +-(PARSE_STRING)
|              | gets command line string
|              +-(STR$UPCASE)
|              |
|              +-(ELIMINATE_BLANKS)
|              | trim lead and trail blanks
|              +-(PARSE_NUMBER)
|              | get number from command line
|              |
|              +-(CHANGE_COMM_SYM)
|              | tell scheduler what's happening
|              +-(PARSE_PRESENCE)
|                find out if qualifier exists
+-FOUND see 2
|
+-GET_DEV_CHAN see 4
| determine which devices and channels to get data from
+-READ_RTD_TABLE_RECORDS see 5
| read the necessary ICF records
+-(MICRO_VAX_LINK_OPEN)
| open the node-to-node communications
+-(RTD720_MASTER_ACQUISITION)
| verify triggers
+-(CHANGE_COMM_SYM)
| tell scheduler what's happening
+-(RTD720_MASTER_PULL_DATA)
| recover the data from an RTD channel
+-WRITE_BIG_FILE-+-(WRITE_BIG) BIG ROUTINES
| write data to  |
| BIG file       +-FOUND see 2
|
+-(MICRO_VAX_LINK_CLOSE)
| close node-to-node communications
+-(ELIMINATE_BLANKS)
|
+-(EXIT) job is complete
```

3.0 REALIZE AND INITIALIZE INTERFACE MANUAL

3.1 Purpose

This maintenance document describes the software interface with (1) INITIALIZE, (2) REALIZE, (3) task-to-task communication, and (4) RTD720_DRIVER that manages Tektronix RTD720 digitizers (RTD) using Digital Equipment Corporation (DEC) VAX computers and DEC VLV4000 workstations.

3.2 Environment

The computers, laser printers (plotting), and RTDs are installed in SNL building 12-909 at the NTS. For information on the computer/instrument network and system hardware, see Volume 1, Section 2: RTD VAX Network. Note that all programs were written in VAX FORTRAN 5.0 for DEC VMS systems.

3.3 Functions

The functions of the programs described by this document are grouped into three functional areas: (1) programs interfacing with INITIALIZE and REALIZE; (2) simple programs illustrating task-to-task communication; and (3) DEC VLC400 interface to RTD720.

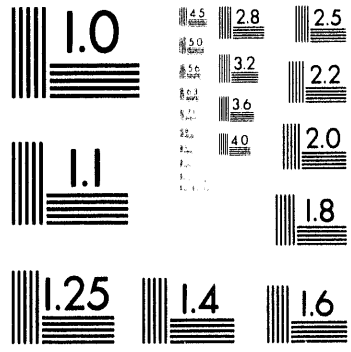
3.3.1 Interface with INITIALIZE and REALIZE

Micro_vax_link_open

This subroutine (1) creates and opens the log file with a name format of the node, date, and time (e.g., RTD720_RMV10_18JUN92_132421.log) and (2) opens the communication (task-to-task) between a VAX computer and a DEC VLC4000 workstation.

Micro_vax_link_close

This subroutine (1) closes the log file and (2) closes the communication (task-to-task) between a VAX computer and a DEC VLC4000 workstation.


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6

2 of 5

RTD720_master_acquisition

This subroutine handles the communication between the DEC VAX and any one of several DEC VLC4000 workstations that have RTD720s attached to their GPIB bus. In addition, this software concerns itself with determining which devices have acquired data.

Additional information is presented in the source code detailing how a specific RTD720 either did or did not acquire data and how it is handled.

RTD720_master_set_up

This subroutine is called by INITIALIZE for one RTD720 at a time. It passes data from the Instrument Control File (ICF) to the program RTD720_slave_enable.

RTD720_master_pull_data

This subroutine handles the communication between the DEC VAX and any one of several DEC VLC4000 workstations that have RTD720s attached to their GPIB bus. In addition, this software concerns itself with pulling only raw data for one RTD720 at a time when invoked by the program REALIZE; and it passes information to program RTD720_slave_ pull_data.

Additional information is presented in the source code detailing how a specific RTD720 either did or did not acquire data and how it was handled.

3.3.2 Task-to-Task Communication

The following sections illustrate task-to-task communication between programs on different nodes (computers).

Controlling

The following source listing of a FORTRAN program illustrates the basics of task-to-task communication between two programs, each on a different node (computer).

```
* Program Controlling

* This is a simple program to illustrate task-to-task
* communication.
*
```

```
* The players are:
*
*Controlling - This program is the main driving module
*    that sends messages (Arrays, Structures,
*    variables) to the program SERVANT executing
*    on a different node (computer). In addition,
*    can receive messages back from SERVANT.
*
*Servant - This program receives and can send messages
*back to the program CONTROLLING
*
*DEMO - This the .COM that resides on the computer
*node that CONTROLLING calls
*
*This example is designed to send only one message to SERVANT
*and receive one message back from SERVANT.
*
*Opening the Task to Task (Link_lun) only occurs once in a *session.
*The same applies to the close (Link_lun)
*
*Messages can be sent back and forth as many times as necessary.
*
*Reminder: For each message sent to SERVANT from
*CONTROLLING (a write statement), there must be a
*corresponding read statement in SERVANT and vice versa.
*


*The messages (Arguments) passed back and forth can be
*different each time, but the corresponding write/read
*pairs between programs must be identical.
*

*Additional information on Task-to-Task can be found:
*
*  -DECnet-VAX Guide 2-12
*      -Networking 1-3, 1-21, 8-1, 8-16, and 8-25
*      -Fortran example - Networking 8-44

      integer*4    LIB$get_lun
      integer*4    Status
```

```
      integer*4     Istat
      integer*4     Link_lun
      character*20  Buffer_1
      character*75  Buffer_2
      character*60  File_name
*
*     ------------------------------------
*     Open the communication (Task to Task)
*     ------------------------------------

      Status = LIB$get_lun (Link_lun)

      File_name = RMV40::"TASK=DEMO"

      Open (Link_lun,
     -      file=File_name,
     -      status=OLD,
     -      IOstat=Istat,
     -      form=UNFORMATTED)

      If (Istat .ne. 0) Stop

*     ---------------------------------
*     Now send a message to another Node
*     ---------------------------------

      Buffer_1 = ' '

      Buffer_1  = 'ET call home now Mom'

      Write (*,'(1x,a)') Buffer_1

      Write (Link_lun,IOstat=Istat) Buffer_1

      If (Istat .ne. 0) Stop

*     ---------------------------------------
*     Now read a message from the another Node
*     ---------------------------------------

      Buffer_2 = ' '
```

```
      Read  (link_lun,IOstat=Istat) Buffer_2

      Write (*,'(1x,a)') Buffer_2

*     ------------------------------------
*     Close the communication (Task to Task)
*     ------------------------------------

      Close (Link_lun,IOstat=Istat)

      End
*               ------------------------------
*               Physical End of Program Source
*               ------------------------------
```

Servant

The following source listing of a FORTRAN program is the servant program in the task-to-task communication between two programs, each on a different node (computer).

```
*
      Program Servant

*     This is a simple program to illustrate task-to-task communication.
*
*        Servant    - This program receives and can send messages
*                     back to the program CONTROLLING
*
*     This example is designed to receive only one message from CONTROLLING
*     and send one message back to CONTROLLING.
*
*     Opening the Task to Task (SYSNET_lun) only occurs once in a session.
*     The same applies to the close (SYSNET_lun)
*
```

```
      integer*4     LIB$get_lun
      integer*4     Istat
      integer*4     SYSNET_lun
      character*20  Buffer_1
      character*75  Buffer_2
*
* ------------------------------------------------
* Open the link (Task to Task) back to the calling task
* ------------------------------------------------

        Status = LIB$get_lun (SYSNET_lun)

        Open (SYSNET_lun,
     -    file='SYS$NET',
     -    status='old',
     -    IOstat=Istat,
     -    form='unformatted')

      Read (SYSNET_lun,IOstat=Istat) Buffer_1

      If (Istat .ne. 0) Stop

      Buffer_2 = ' '

      Buffer_2 (1:57) =

     - 'But Mom do I have to come home? I am still having fun ET'

      Write (SYSNET_lun,IOstat=Istat) Buffer_2

      If (Istat .ne. 0) Stop

      Close (SYSNET_lun,IOstat=Istat)

      If (Istat .ne. 0) Stop

      END

*           ------------------------------
*           Physical End of Program Source
*           ------------------------------
```

Demo.com

The following source listing of a DEC VAX .COM written in DEC Command Language (DCL) is necessary to allow task-to-task communication between two programs, each on a different node (computer).

```
$!
$!      This .COM file illustrates what is necessary to set up
$!      task-to-task communication between two DEC VAX computers
$!
$ run ABQVAX::UC:[rcaudell.Task_to_Task]SERVANT
$!
$ exit
```

3.3.3 DEC VLC400 Interface to RTD720

RTD720_driver

Seven major functions are performed by this program:

(1) Opens the link back to the calling host for task-to-task communication between two computer nodes the first time the program executes.

(2) Obtains the name of the node where RTD720 is currently executing; creates and opens the log file with a name format of the node, date, and time (e.g., RTD720_RMV10_18JUN92_13241.log); and sets up communication to the GPIB interface.

(3) If the request is INIT, attempts to initialize one specific RTD720 based on information from the ICF by calling the program RTD720_slave_set-up.

(4) If the request is ACQU, finds out which RTD720 has completed data acquisition (i.e., an event code of 465 is found in the RTD event stack by calling RTD720_slave_acquisition) and passes this information book to REALIZE.

(5) If the request is PULL, attempts to pull digitized data from a specific RTD by calling the program RTD720_slave_pull_data and passing the data (if found) back to REALIZE for BIG file construction.

(6) If the request is PARM, passes back the PREAMBLE (data header) from the specific RTD if the PULL function 5 was successful to REALIZE for BIG file construction.

(7) If the request is EXIT, closes the log file and closes the task-to-task communication on its respective node.

RTD720_set_up_communication

This program (1) de-assigns the assigned channel that handles communication, the VAX's IEx communication port, and the RTD720 programs, if it is not the first time through in this session, and (2) sets up the necessary communication between the DEC VLC 4000 workstation/EKx communication port attached and the RTDs. Refer to the source code for additional information.

RTD720_slave_acquisition

This subroutine handles the communication between the VLC 4000 workstation (RMV10, etc.) and any one of several RTD720s attached to its GPIB bus. It polls each requested RTD720 address passed to it from RTD720_master_acquisition to see if it has received an EVENT code 465. This means that the RTD720 has completed acquiring and digitizing data.

RTD720_slave_set_up

This program handles the communication between the DEC VLC 4000 workstation (RMV10, etc.) and any one of several RTDs attached to its GPIB bus. It takes information passed to it from the ICF. This sets up an RTD for Dry Run/Shot configuration by translating the information into ASCII strings (the language of the RTD) and sending the commands through the GPIB bus to the RTD.

RTD720_slave_pull_data

This program gets data from a specific RTD. The functions of this program are to retrieve data from an RTD, if it has completed its acquisition cycle, and to retrieve the Preamble.

DEC_GPIB_QIOW_subs

This file is a collection of several subroutines used to access the DEC GPIB bus and general devices attached to the bus. Subroutines are: (1) Initialize the GPIB, (2) Set GPIB timeout, (3) Listen, (4) Unlisten, (5) Talk, (6) Untalk, (7) Clear, (8) Serial poll, (9) Recognize event, (10) Set event, (11) Remote enable, (12) Write to a device, (13) Read from a device (ASCII), and (14) Read from a device (Byte).

IOSB_error

This standard error-handling routine formats and prints the I/O status block (IOSB) from an error detected during a queue I/O request and wait (QIOW) request.

In addition, it calls system routine to translate the IOSB(1) code into a readable description.

The QIOW service queues an I/O request to a channel associated with a specific device. For a complete definition, refer to the VAX/VMS System Service Reference Manual, April 1986, Software version: 4.4, pages SYS-277 through SYS-282.

The IOSB receives the final completion status of the I/O operation. It has three fields: (1) condition value, (2) transfer count, and (3) device-specific information.

Find_node

This subroutine finds the computer node the program RTD720 is currently running by using a system call to lib$sys_trnlog.

Convert_error_code

This subroutine converts an error code into a readable message using a system call to Lib$sys_getmsg.

3.4 Commands Procedures

The programs that interface with INITIALIZE and REALIZE are compiled and linked differently. Refer to the section on compiling programs into INITIALIZE and REALIZE. The programs involved are mentioned in section 5.1 of this document.

3.4.1 Compile and Link RTD720_Driver Only

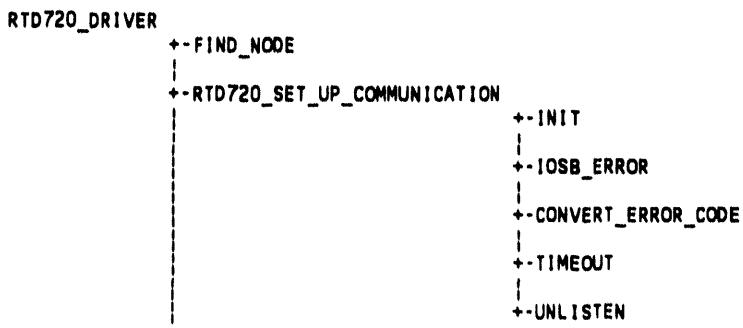
```
$!
$!              ------------------------------
$!              File name: RTD720_driver_cl.com
$!              ------------------------------
$!
$  set verify
$
$  Del  RTD720_driver.exe.*
$
$  For   RTD720_driver
$  For   RTD720_slave_acquisition
$  For   RTD720_slave_pull_data
$  For   RTD720_slave_set_up
$  For   RTD720_set_up_communication
$!
$  Link RTD720_driver,-
       RTD720_slave_acquisition,-
       RTD720_slave_pull_data,-
       RTD720_slave_set_up,-
       RTD720_set_up_communication,-
       Find_node,-
       Convert_error_code,-
       DEC_GPIB_QIOW_subs,-
       IOSB_error,-
       str_length
$!
$  set noverify
$!                    ----------------
$!                    End of .COM file
$!                    ----------------
```

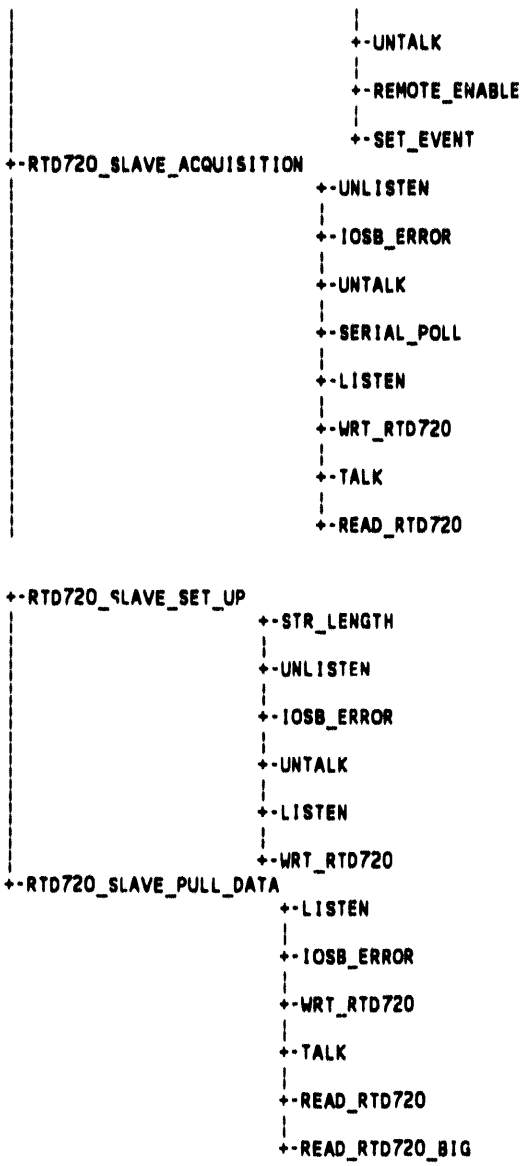
3.4.2 Link RTD720_Driver Only

```
$!
$!              ----------------------------------------
$!              File name: RTD720_driver_link_only.com
$!              ----------------------------------------
$!
$  set verify
$
$  Del  RTD720_driver.exe.*
$
$  Link RTD720_driver,-
       RTD720_slave_acquisition,-
       RTD720_slave_pull_data,-
       RTD720_slave_set_up,-
       RTD720_set_up_communication,-
       Find_node,-
       Convert_error_code,-
       DEC_GPIB_QIOW_subs,-
       IOSB_error,-
       str_length
$
$  set noverify

$!                  ----------------
$!                  End of .COM file
$!                  ----------------
```

3.5 RTD720_Driver Program Call Tree




+-UNTALK
+-REMOTE_ENABLE
+-SET_EVENT
+-RTD720_SLAVE_ACQUISITION
+-UNLISTEN
+-IOSB_ERROR
+-UNTALK
+-SERIAL_POLL
+-LISTEN
+-WRT_RTD720
+-TALK
+-READ_RTD720
+-RTD720_SLAVE_SET_UP
+-STR_LENGTH
+-UNLISTEN
+-IOSB_ERROR
+-UNTALK
+-LISTEN
+-WRT_RTD720
+-RTD720_SLAVE_PULL_DATA
+-LISTEN
+-IOSB_ERROR
+-WRT_RTD720
+-TALK
+-READ_RTD720
+-READ_RTD720_BIG

4.0 OVERVIEW OF ANALYZE

4.1 Introduction

Starting with the DIAMOND FORTUNE test, the NTS Instrumentation Development Department (9321) has developed a computer architecture to acquire and process RTD720 high-speed digital data. Within this architecture, the REALIZE program acquires data from RTD720 devices and stores it together with the appropriate header information in a file called the BIG file. The ANALYZE program then separates the contents of the BIG file into individual channel (or subchannel) files with the appropriate header information. These channel (CHN) files can then be plotted using the PRELEWD program.

4.2 Utilities

The following utility programs are used by ANALYZE to perform the functions indicated. The sources for these routines are located in the LD:[UTILITY] directory. The object modules are contained in UTILITY.OLB in the LD:[UTILITY] directory.

(1) PARSE_STRING.FOR -- parses the command line for the indicated qualifiers and returns the value of the qualifier as a character string.

(2) CNV_NUM_LST_TBL.FOR -- converts a numeric list contained in a character string to an array of integers. The number of integer values passed back in the array is returned as the function value.

(3) CHANGE_COMM_SYM.FOR -- opens and/or adds a comment line to a file used by the RTD_SCHEDULER to keep track of the status of programs INITIALIZE, REALIZE, ANALYZE, and PRELEWD.

(4) OPEN_BIG.FOR -- opens the BIG file which is the input file to ANALYZE. See BIG_ROUTINES.DOC for complete user description.

(5) BIG_DIR.FOR -- returns a directory of channel numbers contained in the BIG file. See BIG_ROUTINES.DOC for complete user description.

(6) READ_BIG_HDR.FOR -- retrieves the header information for a channel from the BIG file and stores the contents in a structure provided by the user. See BIG_ROUTINES.DOC for complete user description.

(7) READ_BIG.FOR -- reads a specified number of bytes of data from the BIG file and stores them in a user-supplied array. See BIG_ROUTINES.DOC for complete description.

(8) CLOSE_BIG.FOR -- closes the BIG file when we are done with it. See BIG_ROUTINES.DOC for complete description.

4.3 Data Structures/Data Base

The information needed to process the data for a given channel of information is contained in an INGRES-type data base. The ANALYZE program does not have access to this data base. The REALIZE program that creates the BIG file does have access to the data base and inserts the necessary information to process the BIG file data in the header portion of the BIG file. The structure that defines the header contents may be found in BIG_STRUCTS.DEF and TABLE_STRUCTS.DEF. TABLE_STRUCTS.DEF describes the data base structure and BIG_STRUCTS.DEF defines those portions of this structure applicable to data processing. For a description of how to use structures in VAX FORTRAN see reference 8. The header structure for the output channel (and subchannel) files written by ANALYZE is defined in CHN_HEADER.DEF. This structure normally includes that portion of the BIG file header applicable to plotting and other signal processing techniques that follow ANALYZE plus certain parameters created by ANALYZE that are not in the BIG file header.

4.4 Common Parameters

There are three common parameters that are used by ANALYZE and other programs associated with data acquisition and processing of RTD720 data. They are:

1. RTD_EXP_SUB_MAX -- the maximum number of RTD720 subchannels when the run type is Signal Dry Run.

2. RTD_LC_SUB_MAX -- the maximum RTD720 subchannels when the run type is Laser Calibration.

3. RTD_CC_SUB_MAX -- the maximum RTD720 subchannels when the run type is Cable Compensation.

These parameters are declared in a document called BIG_STRUCTS.PRM and ANALYZE gains access to them through an include file in BIG_STRUCTS.DEF.

4.5 Input Parameters

Most input parameters to ANALYZE are obtained from the channel information provided in the CHNBIG header. The user must supply the name of the BIG file on the command line.

There are two parameters which may be entered as an argument on the command line. They are

> /CHANNEL=n or /CHANNEL="n1,n2,n3,etc." or /CHANNEL="n1-nn" to select one or more specific channels in a group to be processed in this run. Note: CHANNEL may be shortened to CH.
>
> /SUBCHAN=n or /SUBCHAN="n1,n2,n3,etc." or /SUBCHAN="n1-nn" to select one or more specific sub-channels on one RTD channel processed in this run. Note: SUBCHAN may be shortened to SUB.

Definitions for command line parameters are contained in the file ANALYZE_COMMAND_DEF.CLD which is located in LD:[ANALYZE].

4.6 Program Execution

An example of the simplest case for running ANALYZE is as follows:

```
$ ANALYZE RTDBIGG1011.DAT
```

ANALYZE will obtain the source and recorder codes from the BIG file name.

The next three commands are examples of channel selection from within the total group in a BIG file. The first example is for one channel, the second for several selected channels, and the last for a range of channels.

```
$ ANALYZE/CHANNEL=2 RTDBIGG1011.DAT
$ ANALYZE/CHANNEL="3,7,12,17" RTDBIGG1011.DAT
$ ANALYZE/CHANNEL="3-14" RTDBIGG1011.DAT
```

The following command shows how subchannels may be selected from one specified channel.

```
$ ANALYZE/CHANNEL=2/SUB=1,3 RTDBIGG1011.DAT
```

4.7 Output Files

In the data base definitions of RTD720 data channels there will always be at least one subchannel defined, even if it includes the entire file. In the output there will always be a combined raw data file plus a file for each subchannel. Output file designation is CHN*snnrmmm-kk*.DAT for the combined raw data and for each subchannel. The combined raw data will be identified as subchannel "00".

In the examples above:

snn = source code
r = recorder code
mmm = channel number - decimal
kk = subchannel number - decimal

4.8 Sequence of Operation

The ANALYZE program performs its function of creating channel and sub-channel files from the BIG file created by the REALIZE program in the following sequence:

1. Initialize program variables.

2. Open a log file ANALYZE.LOG for informative messages and diagnostics.

3. Parse the command line for BIG file name and other processing parameters.

4. Extract source and collection point parameters from the BIG file name to use in creating CHN file names.

5. Open the BIG file using the OPEN_BIG utility.

6. Write header information into the log file.

7. Get a directory of data channels in the BIG file using the BIG_DIR utility.

8. For each data channel in the directory, perform the following sequence of operations (unless specific channels have been selected on the command line):

    a. Get the header information for the channel using the READ_BIG_HEADER utility.

    b. Read in a buffer of information from the BIG file for the channel using the READ_BIG utility.

c. Call the PROC_RTD720 subroutine that creates the desired channel and subchannel files based upon data base information contained in the header.

9. Notify the RTD Scheduler that ANALYZE has completed its processing of the BIG file.

10 Close the BIG file using the CLOSE_BIG utility.

4.9 PROC_RTD720 Sequence of Operation

When the PROC_RTD720 subroutine is called, it performs the following sequence of operations for each prescribed data channel:

1. Convert the input data from a byte array to a 16-bit integer array.

2. Obtain a list of applicable subchannels from data base information in the header.

3. Compare the number of subchannel names so obtained with the number of subchannels specified in the data base.

   a. If these numbers do not agree, only create the "00" subchannel which contains all the data for the channel.

4. For each subchannel defined in the data base, the following steps are performed in order to create files based upon the boundaries defined by EXPMT_LEFT and EXPMT_RIGHT parameters in the data base.

   a. Open the designated subchannel file.

   b. Write the applicable portion of the channel information from the BIG file into the subchannel file.

   c. Write the appropriate header information at the beginning of the subchannel file.

   d. Close the subchannel file.

4.10 Compile/Link Instructions

The ANALYZE program may be compiled using the standard FORTRAN statement described in reference 8. To link the resulting OBJ files with the required utility routines, the command procedure LINK_ANALYZE.COM may be used. Its contents are as follows:

```
$ LINK/EXE=ANALYZE ANALYZE, - LD:[UTILITY]UTILITY/LIB
```

4.11 Diagnostics

ANALYZE generates a file called ANALYZE.LOG. For each data channel within the BIG file, ANALYZE displays the messages:

(1) PROCESSING RTD CHANNEL NO. xxx
(2) PROCESSING COMPLETE ON CHANNEL xxx NO SAMPLES PROCESSED: xxxxx

Between these two messages it may display discrepancies applicable to the data channel encountered during processing.

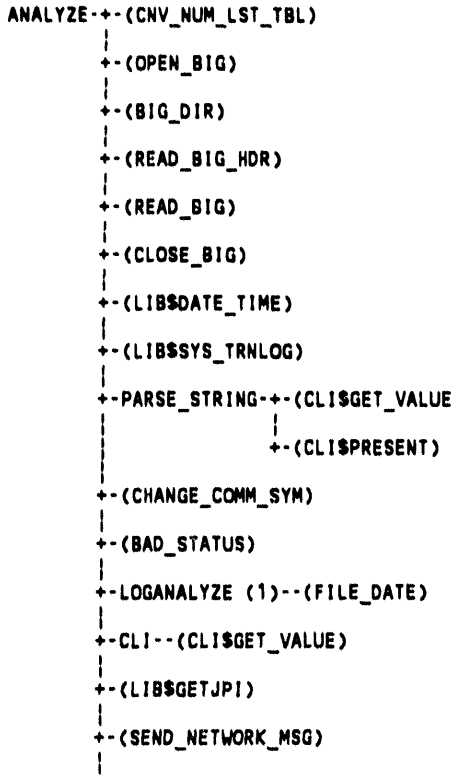
The following is a listing of all diagnostics together with a description of the meaning. The first diagnostic is generated in the main routine (ANALYZE). The rest are generated in PROC_RTD720.

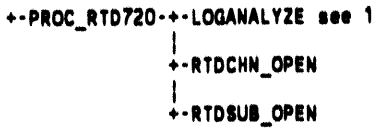
DIAGNOSTIC	DISCUSSION
1. NO DATA FOR THIS CHANNEL	A byte count of 0 was obtained when an attempt was made to read from the RTDBIG file for this memory channel.
2. COULD NOT FIND ALL RTD720 SUB-CHANNELS	The number of subchannels in one section of the data base does not agree with the actual subchannel numbers from another part of the data base. The ANALYZE program, therefore, cannot properly identify output CHN files. The data base administrator should be notified immediately if this discrepancy occurs.
3. SELECTED SUB-CHANNEL NO. XX DOES NOT EXIST.	The subchannel selected on the command line could not be found on the list of acceptable channels from the data base.

DIAGNOSTIC	DISCUSSION
4. TRACE_LEFT IS ZERO IN TABLES CHANGED TO ONE BY ANALYZE	It has been established by convention that the first data point in an array will have an index of 1, not 0. Zero, therefore, would be out of bounds in this convention.
5. NOTE! EXPMT_RIGHT IN TABLES = XX TOTAL SAMPLES = YY	The count associated with experiments in the experiments in the data base is greater than the number of points in the BIG file.
6. ***** DATA IN SUB-CHAN XX AT BAND EDGE	Data values of either 0 or 255 were observed in the input data file indicating that the actual value could have been out of range.

4.12 Flint Tree

This is a primary tree starting at the program ANALYZE.




+-PROC_RTD720-+-LOGANALYZE see 1
|
+-RTDCHN_OPEN
|
+-RTDSUB_OPEN

5.0 PRELEWD MAINTENANCE INFORMATION

5.1 Introduction

PRELEWD is a first-look plotting program for use with the NTS Instrumentation System suite of codes. It reads a CHN file and generates a hard copy plot for each channel as a part of the run sequence. Because it operates in BATCH mode during the run sequence, it is very fault tolerant. PRELEWD can plot on a limited set of graphic terminals, and is, therefore widely used by our customers. It does not have any analysis capability, but there are a limited number of plot formatting instructions available through the ICF. The underlying graphics package is GRAFPAK-GKS.

The name PRELEWD, pronounced prey-lude, and often misspelled PRELUDE, came about in the following way. It was necessary to have a new name to enable users to differentiate it from PLOT, the earlier program with similar goals. The programmer arrived at "Logical Examination With Display" for the plotting portion of the code, which was easy to write. The problem arose in trying to get a reasonable description of the file header and data being passed. The entire instrumentation system was in a state of flux — even experienced people were having trouble with code interfaces. Plotting was the last link in a chain, being forced to wait till other earlier links had been forged before its interface could be formalized. Because of this stressful wait, the programmer added PRE to show the need to know what was coming in.

5.2 Overview

This is not a maintenance manual in the sense that it lists troubleshooting procedures, or instructions for adding undocumented features. Maintenance information is often obtained from the source, which contains comments for guidance. This document, discusses PRELEWD and the routines that perform specific tasks.

This documentation is directed to users familiar with plotting, the Run Time Library, and DEC's FORTRAN, particularly DEC's version of structures and records. A knowledge of the hardware is helpful but not essential. The documentation assumes the user has read and is familiar with "Section 6.0, "Using PRELEWD, with GRAFPAK Comments," in the Operators Manual. It also assumes that the user has run PRELEWD.

5.3 General Description

PRELEWD cannot be viewed as a single entity, because it is closely tied to SNL's NTS Instrumentation System, both the hardware and the software. A change in any part of this system

may affect PRELEWD. It uses CHN files prepared either by PROCESS, which used BIG files prepared by FETCH/DECOM, which decommed data from hardware, or by ANALYZE from BIG files, which were prepared by REALIZE from hardware. The NTS Instrumentation System is a sequence of programs that work, across an extensive network of special and general purpose computers, with special hardware to record and to display short-lived phenomena captured from a hostile environment. The data are received by one of several types of source devices. Source devices such as the SANDUS can record many channels of data, while other source devices can only record one channel. These channel data are then multiplexed with data from other channels and sent to a remote location where the data are recorded in several possible ways: in a memory bank (Mass Memory, MM), on magnetic tape (High Density Digital Recorder, HDDR), or in memory on the source device itself (RTD720). Once captured, there are several programs required. They are FETCH, DECOM, PROCESS, and PRELEWD. FETCH demultiplexes data from the HDDR into BIG files containing one or more channels. DECOM reformats the data from MM into BIG files containing one or more channels. PROCESS separates these files into individual channel files called CHN files. More recently, with the advent of newer digitizers, which have their own memory (the RTD720 specifically), a new sequence of programs has emerged. REALIZE recovers data from the RTD720s, and packs it into BIG files of one or more channels which ANALYZE reformats into individual CHN files.

PRELEWD processes the CHN files, from whatever source, for plotting, and optionally reformats the data into ASCII files for possible release to the user. Such files are called SRAD files, where SRAD is the philosophy for the files. See Section 5.4.5.

5.3.1 PRELEWD Programming Constraints

PRELEWD's main purpose is to create first-look hard copy plots from an increasing variety of recording devices as soon as possible after an event. Because the NTS Instrumentation System is a local area network, PRELEWD must not have limitations, other than VMS security limitations as to where the CHN file comes from, or where output files are to go.

Secondary purposes have been assigned: They are terminal viewing of data, and creation of secondary files in a less machine-dependent format for data transfer to others. These goals have driven PRELEWD's development philosophy in many ways. Unless the situation is hopeless, PRELEWD attempts to create graphic output. There is minimal interactive dialogue, none if the command line facilities are used. Plotting options are minimal, and very little data manipulation is allowed. The user has complete freedom in where the CHN files come from, and in where the graphics output goes, but not in the choice of output devices.

5.4 Suggested Reading

The following list of references is specific to the NTS Instrumentation System, GKS, or to PRELEWD. It does not include references such as the VAX/VMS Documentation Set, or the VAX/VMS FORTRAN Reference Manual, and User's Manual. It is strongly suggested that the user read the INCLUDE files, because they contain extensive comments.

Using PRELEWD, with GRAFPAK Comments - Section 6 of the Operators Manual

It contains a useful description of PRELEWD and of the command line qualifiers and parameters available. It contains a series of definitions, and talks about the symbols INCLUDE$INCLUDE and PRELEWD$INCLUDE which allow the maintenance programmer to change to a new set of include files without having to modify every FORTRAN source module.

Channel Data File Format - Appendix D

This is documentation on the CHN file, with a short section on structures and records. Understanding the CHN file will aid in understanding both PRELEWD and the two programs which generate files for PRELEWD, PROCESS, and ANALYZE.

5.4.1 Structure Definition Files

While there are a number of structures defining files used throughout the NTS Instrumentation System, only two are applicable to PRELEWD. Both of these files usually reside in the directory_name INCLUDE$INCLUDE, which is defined by the following:

$ DEFINE INCLUDE$INCLUDE LD:[INCLUDE] (or wherever they are located)

The two files are listed below.

TABLE_STRUCTS.DEF is one of the important files containing structure definitions needed to understand the CHN file. It contains structure definitions for all but the CAL_... structures and the CHN file header.

CHN_HEADER.DEF is the other important file for understanding CHN files. It contains CAL structure definitions, and defines all the records used in PRELEWD.

5.4.2 GKS Information

GKS is an acronym for Graphics Kernel System, and is an ISO standard. ATC's* implementation of GKS is called GRAFPAK, and was used because it has been certified to conform to the standard. This makes it easier to include a new plotting surface in PRELEWD without having to write extensive code.

GRAFPAK is the name for the shared library of subroutines that creates the graphics, and may not be distributed outside the 9320 network without a valid license. GRAFPAK must exist on any computer on the 9320 network where PRELEWD runs. All logicals necessary to use GRAFPAK are defined system wide.

GKS PRIMER (Lucia McKay; Nova Graphics International, 1984) introduces the ideas used in GKS. This book discusses the basics and is not lengthy, although similar references may exist that are more recently published.

Computer Graphics Programming: GKS - The Graphics Standard (Enderle, Kansy, and Pfaff; Springer-Verlag, 1987) is a guide to GKS programming and the standard. Later editions exist.

GRAFPAK-GKS Users Guide (Advanced Technology Center). This is a description of the GRAFPAK-GKS package used in PRELEWD. It contains some explanatory information. It should also be used in conjunction with the specific workstation guide for the workstation(s) intended for use. These are supplied by ATC.

GRAFPAK GKS: FORTRAN Reference Manual (Advanced Technology Center). This is a description of the calls available from FORTRAN in GRAFPAK, their arguments, and meaning. GRAFPAK calls are specially noted in the source modules, and are listed in Section 5.8.2. A GRAFPAK C Manual is also available but no source was written in C.

5.4.3 INCLUDE Files Containing Common Blocks

The next group of files is usually maintained in the directory_name PRELEWD$INCLUDE, which is

$ DEFINE PRELEWD$INCLUDE LD:[PRELEWD] (or wherever the file is located).

---

* Advanced Technology Center, 22982 Mill Creek Drive, Laguna Hills, California 92653, (714) 583-9119 FAX (714) 583-9213

CHNHEAD.CMN is the include file that defines the record CHN to be the structure /CHN_FILE_HDR_DESC/ and puts it into common CHNHEAD.

DF.CMN is the include file that declares common variables needed by the sources in the file DATA_FETCH.FOR. This group of subroutines reads the CHN file, one block at a time, examines the high-order nibble for an opcode, and the low-order twelve bits for data. It defines the common blocks:

(a) DF_OPCODES - contains the four possible opcodes.
(b) DF_INDICES - contains the many indices and counters used.
(c) DF_LOGICALS - contains various logical variables.
(d) DF_VAR - contains several indices, and other variables.
(e) DF_VAR1 - more of the above, and a block sized array.

FNAME.CMN is the include file that declares common variables that affect the overall logic. Components of file names are declared here.

Consider the following code fragment:

```
integer*4 l_alpha
character*10 alpha
alpha = 'G'
l_alpha = l
```

FORTRAN considers the length of alpha to be 10. The real length is 1, as counted by the number of nonblank characters. PRELEWD recognizes this and carries the real length in a related variable, here l_alpha. Then alpha(1:l_alpha) specifically picks out the real part of alpha.

The common blocks created are:

(a) CHAR_VAR_LEN1 - lengths in real characters of the character variables.
(b) CHAR_VAR_LEN2 - lengths in real characters of the character variables.
(c) CHAR_VAR_1 - the character variables.
(d) CHAR_VAR_2 - more character variables.
(e) CHAR_VAR - still more character variables.
(f) LUN_UNITS - the I/O unit numbers.
(g) FILE_VAR - information about the CHN file.
(h) CMD_LINE - variables, some logical, connected with the command line.
(i) OVERSIGHT - exactly what it says.

PPLOT.CMN is the include file which declares most of the variables used to format the plot, hold titles, determine dimensions etc. PPLOT.CMN includes file PPLOT.PRM, described later. Defaults for many of these variables are set in module DEFAULT_PLOT. Most of the variables in these commons are defined in the sources contained in file BUILD.FOR. This file has extensive comments. It creates the commons listed as follows:

(a)	CONTROL	- overall plot-control variables.
(b)	PLOT_LOGICAL_0	- logicals controlling which pieces to plot.
(c)	PLOT_LOGICAL_1	- logicals controlling which pieces to plot.
(d)	LEGEND_CHAR	- legend variables, location and size.
(e)	LEGENDA	- legend variables, location and size.
(f)	GRIDA	- number of grid lines in each direction.
(g)	TITLE_CHAR	- title line.
(h)	TITLES	- title locations.
(i)	X_AXIS_CHAR	- X-axis labels.
(j)	X_AXIS	- label locations
(k)	Y_AXIS_CHAR	- Y-axis labels.
(l)	Y_AXIS	- label locations.
(m)	RIGHT_CHAR	- right axis labels.
(n)	RIGHTA	- label locations.
(o)	PLOT_CHAR	- characteristics of the plot.
(p)	PLOT_0	- size and location of the plot surface.
(q)	PLOT_1	- size and location of the plot surface.
(r)	DEVICE_CHAR	- strings connected with the plotting devices.
(s)	DEVICES	- characteristics of each device.

SCALES.CMN declares variables that help dynamically scale numerical values to a power of 10 and the 1/2/5 scaling used. It creates only common SCALES.

SRAD.CMN declares variables connected with creating the SRAD files. It creates the common block SRAD.

UNITS.CMN declares variables which generate a proper label to go with the scaled data values. It creates the common blocks:

(a) UNITS_CHAR - strings with base units, and output units.
(b) UNITS_CONV - conversion factor from existing to new units.

VERDATE.CMN creates no common blocks, but contains comments that list the major changes that have occurred in PRELEWD since the last version. This file is included and listed only in the module ENVIRONMENT. It has a good history of PRELEWD.

XTRACT.CMN declares variables that are extracted from the various header records (the XTR_... modules) and used for the duration of the current plot. Defaults are set in DEFAULT_HEADER. It creates the following common blocks:

(1) G_AND_GC_1 - variables from /GEN_DESC/ and
(2) G_AND_GC_2 - /GENERAL_CHAN/ structures
(3) G_AND_GC_3 - such as TEST_NAME

(4) ALL_DEVICES_1 - variables common to all devices
(5) ALL_DEVICES_2 - such as Y_LABEL and PLOT_OPTION
(6) ALL_DEVICES_3

(7) SANDUS_1 - variables peculiar to the SANDUS in one of its
(8) SANDUS_2 - configurations such as PRETRIG_BYTES or
(9) SANDUS_3 - MODULE_TYPE
(10) SANDUS_4

(11) NON_SANDUS_1 - variables peculiar to non-SANDUS devices
(12) NON_SANDUS_2 - such as TRFRACT for RTD720s,
(13) NON_SANDUS_3 - NO_ZERO_CROSS for the Tektronix 7912s
(14) NON_SANDUS_4
(15) NON_SANDUS_5
(16) TRANSFER1 - variables that are obtained from other header variables,
(17) TRANSFER2 - or that were overlooked or ignored when first encountered.

5.4.4 INCLUDE Files Defining Parameters

The dynamic nature of underground testing requires that programs be flexible. It must be easy to change dimensions as each new test comes along. This is a good reason to use FORTRAN's parameter statement.

BIG_STRUCTS.PRM contains, among other parameters, the maximum number of subchannels allowed for those devices with subchannels. This file is maintained in INCLUDE$INCLUDE.

PPLOT.PRM contains parameters that define the maximum size of most of the PPLOT arrays. Other necessary parameters have been included here for consistency. This file is maintained in PRELEWD$INCLUDE.

PRELEWD_COMMAND_DEF.CLD is the file that defines the command line images, qualifiers, and parameters. This file is the final authority for all qualifiers and parameters on the PRELEWD command line. It is maintained in PRELEWD$INCLUDE.

5.4.5 SRAD References

SRAD is an acronym for Storage, Retrieval, Analysis, and Display. The formal SNL document, listed last, contains the information in the first document.

SRAD DATA FORMATS (Marking and Kissel; August 12, 1988) is an informal document that describes the SRAD format available first, and which PRELEWD uses.

UFO (UnFold Operator): Default Data Format (Kissel, Marking, and Biggs; March 7,1991, SAND91-0490) describes a more relaxed data format that has not yet been implemented.

5.5 Useful Information

5.5.1 Source, Object, and Executable Files

The source code is all written in FORTRAN, and while Run Time Library routines and System Service Routines are used, there is only FORTRAN-level I/O. This may not be true of GRAFPAK, but that is not important. The source code files usually contain several program modules that address the same area. The source is compiled /NOOPT/DEBUG, and the object modules are maintained in LEWD.OLB, an object module library created with the Librarian Utility. Because of the I/O in PRELEWD, there may be no advantage in using the /OPTIMIZE qualifier. Thus only one object library is necessary. This library is maintained in the directory PRELEWD$LIBRARY, and is defined by the programmer as follows:

```
$ DEFINE  PRELEWD$LIBRARY  LD:[PRELEWD] (or directory_name where file located)
```

Two executable files are created from LEWD.OLB, PRELEWD.EXE, and LEWD_DEBUG.EXE; the latter is created with the /DEBUG qualifier. The debug version of PRELEWD is for the use of the software developers, and is invoked as follows:

```
$ PRELEWD/DEBUG ... (other qualifiers and parameters as required)
```

COMMON blocks are maintained with the source in the directory PRELEWD$INCLUDE where that symbol is defined by the programmer as follows:

```
$ DEFINE  PRELEWD$INCLUDE  LD:[PRELEWD] (or directory_name where file located)
```

5.5.2 The XTRACT Modules

Because structure definitions and structure variable names can change, PRELEWD extracts data from the records into its own variables whose names do not change. This means that even though each channel description record has a variable called NUM_SUB_CHANS, which can be called out from the record as CHN.RC.NUM_SUB_CHANS for the RTD720, CHN.T.NUM_SUB_CHANS for the Tektronix 7912, or CHN.C.NUM_SUB_CHANS for the Tektronix 7103, having the logic to refer to the correct variable name at each use would be awkward. Instead, in the XTR... modules for each device, there is a statement of the following form:

NUM_SUB_CHANS=CHN.x.NUM_SUB_CHANS

and further references use the single variable NUM_SUB_CHANS.

5.5.3 The Variable Type

Where necessary, logical flow is device dependent. The PRELEWD integer variable TYPE carries two pieces of information about the current CHN file. MOD(TYPE,128) always has the primary channel type description value (i.e., 31 is a RTD720 channel. If the file is a calibration file (SANDUS, or Tektronix 7912 are the only devices that have this type of CHN file) then 128 is added to TYPE to carry that information. Thus, TYPE > 128 implies a calibration file, and MOD(TYPE,128) gives the channel type. This is done in HEADER_READ.FOR

5.5.4 Data Conversion

Data conversion from the A/D counts to engineering units is a linear conversion

Y(engineering units) = SLOPE * Y(counts) + OFFSET,

where SLOPE and OFFSET are determined by PROCESS (for the SANDUS), by PRELEWD (for the 7912) or by the instrument (for the RTD720). SLOPE and OFFSET are in the header, or sufficient information to calculate them is in the header. If PRELEWD cannot determine them, the plot proceeds, but the Y axis label becomes COUNTS.

For certain gauges an additional conversion may be required. These are always specific, and the variable PLOT_OPTION has a value other than NONE, COUNTS, or LINEAR. Certain extra conversions have been built into PRELEWD for ytterbium gauges, slifers, and Type-E and -K thermocouples. Except for slifers, these conversions have not been used.

If a new conversion is requested, first agree with the ICF creator what its callout will be. Then create a module to do it. This module must be integrated into the module GET_ANS (in

DATA_FETCH.FOR) based on a variable called IGA. IGA is generated in module GET_IGA (in CONJECTURE.FOR). Both these modules have extensive comments to help, but basically IGA is used in a COMPUTED GOTO.

5.5.5 Default Plot Values

The source file DEFAULTS.FOR contains two modules that set strange and very obvious default values into plot variables, and some header variables. The purpose was twofold: first to make sure that the variable was reset for each plot, and second to make sure that the variable had a good value. This is an especially useful feature when adding a new recording device to PRELEWD.

5.5.6 Data Thinning

When the number of X values to be plotted exceeds the number of rasters (pixels or ?) on the X axis of the display surface, time is wasted doing the computations for moving the pen (or its equivalent) up and down in the Y direction. An extreme example of this is certain realtime data files with a quarter of a million data points to be plotted on a laser printer with a resolution of 300 rasters/inch. In this situation there are about 80 points/raster over a 10 inch axis.

Two subroutines have been written which greatly speed up plot time for these situations. The two subroutines are THINN, and LOOKN. THINN is called by a user, and calls LOOKN. LOOKN does not call any other subprograms.

THINN is called with the following nine, long word arguments:

```
call thinn(noi,x,y,nop,xstart,xend,xp,yp,nopr)
```

where

NOI is the number of equally spaced intervals into which the span (=XEND-XSTART) is to be divided. Typical values are 300 dots/inch for a laser printer, or 650 for the full screen of a Tektronix 4208.

X and Y are input arrays each containing NOP points. X is assumed to be ordered such that X(I) < X(I+1) for I=1,NOP.

XP, and YP are output arrays each containing NOPR points. XP will be ordered such that XP(I) < XP(I+1) for I=1,NOPR. XP and YP must have dimensions at least 2*(NOI+1). This restriction on XP/YP dimensioning is a seldom-achieved upper bound on the number of output points.

The technique is to divide the span into NOI+1 intervals, and then determine the maximum and minimum Y for all points in each interval. The three cases to be considered are

(1) The trivial case where there are 0 input points in the interval. This will cause the most recently examined input point to be moved to the output array. This seems to work very well in PRELEWD where we normally have several input points per interval. It continues the plot smoothly when the original data were constant across several intervals, and the data compression software in PROCESS signaled a repeated value. This was done for PRELEWD, and may not give the desired results for your situation.

(2) Where there are either 1 or 2 input points in the interval, they are transferred to the output array.

(3) The significant case is where there are more than 2 input points in the interval. Here, (XMIN,YMIN) and (XMAX,YMAX) are determined to correspond the minimum and maximum input Y values in the interval. If the minimum (or maximum) Y value occurs more than once, the largest corresponding XMIN (or minimum will be used. The two points, (XMIN,YMIN) and (XMAX,YMAX), are transferred to the output array with care being taken to insure that XP remains ordered.

The above discussion assumes that the entire X/Y array is available, and even in these days of virtual memory, this may not be the case. When the XSTART, and XEND for the entire axis are known, but only a subset of the X/Y values are available at one time, THINN can be used. Let f=(X(NOP)-X(1))/(XEND_XSTART) be the fraction of the span for this subset of the total array. The call then looks like the following:

```
call thinn(nint(f*real(noi)),x,y,nop,x(1),x(nop),xp,yp,nopr)
```

NOTE: THINN does not check to see if 2*NOI > NOP, this is left to the user. When NOP/NOI > 100, the use of THINN is clear, and when NOP/NOI < 2, the output arrays will be the same as the input arrays (except for some pathological data). When NOP/NOI > 3, it may be advantageous to use THINN.

5.5.7 TEKTRONIX 7912 Calibration

This paragraph describes how 7912 calibration data is used in PRELEWD. The calibration is done in PRELEWD from information in the CHN header supplied by PROCESS, information which has been extracted from selected waveforms in the CALBIG file. Calibration of both the engineering values and the time axis occurs if there is sufficient data. The purpose of calibration is to apply a piecewise

linear correction to account for nonlinearities in both the vertical amplifier and the time base. The module that contains most of this is CAL_DATA_PROVIDED in CONJ_7912.for.

There are two calibrations of the 7912, one which occurs about 48 hours before an event. This is called the early calibrations and the BIG files are called T48BIG files. They are unique to 7912s. Late calibrations occur about four hours prior to an event. They are known as late cals, and the big files are CALBIG files. CALBIG files can also occur for SANDUS devices. It is the late calibration that is used by PROCESS to generate the data that it inserts into the file header.

The CALBIG file contains, for each channel, the following eight waveforms in the order given:

a) 1 baseline for "baseline" sine wave,
b) 1 baseline sine wave,
c) three DC levels,
d) two pulses, and
e) 1 experiment system baseline.

These eight waveforms are a subset of the set of 17 similar waveforms done during the early time calibration. The calibration values are obtained from the baseline sine wave (b, above) and the baseline for the baseline sine wave (a, above) to calibrate the time (X or abscissa) axis, and the three DC levels to calibrate the engineering unit (Y or ordinate) axis. These are obtained from the CALBIG file. Data from the T48 file is not used in calibration of the 7912s, but is available to the experimenter.

The three DC levels are recorded in algebraically increasing order. They are selected from the set of eight DC levels in the T48 file. The logical array CAL_BRIEF (in the T7912_CHAN_DESC structure) contains three TRUE values, where TRUE indicates that the DC level is common to both the early and late time AUTOCAL runs. The index of the TRUE value is the index into the CAL_LEVEL (in the /T7912_CHAN_DESC/ record) array which contains the engineering value (usually voltage) for that DC level.

PROCESS determines the mean value, in counts, for each of the three DC levels. This is done in the following manner:

(1) at every point having an upper and lower value, the mean of the upper and lower values is computed, and

(2) the average and standard deviation of these mean values are computed.

These values are stored by PROCESS, and included in the header of the CAL file and, later in the header of the CHN file for use by PRELEWD. They are the arrays MEAN, and SIGMA respectively (in the /CAL_7912/ record) in the same order as before.

These three values break the 7912s vertical 0-511 count range into four areas, 0-MEAN(1), MEAN(1)-MEAN(2), MEAN(2)-MEAN(3), MEAN(3)-511. A linearly interpolated correction is applied depending on whether the data value is in the range 1, 0-MEAN(2) or range 2, MEAN(2)-511.

y_corrected = slope_in_*j*th_range * y_recorded + offset_in_*j*th_range

where

slope_in_*j*th_range = {(CAL_LEVEL(j+1) - CAL_LEVEL(j)} /{(MEAN(j+1) - MEAN(j)}

offset_in_*j*th_range = CAL_LEVEL(j) - slope_in_*j*th_range * MEAN(j)

Note that extrapolation is done if the values are outside the range MEAN(1)-MEAN(3). The correction for the upper and the lower data values are determined separately.

PROCESS also determines the mean of the baseline for baseline sine wave in the manner described above, and using this value determines the count value of each baseline sine wave intersection with this mean. The number of crossings is NO_ZERO_CROSS (in the CAL_7912 structure), and the count value of each crossing is in the ZERO_CROSS array (in the CAL_7912 structure).

PRELEWD generates an array X_RANGE of the baseline crossing values at the start of each full cycle. The time that a full cycle should take is the sine wave's period which can be obtained from BASE_SINE_FREQ (in the T7912_CHAN_DESC structure). The corrected sample interval in the *j*th cycle is

period/number of counts in that cycle.

The uncorrected sample interval is TIME_DIV/51.1,

where TIME_DIV is in the /T7912_CHAN_DESC/ record.

The corrected time for a point in the *j*th interval is

t_corrected = corrected_sample_rate *

{current_count - count_at_beginning_of_interval} +

elapsed_time_to_beginning_*j*th_interval

If the current point falls outside a full cycle, either before the first crossing, or after the last full cycle crossing, an unmodified sample interval is used to compute the time.

Thus PRELEWD's output consists of two arrays, which may be plotted and/or written to an SRAD file. They are the following:

(a) an array of time values at which the samples occurred (NOTE: samples may not be evenly spaced in time), and

(b) an array of signal amplitude values in engineering units into the 7912.

The above-described corrections are enabled if PLOT_OPTION (in the T7912_CHAN_DESC structure) has the value LINEAR. Even with PLOT_OPTION = LINEAR, they can be defeated by using the /CALIBRATE=NONE qualifier on the PRELEWD command line. PRELEWD attempts to make the corrections unless:

(a) there are fewer then 2 TRUE values found in the CAL_BRIEF array, in which case signal amplitude corrections will be omitted.

(b) the number of counts in any full cycle is less than 0.5 or greater than 1.25 times the number of counts expected based on the period and sample rate.

(c) NO_ZERO_CROSS is less than the ICF value min-cycles, no time base corrections are made.

5.6 Useful Command Files

There are several .COM files which may be useful. Because of the limited disk space available, all of them do a PURGE before exiting. Since object modules are in an object library, they are not retained. If the user wants to retain a copy of something for other uses, it is suggested that he or she either rename the module, or copy it to another directory. The important .COM files are the following

(1) FORLIB.COM requires one parameter, the file to be compiled, with the resulting object modules put into LEWD.OLB. Its three DCL commands are shown below.

```
$   fortran/list/extend/nooptimize/debug 'p1'
$   library/replace/insert lewd 'p1'/log
$   delete *.obj;*
```

(2) LEWD.COM uses FORLIB.COM, and requires one parameter, which is the parameter required by FORLIB.COM, or the string NONE. When FORLIB.COM is done, or if the parameter was NONE, it links the GRAFPAK libraries with LEWD.OLB creating two modules PRELEWD.EXE, and LEWD_DEBUG.EXE. The link command is given below to show the libraries required.

```
$   link/exe=prelewd lewd/lib/include=prelewd, -
    repair$library:dump/lib, -
    utility$library:utility/lib, -
    sys$input/option
    gksdir:gksflb.exe/share
    sys$share:vaxcrtl.exe/share
```

(3) ALL_PRELEWD.COM deletes the existing LEWD.OLB, and creates a new empty LEWD.OLB. It uses DCL to find all .FOR files in the current directory. It sends each as found to FORLIB.COM. It is useful when an INCLUDE file changes, or when LEWD.OLB becomes unreadable.

(4) BATCH.COM uses ALL_PRELEWD.COM to compile all .FOR files in the directory, and LEWD.COM to create new executables. It deletes the .OLB file, and .EXE files. It was written to work with the DCL command SUBMIT.

Assuming that modules BLAH.FOR and CRAM.FOR have been changed, the following sequence is suggested:

```
$ @FORLIB BLAH  ! compile BLAH and put BLAH.OBJ into LEWD.OLB
$ @LEWD CRAM    ! compile CRAM, put CRAM.OBJ into LEWD.OLB and LINK
```

5.7 Source Files, Purpose, and Contents

There are a number of source files containing PRELEWD modules. Each source file contains one or more source modules, usually modules with a similar goal, or that work with similar objects. Changes are usually device oriented, and having all the modules for that device in one file is nice.

Production source code, the object library, executables, and include files are kept in the directory PRELEWD$INCLUDE. Development source code, its object library, and executables are maintained in another directory. A distinct .CLD file must be maintained in each directory.

Below is a brief description of the module's purpose. This is intended to provide a quick guide. The source for each module is prefaced with a more elaborate description of its purpose, the file(s) that calls it, and the file(s) it calls. The major modules also contain a history of their significant changes.

The files and the modules contained in each are as follows:

(1) PRELEWD.FOR contains the main program and other modules described below:

(a) PRELEWD is the main program, and controls the overall flow. It has all INCLUDE files listed. It contains the loop over all files called out in parameter p1, or p2.

(b) FOUND is an error routine used by everyone everywhere. See the discussion in the Operators Manual.

(c) HANDLE_A_FILE notifies the monitor program about the file being used, and controls the logic for each CHN file. It reads the header, and sets up all the header-dependent variables.

(d) SPECIAL_CONSIDERATIONS sets up for unusual DATA_FETCH calls such as SANDUS MUX channels in which each subchannel is on a new page.

(2) BUILD.FOR defines all plot parameters and locations for the various pieces of the plot, the colors, legend, and builds the static parts of the plot. It contains modules listed below:

(a) BUILD_PLOT makes many of the GRAFPAK calls, and controls the format of the plot.

(b) GRID generates a grid.

(c) LEGEND_XFORM creates the transformation that keep the data out of the legend. Legends are very awkward, requiring two different transformations, one for data with a time up to the time of the right edge of the legend, and one for data with times greater than that.

(d) LEGEND defines the legend box and contents. It is very device specific.

(e) RIGHT writes the labels on the right Y axis. These labels correspond to upper and lower band edge, and to calibration levels when they are known.

(f) WRITE_FILE writes title lines two through four, the standard lines. The first title line is an error line, and is written in CONJECTURE.

(g) U2NDC converts from user (data) to normalized device coordinates.

(h) WLABEL writes a label under a tick mark.

(i) LAB_LENGTH determines the length of a label based on the character size, and the number of characters.

(j) TAL sets the length of the tick marks as a fraction of the character size, and the specific axis they are on. It controls the tick and tick label logic.

(k) XAXIS controls both the top and bottom X-axis generation, its labels, and tick marks.

(l) YAXIS controls both the left and right Y- axis generation, its labels and tick marks.

(m) PRETTY_PLOT defines the plot basics such as character height and spacing, label and axis location. It sets the position for all the fixed elements of the plot except for the legend.

(n) MODIFY_X_LABEL modifies the X-axis label for special situations such as early and late calibrations.

(3) CLOSES.FOR closes GKS and the workstation, and contains these modules:

(a) CLOSE_STATION closes the current workstation when all CHN files have been plotted. It also closes the workstation, which closes the PLT file when GROUP files are in the current PLT file. It creates the COM file to send the plot file as requested, and initiates the SPAWN.

(b) DEACTIVATE_AND_CLOSE deactivates the current workstation prior to exit from GRAFPAK.

(c) PARSE_CLI parses the COM file DCL command into short pieces.

(4) COMMON.FOR is a collection of modules which have no INCLUDE files, hence seldom change, and contains the following modules:

(a) CLOSE_GKS closes GKS.

(b) NO_DUPES eliminates duplicate real-values entries from an array. Used for multiple baseline calibration marks on the right axis, if there can be duplicates.

(c) SORT_EM sorts an array of real numbers. Used with NO_DUPES.

(d) SAMUX_CAL is a dead-end module for SANDUS MUX cals.

(e) SDIG_CAL is a dead-end module for SANDUS digital cals.

(f) THINN cuts the number of data points to match the resolution of the workstation. See Section 5.5.6 for further information.

(g) LOOKN works with THINN above. Extracts the maximum and minimum values from an array of real values, and can identify the special cases where there are 0, 1, or 2 points in the array.

(h) COMMAND parses the command line for the parameter(s) and qualifier(s). It obtains the value for p1, and p2 if p1 is empty. Some of the qualifiers have a value, while the value for others is their presence or absence. Defaults are set here. Only qualifiers defined in the CLD file can be parsed.

(i) DETERMINE_DIMENSIONS determines dimensions for the data arrays. This is device dependent, with fixed-length arrays for the 7912s, and variable-length arrays for other devices.

(j) ELIMINATE_BLANKS removes leading and trailing blanks from a string, and returns the trimmed string, and its real length. See discussion in Section 5.4.3.

(k) FILE_COMPS breaks a file specification into its components, and returns those components and their real length.

(l) CLI parses the command line for the value of an input parameter, and returns that value, and its real length.

(m) LENGT determines the real length of a string.

(n) PARSE_STRING parses the command line for a qualifier, and returns its value, or a default value, if no value was supplied.

(o) PARSE_NUMBER parses the command line for a qualifier and returns either a default value, or the command-line value.

(p) ADJUST_MAX makes final adjustments to maximum scaling on axis. A first cut has been made on the maximum axis value. We now want to possibly modify it so that the maximum data value is less than the axis limit.

(q) ADJUST_MIN makes final adjustments to minimum scaling on axis. A first cut has been made on the minimum axis value. We now want to possibly modify it so that the minimum data value is greater than the axis limit.

(r) DATA_MAGNITUDE determines a scale factor which scales the data into a specified range. Returns the scale factor as the I in 10**I.

(s) MAJ_ANNOTATE aids in major tick mark annotation. It sets the number of labels on an axis, and determines the NDC increment between major tick marks.

(t) ORIGIN_OFFSET evaluates the axis offset, if any. Sets the axis origin at the minimum data value if the range of the data (max-min) is small compared to the maximum axis value.

(u) SET_FILE_NAME modifies an input file name by replacing the prefix with either PLT or SAD.

(v) LOOKS_OK tries to OPEN a file to determine if the user can write in the current directory. If successful, CLOSE the file and DELETE it.

(w) SHO_ERROR prints a string and notes a position in that string. This module works with SPECIAL_STR to put bad strings into the LOG file, pointing out the nonprinting character in the string.

(x) SPECIAL_STR removes nonprinting characters and annotates them.

(y) WTLENG determines the screen length of a string in current graphic units.

(z) WLIMIT determines the size of the current plotting surface in NDC units.

(aa) WAXIS draws an axis.

(ab) YT_ONE_D is a special one-dimensional ytterbium gauge conversion.

(ac) YT_HYDRO is a special two-dimensional ytterbium gauge conversion.

(ad) TYPE_K is a special type-K thermocouple conversion which uses a fifth-degree polynomial fit in one of two ranges, 0-400 degrees, or 400-1000 degrees C.

(ae) TYPE_E is a special type E thermocouple conversion which uses a fifth-degree polynomial fit in one of two ranges, 0-400 degrees, or 400-1000 degrees C.

(af) SLIFER is a special slifer gauge conversion from voltage to KHz. This is a linear conversion.

(ag) POLEVAL evaluates a fifth-degree polynomial. Used in the special conversions above. Contains the coefficients for each conversion.

(ah) CRACK_DIRECTORY aids in creating a meaningful LOG file tag.

(ai) DECODE_8_BIT_EXT decodes data from a very special SANDUS mode. This feature, while implemented, has never been used. It is described in *TA591 100 KHZ Data Module Operation and Maintenance Manual*, (D. Werling, SAND87-1736).

(aj) ELIMINATE_DBLE_TICS takes the ' " 's off command line qualifiers created by SUPERMON.

(5) CONJECTURE.FOR controls the translation of header information into plot information, and contains the modules:

(a) CONJECTURE generates the titles then calls the device-dependent routines for translation of header information into plot information.

(b) WHAT_LABEL returns a scaled axis label, given the base units and maximum axis value.

(c) LOR finds the location of and the text for calibration-level labels on the right axis.

(d) GET_IGA determines IGA, which controls the data conversion in GET_ANS. See Section 5.5.4.

(e) APPEND_ERROR starts the first title line, and/or appends an error to it. This line is formed based on logical variables in the record /GENERAL_CHAN/.

(6) CONJ_7912.FOR translates header information into plot information for the Tektronix 7912, and contains the following modules:

(a) T7912 sets the necessary variables for a Tektronix 7912 plot. It calculates the sample interval, defines variables that cause either one or two graph lines on the plot, prepares the x- and y-axis labels, and computes the slope and offset for the data conversion.

(b) T7912_CAL prepares for a multiple page Tektronix 7912 calibration plot. The calibration file can be either early or late. This causes differences in the number of graph lines per plot, and in the axis labels. Calibration plots are always in COUNTS. The contents of a CAL file is described in Appendix D.

(c) PAGE1_LABELS prepares axis labels for the first page of a multiple page 7912 cal plot.

(d) SCOPE converts a cal level to engineering units for the right axis label.

(e) PAGE2_LABELS prepares axis labels for the second page of a multiple page 7912 cal plot.

(f) PAGE3_LABELS prepares axis labels for the third page of a multiple page 7912 cal plot.

(g) T7912_CAL_EARLY reads only the T48 7912 CAL file.

(h) T7912_CAL_LATE reads only the 7912 CAL file.

(i) CAL_DATA_PROVIDED calculates the calibrations in time and/or Y. See section 5.5.7 for further information.

(j) XTR_T extracts data from the record /T7912_CHAN_DESC/ which contains channel (device) setup information.

(k) XTR_TC extracts data from the record /CAL_7912/, which contains channel calibration information.

(l) XTR_TD controls data extraction from Tektronix 7912 records.

(m) XTR_TS extracts data from the record /T7912_SUB_DESC/, which contains experiment information.

(7) CONJ_RTD720.FOR translates RTD720 header data into plot parameters, and contains the following modules:

(a) RTDS sets the necessary information for a RTD720 plot. This includes determination of the x- and y-axis span, based on the number of data points in the record, the axis labels, and the initial and final data time, based on the start of this segment's data in the record.

(b) RTD720_CAL is a dead-end module for nonexistent RTD720 cal files. It follows the established format for CONJ_* files.

(c) XTR_RD directs the data extraction from the RTD720 structures.

(d) XTR_RC extracts data from the record /RTD720_CHAN_DESC/, which contains channel-specific setup information.

(e) XTR_R_CAL extracts data from the record /CAL_RTD/, which currently contains no important data, but is retained for future use.

(f) XTR_RES extracts data from the record /RTD720_EXP_SUB_DESC/.

(g) XTR_RLS extracts data from the structure /RTD720_LC_SUB_DESC/.

(h) XTR_RCS extracts data from the structure /RTD720_CC_SUB_DESC/.

(i) XTR_R extracts data from the structure /RTD720_DEV_DESC/.

(8) CONJ_SALOG.FOR translates SANDUS ANALOG header data into plot parameters, and contains modules:

(a) SALOG sets values for a SANDUS ANALOG plot. This includes determination of the X- and Y-axis span, based on the number of data points in the record, the axis labels, and the initial and final data time.

(b) SALOG_CAL sets values for a SANDUS ANALOG CAL plot. This includes locating the cal levels in the file, and modification of the X axis to show the cal level.

(c) SANDUS converts counts into engineering units for the right label. This is complicated by the many special conversions that occur for SANDUS ANALOG data.

(d) YLIMITS converts counts at band edge to engineering units to define the Y axis extent.

(9) CONJ_SAMUX.FOR sets parameters for a SANDUS MUX plot. A SANDUS analog multiplex channel has not been used recently. It contains the following modules:

(a) SAMUX set values for a SANDUS MUX plot. This includes determination of the axis span, labels (there may be as many as 32 subchannels for each channel), and initial and final times for each subchannel. All subchannels may be done on one plot, or done one per page. See the MUX qualifier.

(b) SAMUX_POST prepares for the next subchannel of a SANDUS MUX plot.

(10) CONJ_SDIG.FOR translates SANDUS digital header data, and contains only module SDIG. There can be up to 8 digital subchannels for each channel. Each subchannel is put on horizontally. Subchannels are stacked vertically. Each subchannel is one bit in a channel data byte.

(11) CONJ_T7103.FOR translates Tektronix 7103 header data into plot parameters. This type of device was used experimentally on one event, and will probably never be used again. The file contains modules:

(a) CCD set values for a Tektronix 7103 CCD camera plot.

(b) CCD_CAL prepares for a 7103 CAL plot.

(c) XTR_C extracts data from the record /T7103_CHAN_DESC/.

(d) XTR_CC extracts data from the record /CAL_T7103/.

(e) XTR_CD drives the data extraction for the Tektronix 7103.

(f) XTR_CS extracts data from the record /T7103_SUB_DESC/.

(12) DATA_FETCH.FOR reads a block at time, and converts the data from counts to engineering units. It contains the following modules:

(a) DATA_FETCH initializes counters, reads all data, and initializes plotting. The source module has extensive documentation.

(b) FIRST_DATA_POINT handles the very first data point. The first point requires initialization for the X data value.

(c) PROCESS_ONE_RECORD controls the data recovery for one data record. It takes one data record, and breaks out the opcode, handling each of the possible opcodes.

(d) UNKNOWN_OPCODE flags the presence of an unknown or bad opcode.

(e) NEW_REPEAT has logic to control the repeat factor. A repeat factor arises during data compression only, and tells how many times the last valid data value is repeated before the next data value has changed by more than the compression factor. See Appendix D.

(f) DATA_VALUE initiates the conversion of data values to engineering units. In most cases, this is a linear conversion.

(g) PLOT_ARRAY_FILLED assumes control when data needs to be plotted. GRAFPAK can only handle 10,000 points. When that number of data values has been translated, it is necessary to plot, and start collecting again. The program also handles missing data.

(h) SAMP_RATE_CHANGE handles the sample-rate switchers. There can only be a sample-rate change from SANDUS ANALOG channels. If there is one, an error message is generated on the last data point indicating that it is looking for a sample rate beyond the allowed two rates. This can be ignored.

(i) FILL_GAP sets data values when there is a repeat factor.

(j) OPCODE extracts the opcode and the data from the input I*2 word.

(k) GET_ANS returns a Y value given a count value. Conversion is done based on the variable IGA (Section 5.5.4), which is based on the device and the value of PLOT_OPTION.

(l) GET_TIME returns a time value given the time at the last data point, the sample interval, and the current repeat factor. Up until the GKS routines are called, time is carried as a real*8 value.

(m) RDR reads the ith block of data.

(n) DATA thins the data, if necessary, and converts the time array to real*4. The module chooses the correct transformation if there is a legend, and calls the GKS plot routine.

(o) MUDDLE_7912 makes a single trace from a 7912 upper and lower trace. It is a simple mean of the two values.

(13) DEFAULTS.FOR resets commons between plots, and contains the following modules:

(a) DEFAULT_HEADER sets default values into the XTRACT commons. Some defaults will cause fatal errors, others will just call attention to themselves.

(b) DEFAULT_PLOT sets default values into the PLOT commons.

(14) ENVIRONMENT.FOR evaluates command line options, determines the files to be plotted, defines global variables, and contains the following modules:

(a) ENVIRONMENT defines logical units used, determines the current date and time, the username, the current directory name, and the current node name. It calls routines which parse the command line. It concerns itself with items that will hold for the entire run.

(b) CONDITIONS determines the files to be plotted if they are specified with the p1 parameter. The module checks raw data files, and the validity of the file names.

(c) INDIRECT parses the file named in the p2 parameter, and validates the file names given there.

(15) GET_FILE.FOR has most of the CHN file I/O routines, and contains the following modules:

(a) GET_FILE first inquires about then opens the current data file.

(b) SORT_BY_EXP will sort the list of data files, first by experimenter name, and then by experiment ID. See the /SORT qualifier. This is a political qualifier, never used, but available.

(c) DELETE_FILE deletes an unavailable file from the list of files, compresses the list, and readjusts the count. This occurs when the file cannot be opened.

(16) HEADER_READ.FOR reads the first block of the current data file's header, and extracts values from the two records /GEN_DESC/ and /GENERAL_CHAN/. This supplies the total header length, which allows the remaining header blocks to be read in. This is a useful "bootstrap" technique. This file contains only the module HEADER_READ.

(17) NICE_LIMITS.FOR adjusts the X- and Y- axis spans to the data. It gets any offset required, determines the number of major axis divisions, and adjusts the maximum and minimum of each axis to make 1,2,5 scaling. It contains only one module, NICE_LIMITS.

(18) OPENS.FOR opens both GKS and the work station, and contains the following modules:

(a) OPEN_GKS opens the plotting interface GKS, and turns error checking on.

(b) OPEN_STATION sets GKS to generate plots for the requested plotting surface. This is where the parameters for each possible plotting surface are defined (via DATA statements). Special calls for each plotting surface are done here. This module is very GKS specific.

(c) INQUIRE_STATION asks the user to define the workstation, if it was not defined on the command line, and if a BATCH job is not currently running.

(19) SRAD.FOR contains modules that write the SRAD file (SAD---.DAT). The headers are written with the same modules used with TABLEREAD, BIGREAD and CHNREAD (see Section 5.8.2 and Appendix I). This file contains the following modules:

(a) SRAD_CLOSE writes the final line, and closes the current SAD file.

(b) SRAD_DATA writes data to the current SAD file.

(c) SRAD_HEADER controls the writing of the header to the SAD file based on the primary channel type.

(d) SRAD_ANALOG writes the SANDUS ANALOG header to the SAD file.

(e) SRAD_MORE controls the writing of data when > 1 piece of plot. This occurs if there is missing data. This is usually a problem only with 7912s.

(f) SRAD_7912 writes the Tektronix 7912 header to the SAD file.

(g) SRAD_7912_MORE controls the writing of T7912 data when > 1 piece of plot.

(h) SRAD_T7103 writes the T7103 header to the SAD file.

(i) SRAD_DECIDE controls which SRAD data writing module to call.

(j) SRAD_RTD720 writes the header for the RTD720 to the SAD file.

(20) XTR.FOR extracts data from the header for common structures and for SANDUS structures, and contains the following modules:

(a) XTR_G extracts data from the record /GEN_DESC/. This consists of the variables TEST_NAME and SOURCE_NAME.

(b) XTR_GC extracts data from the record /GENERAL_CHAN/. This consists of information available to PROCESS/ANALYZE, and not available in other records.

(c) XTR_SA extracts data from the record /S_ALOG_CHAN_DESC/, which contains information about the instrument and about the experiment.

(d) XTR_SAC extracts data from the record /CAL_S_ALOG/, which is calibration information for this channel.

(e) XTR_SAD calls XTR_SA, and XTR_SAC.

(f) XTR_SD extracts data from the record /S_DIG_CHAN_DESC/, which contains information about the instrument setup.

(g) XTR_SDC extracts data from the structure /CAL_S_DIG/, which contains calibration information available from PROCESS.

(h) XTR_SDD calls XTR_SD, XTR_SDS, and XTR_SDC.

(i) XTR_SDS extracts data from the record /S_DIG_SUB_DESC/, which contains experiment information about each subchannel.

(j) XTR_SM extracts data from the record /S_AMUX_CHAN_DESC/, which contains instrument setup information.

(k) XTR_SMC extracts data from the record /CAL_S_AMUX/, which contains calibration information from PROCESS.

(l) XTR_SMD calls XTR_SM, XTR_SMS, and XTR_SMC.

(m) XTR_SMS extracts data from the record /S_AMUX_SUB_DESC/, which contains experiment information for each subchannel.

(21) XTRACT.FOR calls the appropriate extraction module for the primary channel type, and contains only one module, XTRACT.

5.8 External Modules

Two external object libraries, and one shared library supply modules not in PRELEWD's source files. Note that LEWD.OLB is external to a number of other executables.

5.8.1 Modules Needed from UTILITY.OLB

This library consists of a number of useful, basic modules written by the staff, and readily available (see Appendix H). UTILITY.OLB is maintained in UTILITY$LIBRARY, which is defined as

$ DEFINE UTILITY$LIBRARY LD:[UTILITY] (or wherever the library is maintained)

(1) CHANGE_COMM_SYM enables program-command procedure communications on one node.
(2) CHN_CHAR_INT converts numeric characters to an integer in specified base.
(3) DISCONNECT_NETWORK_LINK disconnects a node-to-node communications link.
(4) FILE_DATE obtains the creation date of the executable module.
(5) PARSE_PRESENCE gets information about a command line parameter.
(6) REQUEST_NETWORK_LINK opens a node-to-node communications link.
(7) SEND_NETWORK_MSG sends a short message over the node-to-node link.

5.8.2 Modules Needed from DUMP.OLB

This object library contains at least one module for every nonempty structure defined in BIG_STRUCTS.DEF, TABLE_STRUCTS.DEF, and CHN_HEADER.DEF. It is used as part of the debugging facilities developed for the NTS Instrumentation System (see Appendix I).

DUMP.OLB is maintained in REPAIR$LIBRARY, which is defined as

$ DEFINE REPAIR$LIBRARY LD:[REPAIR] (or wherever the library is maintained).

The following routines are called only from the SRAD_... modules, and creates an ASCII dump of the record whose structure is given in the module name after the DUMP_ or DUMPI_. In the case of DUMPI, it dumps the *i*th record in the record array.

The following modules dump records whose structures are defined in TABLE_STRUCTS.DEF.

(1) DUMPI_RTD_CC_SUB_DESC
(2) DUMPI_RTD_EXP_SUB_DESC
(3) DUMPI_RTD_LC_SUB_DESC
(4) DUMPI_T7103_SUB_DESC
(5) DUMPI_T7912_SUB_DESC
(6) DUMP_GEN_DESC
(7) DUMP_RTD_CHAN_DESC
(8) DUMP_RTD_DEV_DESC
(9) DUMP_S_ALOG_CHAN_DESC

(10) DUMP_T7103_CHAN_DESC
(11) DUMP_T7912_CHAN_DESC

The following modules dump records whose structures are defined in CHN_HEADER.DEF.

(1) DUMP_GENERAL_CHAN
(2) DUMP_CAL_7912
(3) DUMP_CAL_RTD
(4) DUMP_CAL_S_ALOG
(5) DUMP_CAL_T7103

5.8.3 Modules Needed from the Shared Library GKSFLB.EXE

This is the GKS library. It is supplied under license from ATC and must be available on all nodes where PRELEWD is to be run. PRELEWD uses only a small part of the capabilities available in GKS. GKSFLB.EXE is maintained in the directory specified by the system-wide symbol GKSDIR.

(1) GACWK - activates a workstation.
(2) GCLKS - closes GKS.
(3) GCLRWK - clears a workstation.
(4) GCLWK - closes a workstation.
(5) GDAWK - deactivates a workstation.
(6) GECLKS - initiates an emergency close of GKS.
(7) GOPKS - opens GKS.
(8) GOPWK - opens a workstation.
(9) GPL - draws a polyline.
(10) GPREC - packs a data record.
(11) GQACWK - inquires about active workstations.
(12) GQCHUP - inquires about the current character up vector.
(13) GQDSP - inquires about the workstation display size.
(14) GQOPS - inquires about the GKS operating state.
(15) GQTXP - inquires about the current text path.
(16) GQTXR - inquires about the current text representation.
(17) GQTXX - inquires about a text string's extent.
(18) GSASF - sets aspect source flags.
(19) GSCHH - sets the character height.
(20) GSCHUP - sets the character up vector.
(21) GSCR - sets the color representation.
(22) GSDS - sets the deferral state.
(23) GSELNT - selects the normalization transformation.
(24) GSPLCI - sets the line color.

(25) GSPLI - sets a polyline bundle index.
(26) GSPLR - sets polyline representation.
(27) GSTXI - sets a text bundle index.
(28) GSTXP - sets the text path.
(29) GSTXR - sets text representation.
(30) GSVP - sets viewport.
(31) GSWKVP - sets workstation viewport.
(32) GSWKWN - sets workstation window.
(33) GSWN - sets the window.
(34) GTX - writes text.
(35) GUESC - an unregistered escape from GKS (opens and closes the Tektronix 4208 dialogue area).
(36) GUESC002 - an escape from GKS allowing a pause between screen plots.
(37) GUESC050 - an escape from GKS allowing the PLT file to be named.
(38) GUESC051 - an escape from GKS to set the error severity level.
(39) GUESC052 - an escape for GKS which allows error checking.
(40) GUESC302 - an escape from GKS allowing a look at the last GKS error.

5.9 Records and Structures

For a detailed description of the DEC's nonstandard implementation of records and structures, see both the VAX FORTRAN Language Reference Manual Order Number AA-D034E-TE, dated June 1988, and the VAX FORTRAN User Manual Order Number AA-D035E-TE dated June 1988. More recent editions exist.

If the user chooses not to delve deeply into records and structures, he or she should think of a structure as a description of a collection of variables, a plan, and a record as the structure's realization in memory. The structure is the architect's plan, while the record is the construction company's building, realizing that other buildings may be built using the same set of plans. The variables described in the structure may be any legal FORTRAN type in any order. In a record, the variables are stored in structure order with no blank space. A record statement assigns a variable name to a structure. A structure description may contain record statements.

Use the variable name defined in a record statement in much the same way that usual variable names can be used, but the name refers to all the elements contained in that record. The structure defined by a record statement may be dimensioned. An individual element in a record is referenced by prefixing its name with the name of each record it is a member of, working outward. Thus the variable name A.B.C(i).X is element X of the ith record C in record B in record A. There is a difference between a structure and a record, but many authors use the two words interchangeably.

5.9.1 CHN Header Structure

The following is taken from CHN_HEADER.DEF and is the structure that describes the CHN file header.

If the operator understands this structure, he or she will have no problem with any other structure. The lines have been numbered to help explain this structure.

```
01  structure / CHN_FILE_HDR_DESC /
02    union
03        map
04          record / GENERAL_CHAN / gc
05          record / GEN_DESC / g
06          union
07              map
08                record / S_ALOG_CHAN_DESC / sa
09                record / CAL_S_ALOG / sa_cal
10              end map
11              map
12                record / S_DIG_CHAN_DESC / sd
13                record / S_DIG_SUB_DESC / sds(1:S_DIG_SUB_MAX)
14                record / CAL_S_DIG / sd_cal
15              end map
16              map
17                record / S_AMUX_CHAN_DESC / sm
18                record / S_AMUX_SUB_DESC /  sms(1:S_AMUX_SUB_MAX)
19                record / CAL_S_AMUX / sm_cal
20              end map
21              map
22                record / T7912_CHAN_DESC / t
23                record / T7912_SUB_DESC / ts
24                record / CAL_7912 / t_cal
25              end map
26              map
27                record / T7103_CHAN_DESC / c
28                record / T7103_SUB_DESC / cs
29                record / CAL_T7103 / c_cal
30              end map
31              map
32                record / RTD_DEV_DESC / r
33                record / RTD_CHAN_DESC / rc
34                union
```

```
35                    map
36                      record / RTD_EXP_SUB_DESC / res
37                    end map
38                    map
39                      record / RTD_LC_SUB_DESC / rls
40                    end map
41                    map
42                      record / RTD_CC_SUB_DESC / rcs
43                    end map
44                  end union
45                  record / CAL_RTD / r_cal
46                end map
47              end union
48            end map
49            map
50              byte Z(512,6)
51            end map
52        end union
53    end structure
```

Note that this structure has both variables (Z, line 50) and records as elements. Some of the records are dimensioned (lines 13 and 18). The parameters that give the upper bounds of these dimensions are in BIG_STRUCTS.PRM. For every STRUCTURE there is an END STRUCTURE, for every UNION an END UNION, and for every MAP an END MAP.

Starting at the right-most indentation levels, the UNION/END UNION of lines 34-44 includes three MAP/END MAP groups, lines 35-37, 38-40, and 41-43. A UNION/END UNION is similar to a FORTRAN equivalence statement, and the MAP/END MAP groups define the elements that are equivalent. The records named RES, RLS, and RCS will start at the same location, and the longest will define the space taken by this UNION/END UNION group.

Now consider the UNION/END UNION group, lines 6-47. It has the 6 MAP/END MAP groups, lines 7-10, 11-15, 16-20, 21-25, 26-30, and 31-46. Each of these 6 groups is specific to a specific source device, and each defines the layout of the CHN file header for that source device. Lines 31-46 define the RTD720 specific part of the header.

Now look at the UNION/END UNION group, lines 2-52. It covers two MAP/END MAP groups, lines 3-48, and 49-51. Lines 4 and 5 define the two records which are PHYSICALLY and LOGICALLY first in the header, and in ALL headers. The MAP/END MAP group at line 49-51 defines a byte variable Z dimensioned (512,6). Z and the MAP/END MAP group, lines 3-48, occupy the same address space. Six is the length in physical records of the longest device header, and 512 is the byte length of a physical record. This construction allows the first physical record

from the file to be read using an ordinary binary file read. The variable CHN.GC.LEN_HEADER in the record GC then indicates how many more physical records need to be read from this CHN file. See module HEADER_READ for details.

The STRUCTURE/END STRUCTURE statements, lines 1-53, complete the structure definition. The include file CHNHEAD.CMN declares this structure to be a record named CHN. There may be several record statements, each declaring this structure to have a locally unique name.

5.10 Files Created

FORTRAN logical units 2, and 5-10 are defined in module ENVIRONMENT. For temporary use, logical unit numbers come from the RTL routines LIB$GET_LUN and LIB$FREE_LUN. LUN values below 5 are reserved for the output plot file. Currently, only the value 2 is used. LUN (=10) is the CHN file. INLUN (=5) is the keyboard for input if this is not a batch job. IOUTLUN (=6) is for the terminal, and is used if this is not a batch job. IOUTLOG (=7) is for the LOG file. SRADLUN (=9) is for the SRAD file.

5.10.1 The LOG File

A file, PRELEWD.LOG, is always created in the current directory for each execution of PRELEWD. This file is written by the module FOUND, which is described later. PRELEWD.LOG is the first place to begin troubleshooting. It has an execution date/time stamp, the command line, and the name and creation date of the .EXE file. It lists each existing file that is supplied either by parameter p1, or p2. As each CHN file is started, a message is sent to the .LOG file, and to the screen if the job is interactive. It tells of any unusual values found in the header, and what was done to fix the value; the number of data values before and after compression, and sent to the plotting surface. Each entry is annotated with the originating module's name.

5.10.2 FOUND, The Error Routine

Module FOUND is the standard error routine used here, though not always exactly the same version. For further information on the LOG file format required by SLOG, see Appendix G. A typical call might look like the following:

```
                    ..... from CLOSE STATION .....

        if( .not. istat )
1           call found(true,istat,
2                   'CLOSE STATION',2,2,
3                   'Trying to spawn the '
4                   //'string: '//cf(1:lcf),
5                   lplot_que_name,3)

                    ..... End of code fragment .....
```

Explanation: A command procedure to print a PLT file has been generated, and the necessary LIB$SPAWN command executed, but for some reason it failed (istat = .false.). This call to FOUND indicates that something out of the ordinary happened, and shows the command trying to be executed.

Note the use of the concatenation symbol "//" to extend the sixth argument. The significance of each of the eight arguments is (in order) as follows:

(1) .true. — a logical and if .true. then argument 2 is meaningful.

(2) istat — the status value returned by a system routine, in this case LIB$SPAWN. If this argument is meaningful, LIB$SIGNAL is called, and the message is added to the LOG file.

(3) CLOSE STATION — a string containing the name of the calling module.

(4) 2 — an integer in the range 1 to 3 which defines the error level written to the LOG file, and controls some module logic.

1 = = > 'INFORMATIVE'
2 = = > 'NON-FATAL ERROR'
3 = = > 'FATAL ERROR'

(5)	2	an integer in the range 1 to 17, and an index into a set of canned messages. They are defined in the source code. In this case the message is, "Occurred trying to use RTL routine."
(6)	'Trying...'//cf(1:lcf)	the specific message about what was found, and what, if anything, was done about it. Notice here how it includes the string cf(1:lcf). This argument can be any reasonable length.
(7)	lplot_que_name	an integer, which may or may not be meaningful. In this case it is the length the string plot_que_name.
(8)	3	an integer value in the range of 1 to 9 which indicates the programmer's sense of the severity of the message. 1 is usually fatal, and 9 is usually informative. Here, the user will not get plots he requested because the spawn failed. This value is put into the SLOG tag. See Appendix G.

Because every error is in the log file, and each is tagged with the name of the originating program, all the possible errors are not listed here. The DEC utility program SEARCH performs this function.

For further information on the LOG file format required by SLOG, see Appendix G.

5.10.3 The PLT File

A PLT file is created if and only if the plotting surface is chosen to be, or defaults to a printer. The printer must be a QMS laser printer. The *nnnxxxyzzz-ww*.DAT of the first CHN file name encountered in a new group is changed to PLT*xxxyzzz-ww*.DAT, and GROUP (default=4) CHN file plots are included in that file. This occurs in the module CLOSE_STATION.

If the PLT files are queued to a printer via the command line, they are deleted after printing. If PLT files are sent to another node and queued for printing via the command line, then the original remains in the current directory. In this case, for each PLT*xxxyzzz*.DAT file created, a PLT*xxxyzzz-ww*.COM file is created in the current directory that does the COPY to the new node, and queues the file for printing on that node. The RTL routine LIB$SPAWN (no wait) executes this command procedure. If created, the .COM file is not deleted. An attempt is made to be sure that PRELEWD does not use DECnet to send files to the node it is currently running on.

5.10.4 The SRAD File

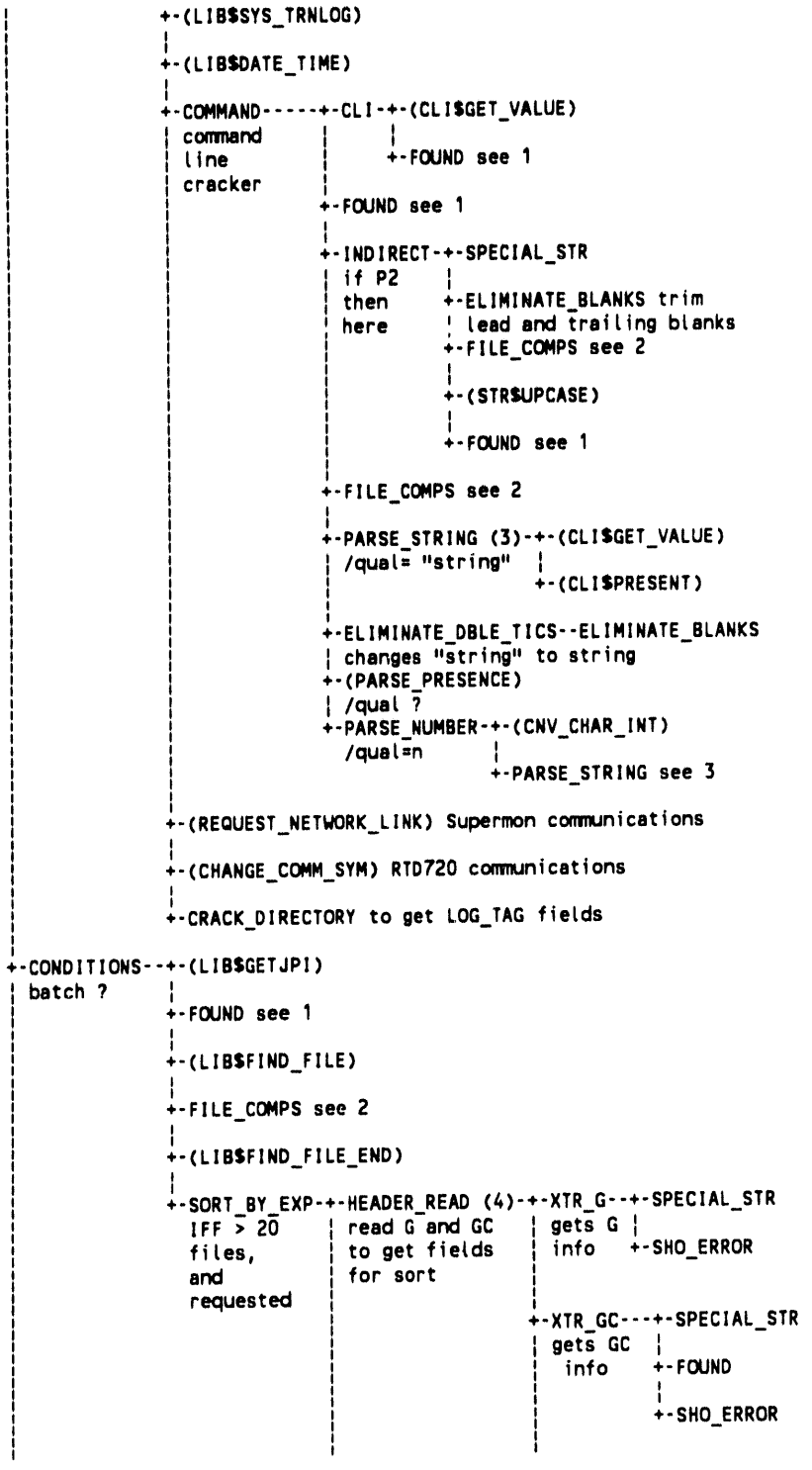
SRAD files may be requested from the command line, in which case all the CHN files processed will have SRAD files created, or on a file-by-file basis using the logical header variable SRAD_FILE (= .TRUE.). SRAD files are much larger than the parent CHN file, so caution is advised. If the user must create SRAD files on a regular basis, consider using the QRAD qualifier. See Section 3.1. SRAD files are named SAD*xxxyzzz-ww*.DAT after the parent. A copy of the SRAD file will remain in the current directory if the file is sent to another node. No .COM file is involved; the COPY command is SPAWNed directly using LIB$SPAWN.

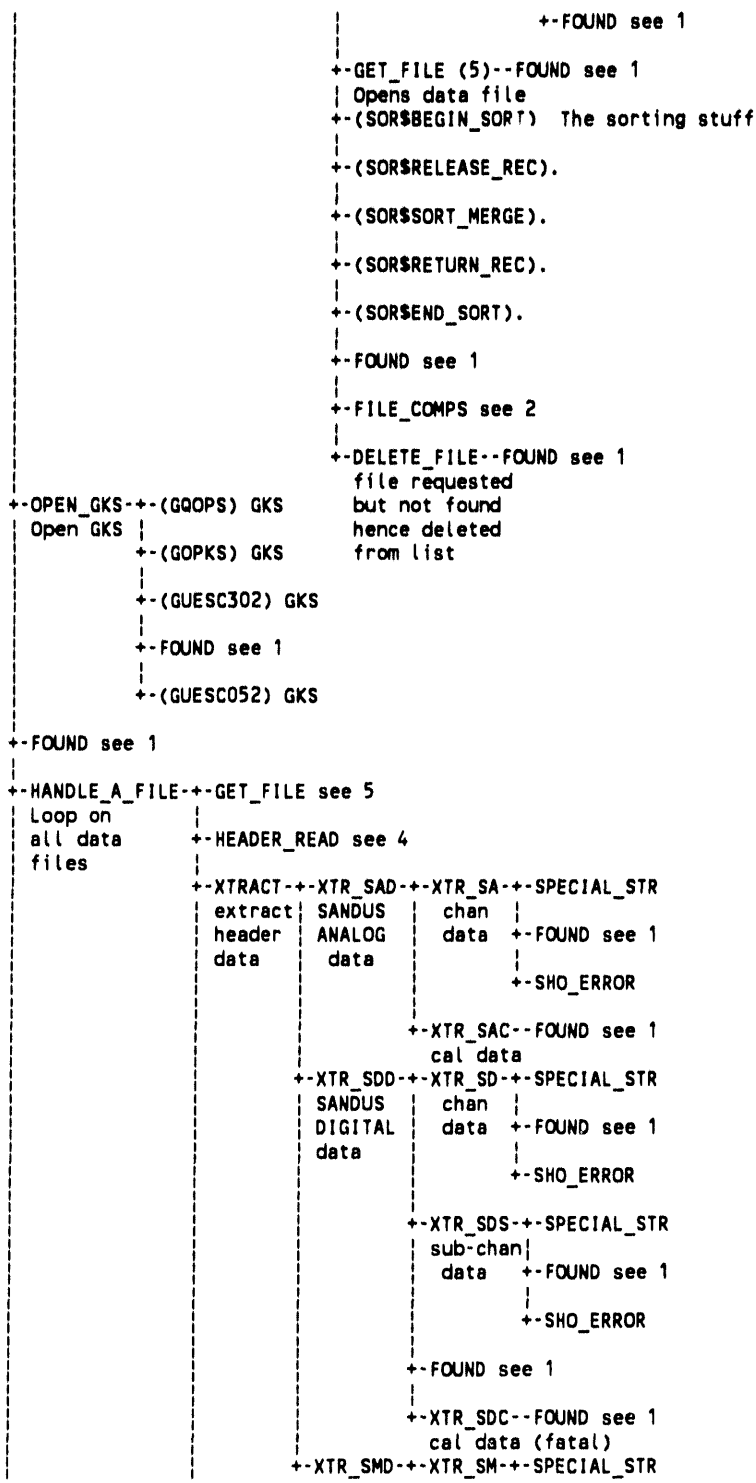
5.11 The CALL Tree

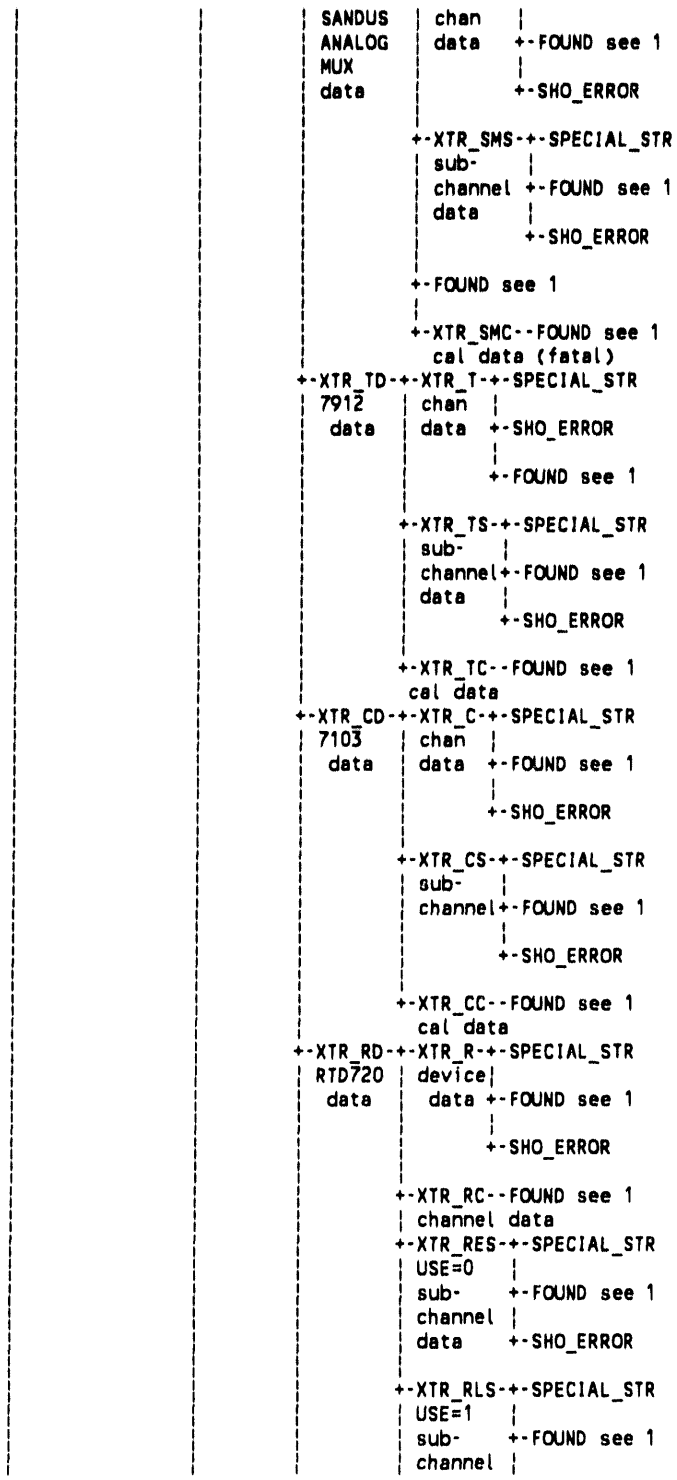
The following call tree is from the program FLINT (v 2.71). At least the first occurrence of each routine is annotated here. Those routines that appear as (name) are external to PRELEWD. Those with names followed by (*n*) are referenced later as "see n."

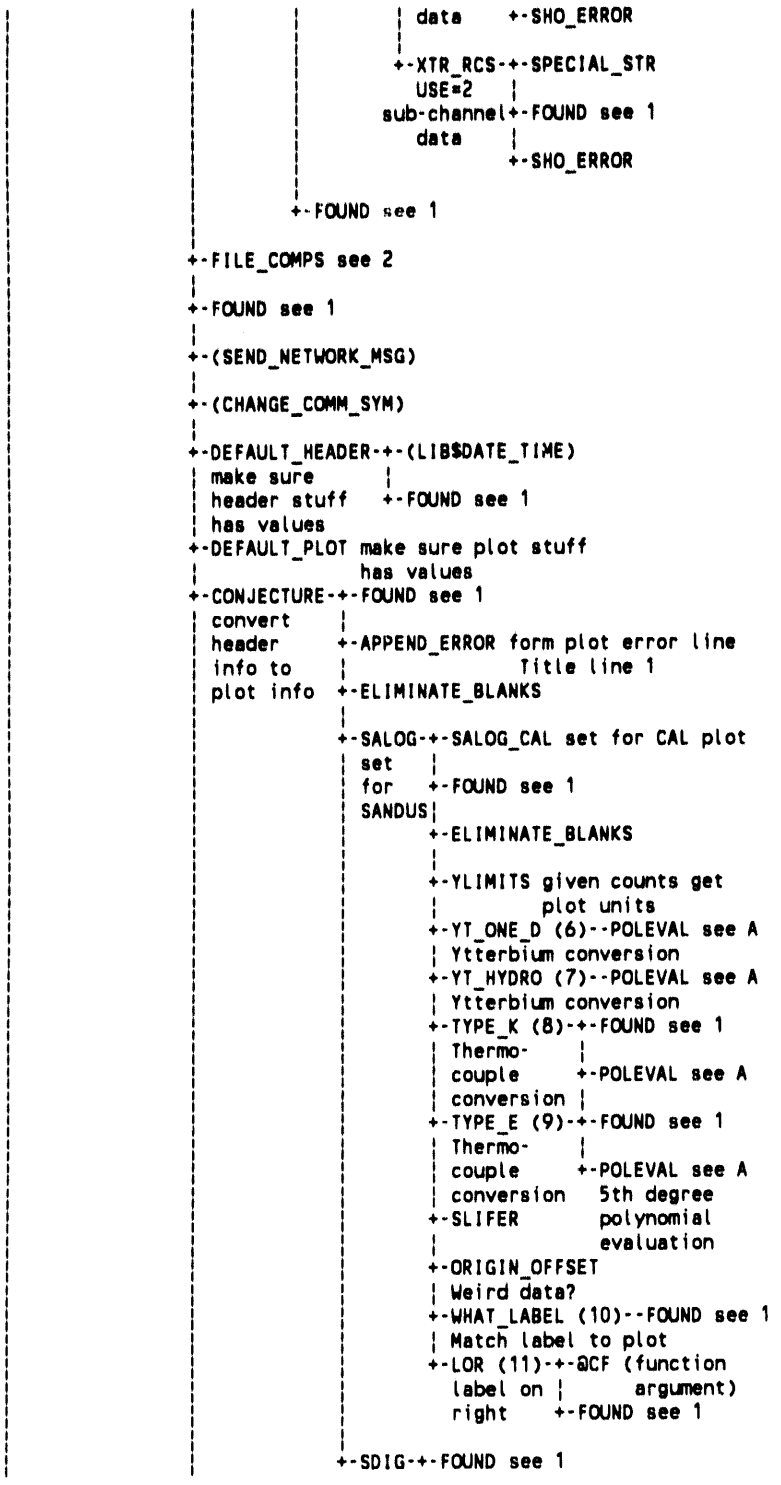
This is a primary tree starting at the program 'PRELEWD'

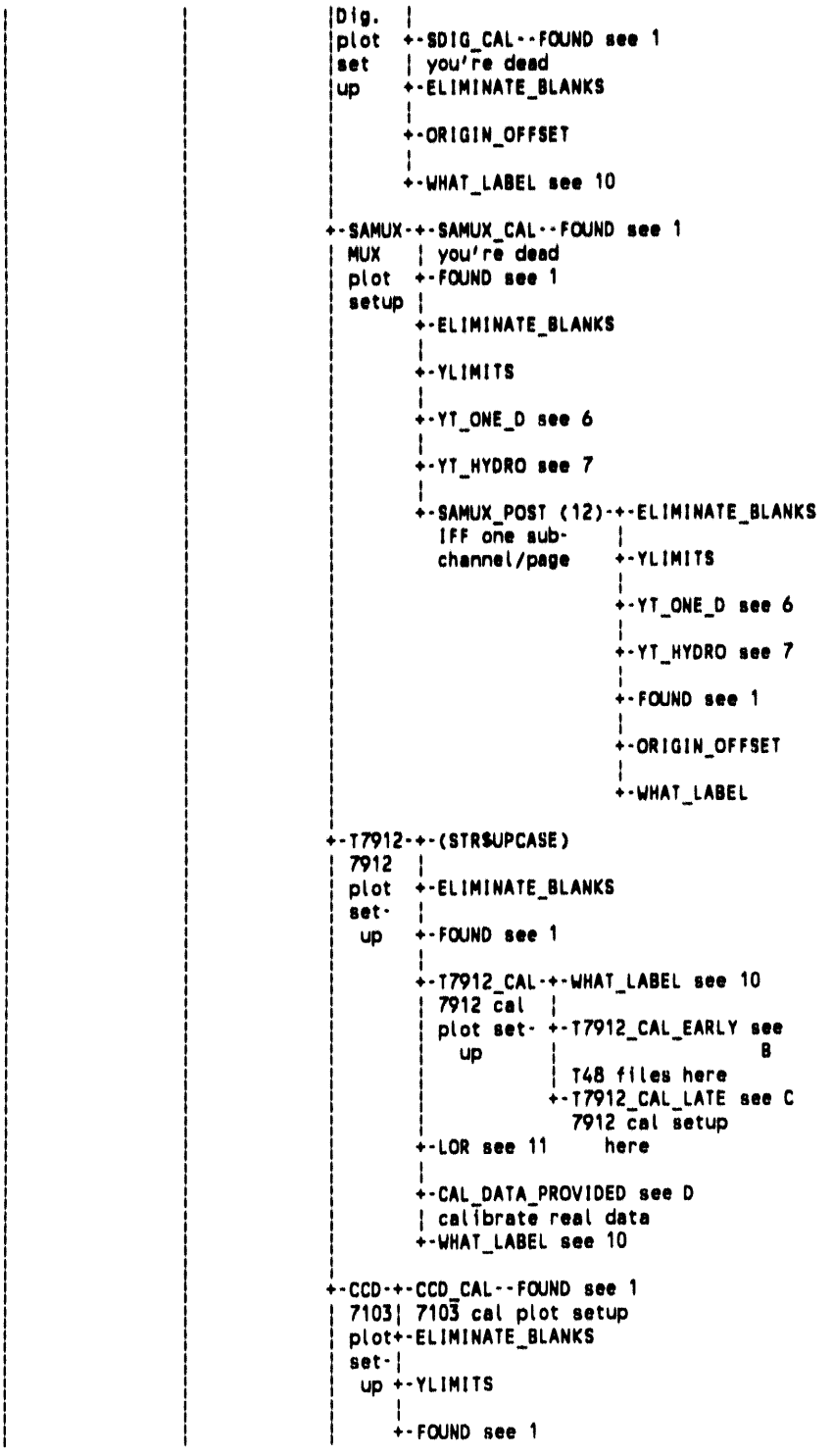
```
PRELEWD-+-ENVIRONMENT-+-(FILE_DATE) name and date of executable file
 main   | who, what   |
program | where, when +-(LIB$GETJPI)
        | why         |
                      +-FOUND (1)+-(LIB$SYS_GETMSG)
                      | log file |
                      | messages-+-(SEND_NETWORK_MSG) Supermon responses
                      | written  |
                      |  here    +-(CHANGE_COMM_SYM) RTD responses
                      |          |
                      |          +-(DISCONNECT_NETWORK_LINK) Supermon
                      |          |                           turnoff
                      |          |
                      |          +-(GQOPS) GKS close out
                      |          |
                      |          +-(GECLKS) GKS close out
                      |          |
                      |          +-(LIB$SIGNAL)
                      |          |
                      |          +-(LIB$ENABLE_CTRL)
                      |          |
                      |          +-(EXIT) iff error was FATAL
                      |
                      +-(LIB$FIND_FILE)
                      |
                      +-(LIB$FIND_FILE_END)
                      |
                      +-FILE_COMPS (2)-+-(SYS$FILESCAN)
                      | file           |
                      | components     +-FOUND see 1
                      |                |
                      |                +-LENGT
                      |
```

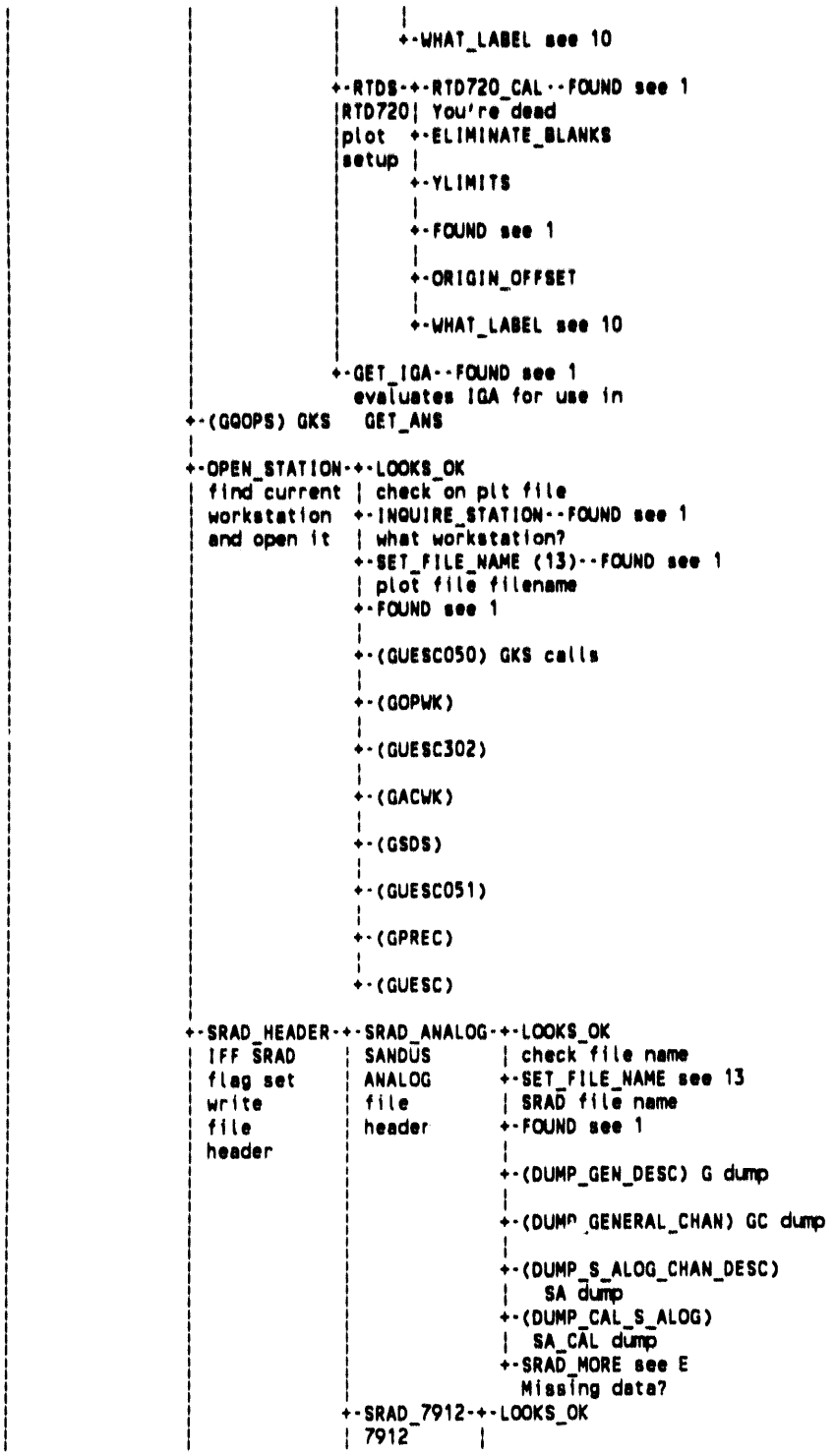

+-(LIB$SYS_TRNLOG)
+-(LIB$DATE_TIME)
+-COMMAND-----+-CLI-+-(CLI$GET_VALUE)
command line cracker
+-FOUND see 1
+-FOUND see 1
+-INDIRECT-+-SPECIAL_STR
if P2 then here
+-ELIMINATE_BLANKS trim lead and trailing blanks
+-FILE_COMPS see 2
+-(STR$UPCASE)
+-FOUND see 1
+-FILE_COMPS see 2
+-PARSE_STRING (3)-+-(CLI$GET_VALUE)
/qual= "string"
+-(CLI$PRESENT)
+-ELIMINATE_DBLE_TICS--ELIMINATE_BLANKS
changes "string" to string
+-(PARSE_PRESENCE)
/qual ?
+-PARSE_NUMBER-+-(CNV_CHAR_INT)
/qual=n
+-PARSE_STRING see 3
+-(REQUEST_NETWORK_LINK) Supermon communications
+-(CHANGE_COMM_SYM) RTD720 communications
+-CRACK_DIRECTORY to get LOG_TAG fields
+-CONDITIONS--+-(LIB$GETJPI)
batch ?
+-FOUND see 1
+-(LIB$FIND_FILE)
+-FILE_COMPS see 2
+-(LIB$FIND_FILE_END)
+-SORT_BY_EXP-+-HEADER_READ (4)-+-XTR_G--+-SPECIAL_STR
IFF > 20 files, and requested
read G and GC to get fields for sort
gets G info
+-SHO_ERROR
+-XTR_GC---+-SPECIAL_STR
gets GC info
+-FOUND
+-SHO_ERROR

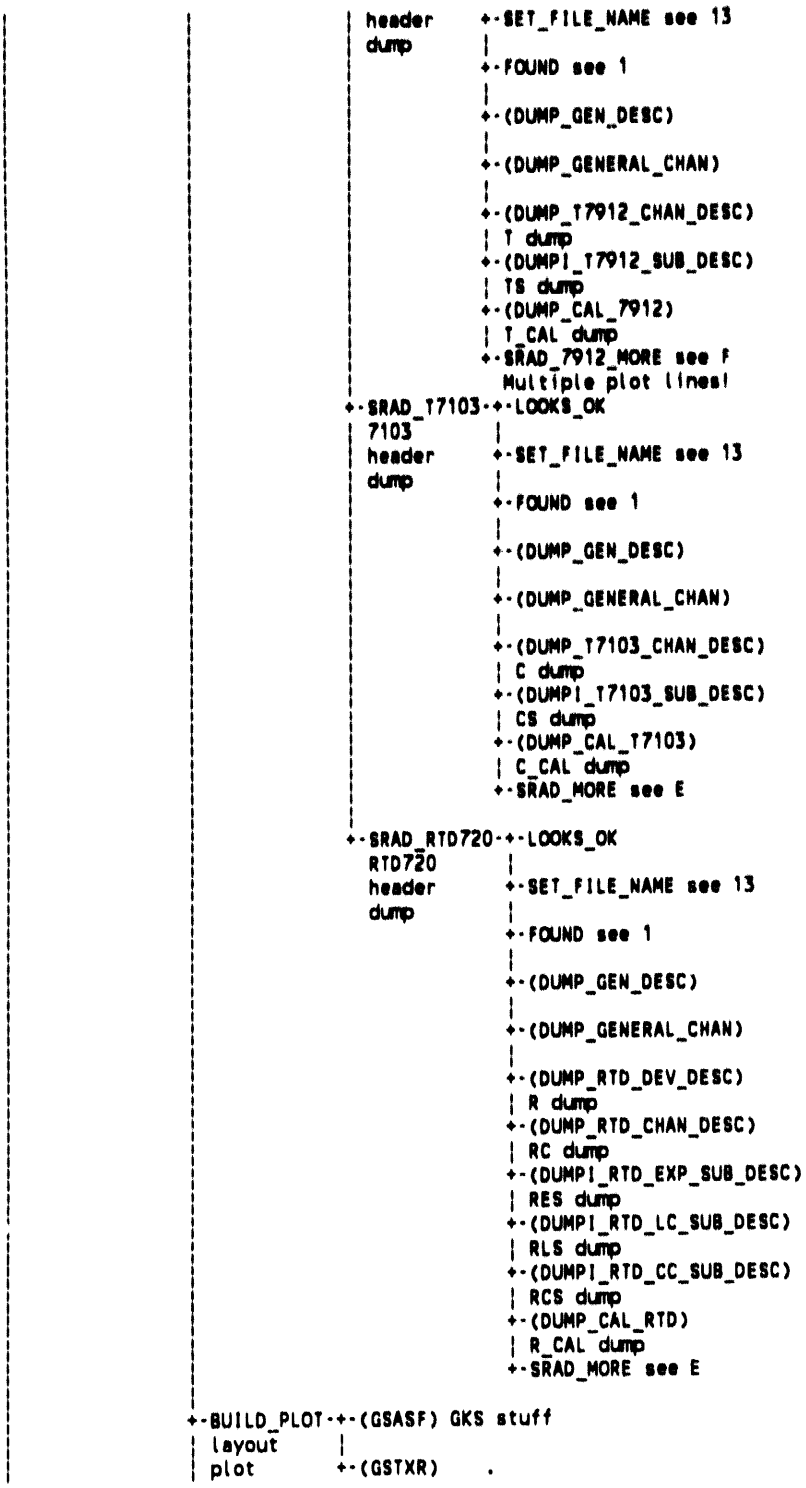

+-FOUND see 1
+-GET_FILE (5)--FOUND see 1
Opens data file
+-(SOR$BEGIN_SORT) The sorting stuff
+-(SOR$RELEASE_REC).
+-(SOR$SORT_MERGE).
+-(SOR$RETURN_REC).
+-(SOR$END_SORT).
+-FOUND see 1
+-FILE_COMPS see 2
+-DELETE_FILE--FOUND see 1
file requested
but not found
hence deleted
from list
+-OPEN_GKS-+-(GQOPS) GKS
Open GKS +-(GOPKS) GKS
+-(GUESC302) GKS
+-FOUND see 1
+-(GUESC052) GKS
+-FOUND see 1
+-HANDLE_A_FILE-+-GET_FILE see 5
Loop on all data files
+-HEADER_READ see 4
+-XTRACT-+-XTR_SAD-+-XTR_SA-+-SPECIAL_STR
extract header data
SANDUS ANALOG data
chan data +-FOUND see 1
+-SHO_ERROR
+-XTR_SAC--FOUND see 1
cal data
+-XTR_SDD-+-XTR_SD-+-SPECIAL_STR
SANDUS DIGITAL data
chan data +-FOUND see 1
+-SHO_ERROR
+-XTR_SDS-+-SPECIAL_STR
sub-chan data +-FOUND see 1
+-SHO_ERROR
+-FOUND see 1
+-XTR_SDC--FOUND see 1
cal data (fatal)
+-XTR_SMD-+-XTR_SM-+-SPECIAL_STR

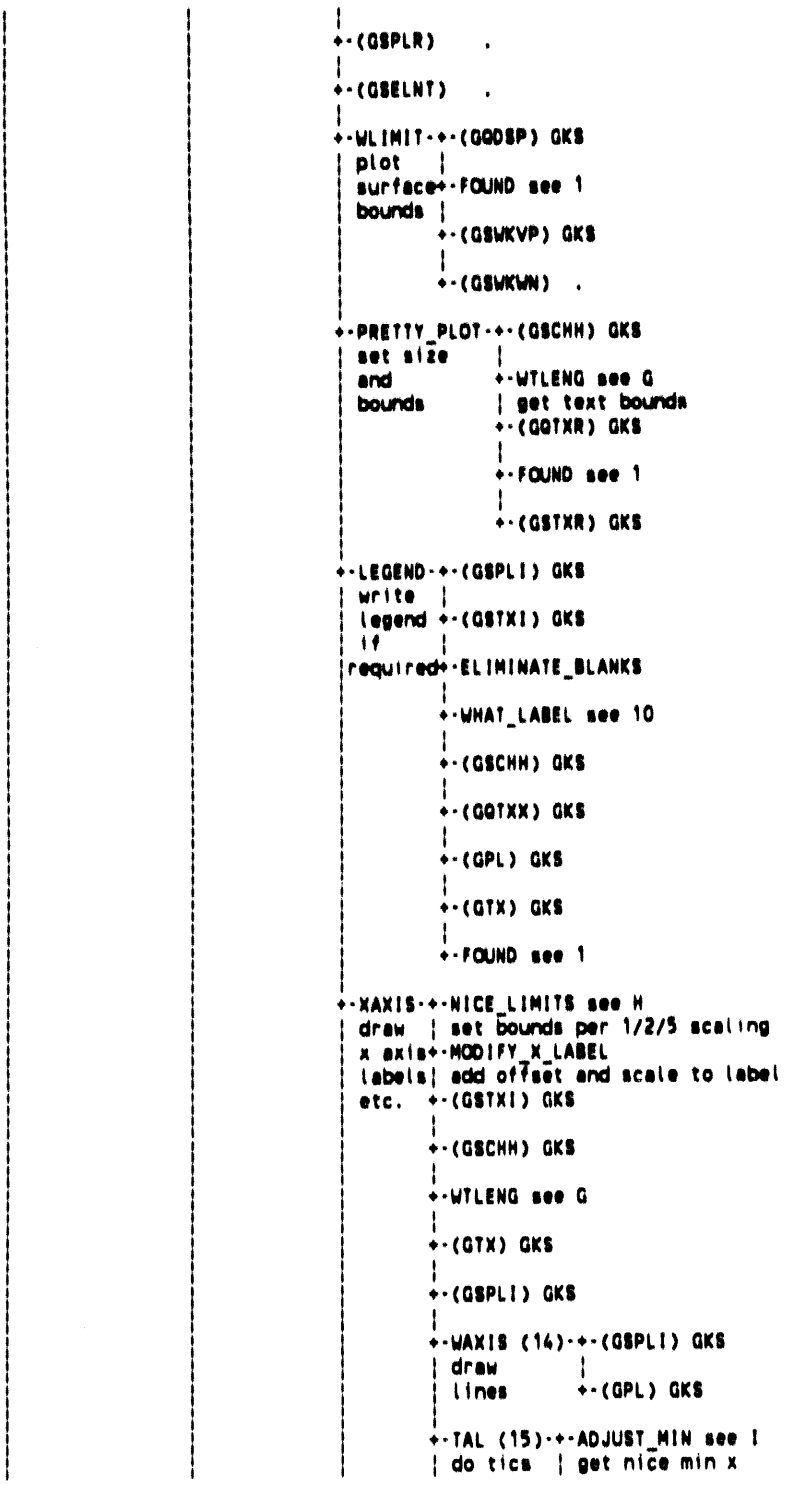

SANDUS ANALOG MUX data
chan data
+-FOUND see 1
+-SHO_ERROR
+-XTR_SMS-+-SPECIAL_STR
sub- channel data
+-FOUND see 1
+-SHO_ERROR
+-FOUND see 1
+-XTR_SMC--FOUND see 1
cal data (fatal)
+-XTR_TD-+-XTR_T-+-SPECIAL_STR
7912 data
chan data
+-SHO_ERROR
+-FOUND see 1
+-XTR_TS-+-SPECIAL_STR
sub- channel data
+-FOUND see 1
+-SHO_ERROR
+-XTR_TC--FOUND see 1
cal data
+-XTR_CD-+-XTR_C-+-SPECIAL_STR
7103 data
chan data
+-FOUND see 1
+-SHO_ERROR
+-XTR_CS-+-SPECIAL_STR
sub- channel
+-FOUND see 1
+-SHO_ERROR
+-XTR_CC--FOUND see 1
cal data
+-XTR_RD-+-XTR_R-+-SPECIAL_STR
RTD720 data
device data
+-FOUND see 1
+-SHO_ERROR
+-XTR_RC--FOUND see 1
channel data
+-XTR_RES-+-SPECIAL_STR
USE=0 sub- channel data
+-FOUND see 1
+-SHO_ERROR
+-XTR_RLS-+-SPECIAL_STR
USE=1 sub- channel
+-FOUND see 1

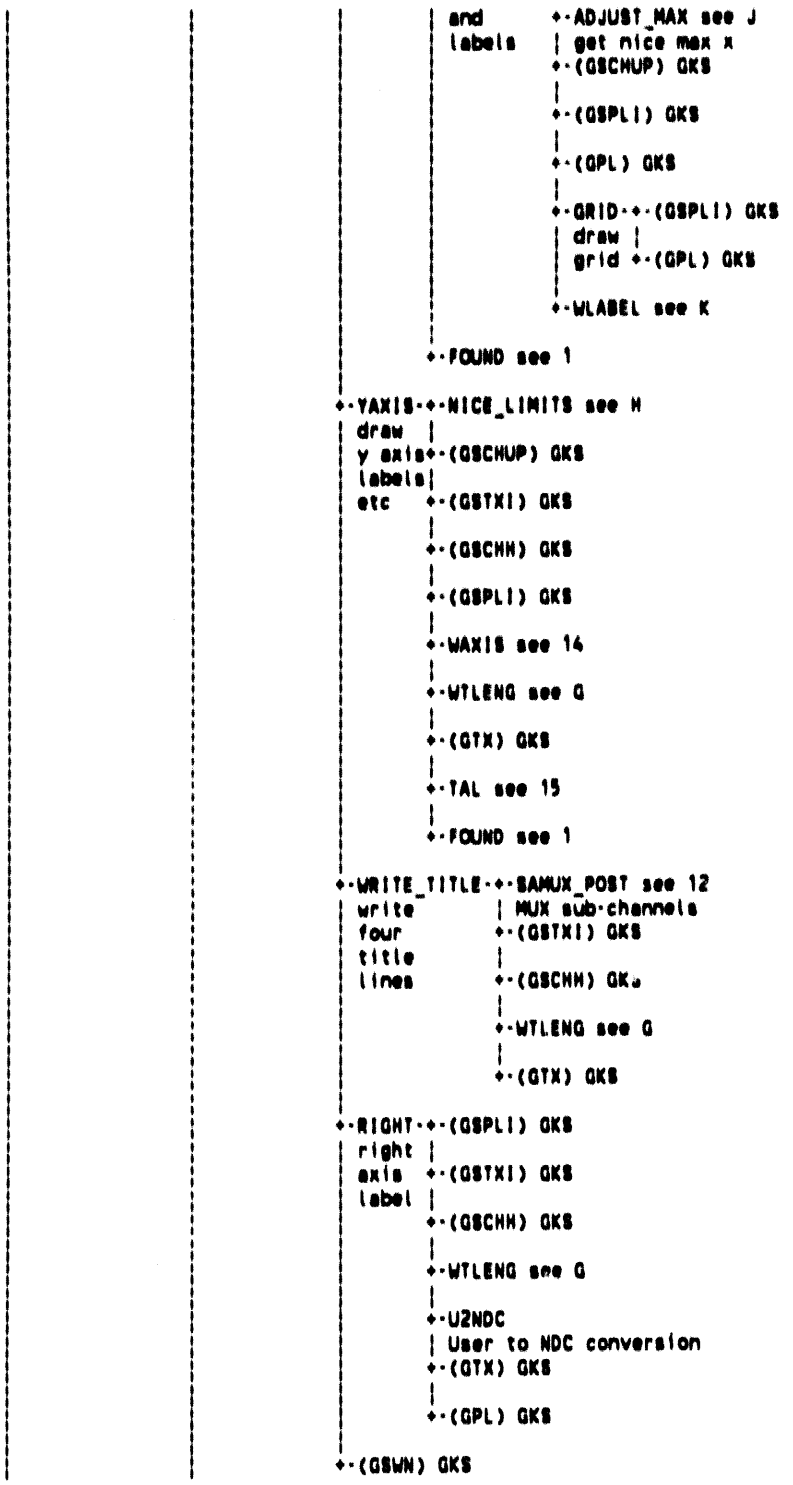

data +-SHO_ERROR
+-XTR_RCS-+-SPECIAL_STR
USE=2 |
sub-channel+-FOUND see 1
data |
+-SHO_ERROR
+-FOUND see 1
+-FILE_COMPS see 2
+-FOUND see 1
+-(SEND_NETWORK_MSG)
+-(CHANGE_COMM_SYM)
+-DEFAULT_HEADER-+-(LIB$DATE_TIME)
make sure |
header stuff +-FOUND see 1
has values
+-DEFAULT_PLOT make sure plot stuff
has values
+-CONJECTURE-+-FOUND see 1
convert |
header +-APPEND_ERROR form plot error line
info to | Title line 1
plot info +-ELIMINATE_BLANKS
|
+-SALOG-+-SALOG_CAL set for CAL plot
set |
for +-FOUND see 1
SANDUS|
+-ELIMINATE_BLANKS
|
+-YLIMITS given counts get
| plot units
+-YT_ONE_D (6)--POLEVAL see A
| Ytterbium conversion
+-YT_HYDRO (7)--POLEVAL see A
| Ytterbium conversion
+-TYPE_K (8)-+-FOUND see 1
| Thermo- |
| couple +-POLEVAL see A
| conversion |
+-TYPE_E (9)-+-FOUND see 1
| Thermo- |
| couple +-POLEVAL see A
| conversion 5th degree
+-SLIFER polynomial
| evaluation
+-ORIGIN_OFFSET
| Weird data?
+-WHAT_LABEL (10)--FOUND see 1
| Match label to plot
+-LOR (11)-+-@CF (function
label on | argument)
right +-FOUND see 1
|
+-SDIG-+-FOUND see 1

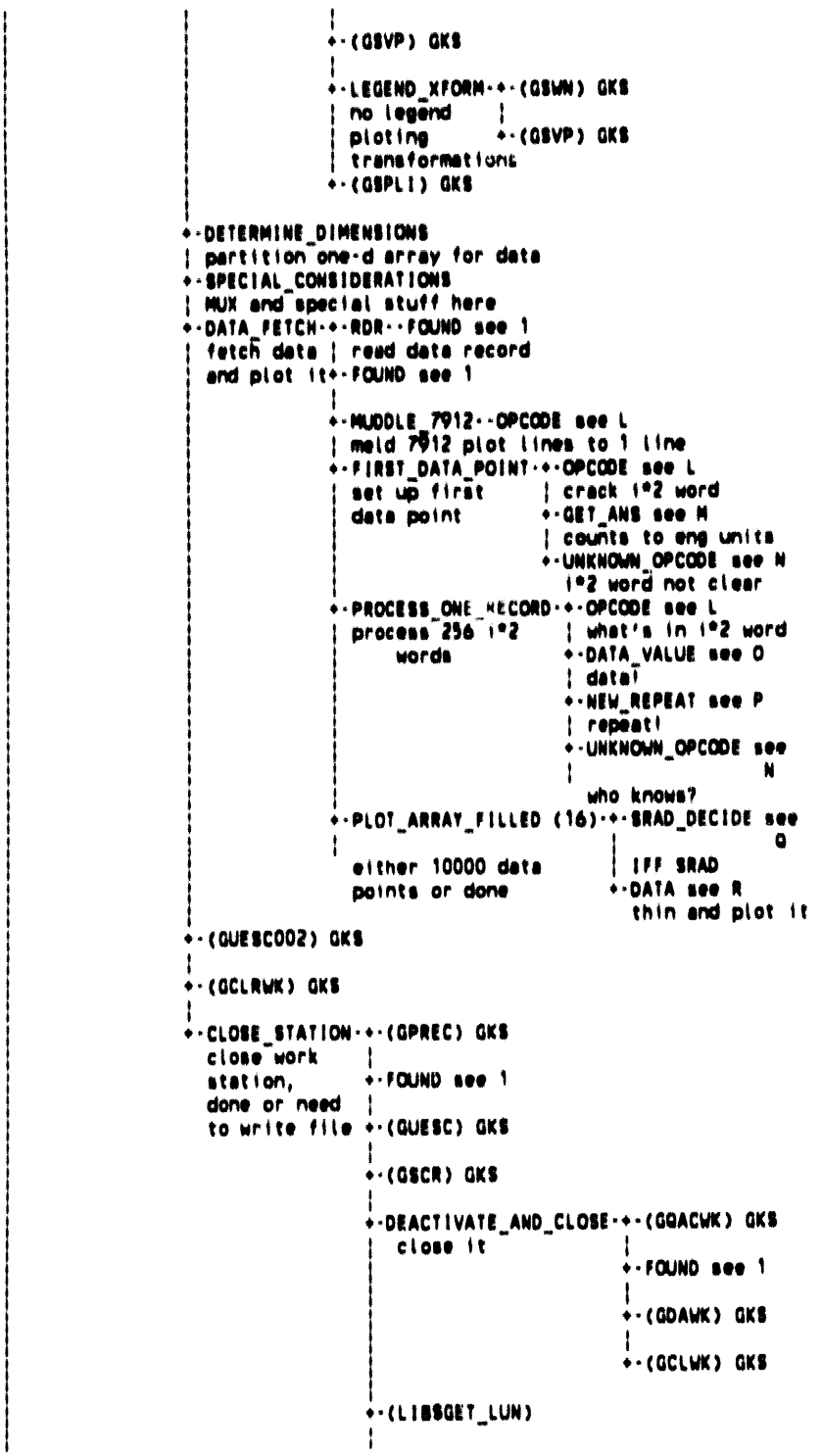

Dig. plot set up
SDIG_CAL--FOUND see 1
you're dead
ELIMINATE_BLANKS
ORIGIN_OFFSET
WHAT_LABEL see 10
SAMUX MUX plot setup
SAMUX_CAL--FOUND see 1
you're dead
FOUND see 1
ELIMINATE_BLANKS
YLIMITS
YT_ONE_D see 6
YT_HYDRO see 7
SAMUX_POST (12) IFF one sub-channel/page
ELIMINATE_BLANKS
YLIMITS
YT_ONE_D see 6
YT_HYDRO see 7
FOUND see 1
ORIGIN_OFFSET
WHAT_LABEL
T7912 7912 plot set-up
(STR$UPCASE)
ELIMINATE_BLANKS
FOUND see 1
T7912_CAL 7912 cal plot set-up
WHAT_LABEL see 10
T7912_CAL_EARLY see B
T48 files here
T7912_CAL_LATE see C
7912 cal setup here
LOR see 11
CAL_DATA_PROVIDED see D
calibrate real data
WHAT_LABEL see 10
CCD 7103 plot set-up
CCD_CAL--FOUND see 1
7103 cal plot setup
ELIMINATE_BLANKS
YLIMITS
FOUND see 1


+-WHAT_LABEL see 10
+-RTDS-+-RTD720_CAL--FOUND see 1
RTD720| You're dead
plot +-ELIMINATE_BLANKS
setup |
+-YLIMITS
+-FOUND see 1
+-ORIGIN_OFFSET
+-WHAT_LABEL see 10
+-GET_IGA--FOUND see 1
evaluates IGA for use in
+-(GQOPS) GKS GET_ANS
+-OPEN_STATION-+-LOOKS_OK
find current | check on plt file
workstation +-INQUIRE_STATION--FOUND see 1
and open it | what workstation?
+-SET_FILE_NAME (13)--FOUND see 1
| plot file filename
+-FOUND see 1
+-(GUESC050) GKS calls
+-(GOPWK)
+-(GUESC302)
+-(GACWK)
+-(GSDS)
+-(GUESC051)
+-(GPREC)
+-(GUESC)
+-SRAD_HEADER-+-SRAD_ANALOG-+-LOOKS_OK
IFF SRAD | SANDUS | check file name
flag set | ANALOG +-SET_FILE_NAME see 13
write | file | SRAD file name
file | header +-FOUND see 1
header +-(DUMP_GEN_DESC) G dump
+-(DUMP_GENERAL_CHAN) GC dump
+-(DUMP_S_ALOG_CHAN_DESC)
SA dump
+-(DUMP_CAL_S_ALOG)
SA_CAL dump
+-SRAD_MORE see E
Missing data?
+-SRAD_7912-+-LOOKS_OK
7912


header
dump
+-SET_FILE_NAME see 13
+-FOUND see 1
+-(DUMP_GEN_DESC)
+-(DUMP_GENERAL_CHAN)
+-(DUMP_T7912_CHAN_DESC)
T dump
+-(DUMPI_T7912_SUB_DESC)
TS dump
+-(DUMP_CAL_7912)
T_CAL dump
+-SRAD_7912_MORE see F
Multiple plot lines!
+-SRAD_T7103-+-LOOKS_OK
7103
header
dump
+-SET_FILE_NAME see 13
+-FOUND see 1
+-(DUMP_GEN_DESC)
+-(DUMP_GENERAL_CHAN)
+-(DUMP_T7103_CHAN_DESC)
C dump
+-(DUMPI_T7103_SUB_DESC)
CS dump
+-(DUMP_CAL_T7103)
C_CAL dump
+-SRAD_MORE see E
+-SRAD_RTD720-+-LOOKS_OK
RTD720
header
dump
+-SET_FILE_NAME see 13
+-FOUND see 1
+-(DUMP_GEN_DESC)
+-(DUMP_GENERAL_CHAN)
+-(DUMP_RTD_DEV_DESC)
R dump
+-(DUMP_RTD_CHAN_DESC)
RC dump
+-(DUMPI_RTD_EXP_SUB_DESC)
RES dump
+-(DUMPI_RTD_LC_SUB_DESC)
RLS dump
+-(DUMPI_RTD_CC_SUB_DESC)
RCS dump
+-(DUMP_CAL_RTD)
R_CAL dump
+-SRAD_MORE see E
+-BUILD_PLOT-+-(GSASF) GKS stuff
layout
plot
+-(GSTXR)


+-(GSPLR) .
+-(GSELNT) .
+-WLIMIT-+-(GQDSP) GKS
plot |
surface+-FOUND see 1
bounds |
+-(GSWKVP) GKS
+-(GSWKWN) .
+-PRETTY_PLOT-+-(GSCHH) GKS
set size |
and +-WTLENG see G
bounds | get text bounds
+-(GQTXR) GKS
+-FOUND see 1
+-(GSTXR) GKS
+-LEGEND-+-(GSPLI) GKS
write |
legend +-(GSTXI) GKS
if |
required+-ELIMINATE_BLANKS
+-WHAT_LABEL see 10
+-(GSCHH) GKS
+-(GQTXX) GKS
+-(GPL) GKS
+-(GTX) GKS
+-FOUND see 1
+-XAXIS-+-NICE_LIMITS see H
draw | set bounds per 1/2/5 scaling
x axis+-MODIFY_X_LABEL
labels| add offset and scale to label
etc. +-(GSTXI) GKS
+-(GSCHH) GKS
+-WTLENG see G
+-(GTX) GKS
+-(GSPLI) GKS
+-WAXIS (14)-+-(GSPLI) GKS
draw |
lines +-(GPL) GKS
+-TAL (15)-+-ADJUST_MIN see I
do tics | get nice min x


and labels
ADJUST_MAX see J
get nice max x
(GSCHUP) GKS
(GSPLI) GKS
(GPL) GKS
GRID
draw grid
(GSPLI) GKS
(GPL) GKS
WLABEL see K
FOUND see 1
YAXIS
draw y axis labels etc
NICE_LIMITS see H
(GSCHUP) GKS
(GSTXI) GKS
(GSCHH) GKS
(GSPLI) GKS
WAXIS see 14
WTLENG see G
(GTX) GKS
TAL see 15
FOUND see 1
WRITE_TITLE
write four title lines
SAMUX_POST see 12
MUX sub-channels
(GSTXI) GKS
(GSCHH) GKS
WTLENG see G
(GTX) GKS
RIGHT
right axis label
(GSPLI) GKS
(GSTXI) GKS
(GSCHH) GKS
WTLENG see G
U2NDC
User to NDC conversion
(GTX) GKS
(GPL) GKS
(GSWN) GKS


(GSVP) GKS
LEGEND_XFORM
no legend plotting transformations
(GSWN) GKS
(GSVP) GKS
(GSPLI) GKS
DETERMINE_DIMENSIONS
partition one-d array for data
SPECIAL_CONSIDERATIONS
MUX and special stuff here
DATA_FETCH
fetch data and plot it
RDR
FOUND see 1
read data record
FOUND see 1
MUDDLE_7912
OPCODE see L
meld 7912 plot lines to 1 line
FIRST_DATA_POINT
set up first data point
OPCODE see L
crack i*2 word
GET_ANS see M
counts to eng units
UNKNOWN_OPCODE see N
i*2 word not clear
PROCESS_ONE_RECORD
process 256 i*2 words
OPCODE see L
what's in i*2 word
DATA_VALUE see O
data!
NEW_REPEAT see P
repeat!
UNKNOWN_OPCODE see N
who knows?
PLOT_ARRAY_FILLED (16)
either 10000 data points or done
SRAD_DECIDE see Q
IFF SRAD
DATA see R
thin and plot it
(QUESCO02) GKS
(GCLRWK) GKS
CLOSE_STATION
close work station, done or need to write file
(GPREC) GKS
FOUND see 1
(QUESC) GKS
(GSCR) GKS
DEACTIVATE_AND_CLOSE
close it
(GDACWK) GKS
FOUND see 1
(GDAWK) GKS
(GCLWK) GKS
(LIBSGET_LUN)

```
|                                  +-PARSE_CLI--FOUND see 1
|                                  | split spawn cmd into pieces
|                                  +-(LIBSFREE_LUN)
|                                  |
|                                  +-(LIBSSPAWN)
|                                  |
|                                  +-SRAD_7912_DATA--FOUND see 1
|                                  | IFF SRAD & 7912 finish data
|                                  +-SRAD_CLOSE--FOUND see 1
|                                    IFF SRAD close SAD file
|
+-CLOSE_GKS--(GCLKS) GKS
| close GKS, all done
+-(SEND_NETWORK_MSG)
| tell Supermon we are done
+-(DISCONNECT_NETWORK_LINK)
| break Supermon communications
+-(CHANGE_COMM_SYM)
| tell RTD_SCHEDULER we're done
+-(LIBSENABLE_CTRL)
| Enable control Y
+-(EXIT)
 "CIAO babe"
```

This is a detached tree starting at the subprogram SHO_ERROR. SHO_ERROR writes a line pointing to an nonprinting character in the original string. This is a detached tree starting at the subprogram SANDUS

```
SANDUS-+-YT_ONE_D see 6  These are all SPECIAL conversions
       |                 of data beyond the A*y(counts)+B
       +-YT_HYDRO see 7  conversion.
       |
       +-TYPE_K see 8
       |
       +-TYPE_E see 9
       |
       +-SLIFER
```

This is a detached tree starting at the subprogram SPECIAL_STR. SPECIAL_STR works with SHO_ERROR. This is a detached tree starting at the subprogram XTR_R_CAL XTR_R_CAL--FOUND see 1 if at this point no cal files are known. This is a detached tree starting at the subprogram SCOPE plot bounds. This is a continuation tree starting at the subprogram POLEVAL

```
(A) POLEVAL--FOUND see 1
```

This is a continuation tree starting at the subprogram 'T7912_CAL_EARLY' (B) T7912_CAL_EARLY-+-PAGE1_LABELS special labels for T48 cals page 1

```
                           |
                           +-FOUND see 1
                           |
                           +-PAGE2_LABELS (17)--LOR see 11
                           | special labels for T48 cals, page 2
                           +-PAGE3_LABELS
                             special labels for T48 cals page 3
```

This is a continuation tree starting at the subprogram 'T7912_CAL_LATE' (C) T7912_CAL_LATE-+-PAGE1_LABELS as above but normal cals

```
                           |
                           +-FOUND see 1
                           |
                           +-PAGE2_LABELS see 17
                           |
                           +-PAGE3_LABELS
```

```
This is a continuation tree starting at the subprogram 'CAL_DATA_PROVIDED' (D)
CAL_DATA_PROVIDED-+-APPEND_ERROR adds to error title line if bad
                  |              cal info for data
                  +-FOUND see 1
                  |
                  +-NO_DUPES cal level labels on right axis
                  |
                  +-SORT_EM makes sure they are in order

This is a continuation tree starting at the subprogram 'SRAD_MORE' (E) SRAD_MORE--FOUND see 1.  This is a
continuation tree starting at the subprogram 'SRAD_7912_MORE' (F) SRAD_7912_MORE--FOUND see 1.  This is a
continuation tree starting at the subprogram 'WTLENG' (G) WTLENG-+-(GQCHUP) GKS
                  |
                  +-FOUND see 1
                  |
                  +-(GSCHUP) GKS
                  |
                  +-(GQTXP) GKS
                  |
                  +-(GSTXP) GKS
                  |
                  +-(GQTXX) GKS

This is a continuation tree starting at the subprogram 'NICE_LIMITS' (H) NICE_LIMITS-+-ORIGIN_OFFSET weird
data?
                  |
                  +-DATA_MAGNITUDE--FOUND see 1
                  | micro or mega?
                  +-MAJ_ANNOT--FOUND see 1
                  | major and minor divisions on plot
                  +-ADJUST_MIN see I
                  |
                  +-ADJUST_MAX see J

This is a continuation tree starting at the subprogram 'ADJUST_MIN' (I) ADJUST_MIN--FOUND see 1.  This is a
continuation tree starting at the subprogram 'ADJUST_MAX' (J) ADJUST_MAX--FOUND see 1.  This is a
continuation tree starting at the subprogram 'WLABEL' (K) WLABEL-+-(GSTXI) GKS
                  |
                  +-LAB_LENGTH--FOUND see 1
                  | # char positions for label
                  +-ELIMINATE_BLANKS
                  |
                  +-WTLENG see G
                  |
                  +-(GTX) GKS
                  |
                  +-FOUND see 1

This is a continuation tree starting at the subprogram 'OPCODE'

          (L) OPCODE-+-(LIBSEXTZV)
                     |
                     +-DECODE_8_BIT_EXT--(LIBSEXTZV) don't look here for help

This is a continuation tree starting at the subprogram 'GET_ANS'

          (M) GET_ANS-+-FOUND see 1
                      |
                      +-YT_ONE_D see 6  These are all special conversions
                      |                   beyond A*y+B
                      +-YT_HYDRO see 7
```

```
                    |
                    +-TYPE_K see 8
                    |
                    +-TYPE_E see 9
                    |
                    +-SLIFER

This is a continuation tree starting at the subprogram 'UNKNOWN_OPCODE' (N) UNKNOWN_OPCODE--FOUND see 1.
This is a continuation tree starting at the subprogram 'DATA_VALUE'

          (O) DATA_VALUE-+-GET_TIME (18)--FOUND see 1
                         | gets X value at current point
                         +-GET_ANS see M
                         | gets Y value at current point
                         +-PLOT_ARRAY_FILLED see 16
                         | either done or 10000 points
                         +-SAMP_RATE_CHANGE (19)--FOUND see 1
                           need to change sample interval

This is a continuation tree starting at the subprogram 'NEW_REPEAT'

          (P) NEW_REPEAT-+-FILL_GAP-+-GET_TIME see 18
                         | make plot|
                         | look OK  +-PLOT_ARRAY_FILLED see 16
                         |
                         +-SAMP_RATE_CHANGE see 19

This is a continuation tree starting at the subprogram 'SRAD_DECIDE'

          (Q) SRAD_DECIDE-+-SRAD_MORE see E
                          |
                          +-SRAD_DATA--FOUND see 1

This is a continuation tree starting at the subprogram 'DATA'

          (R) DATA-+-(GSPLCI) GKS
                   |
                   +-THINN--LOOKN
                   | thin the data to the workstation limits
                   +-QRAD_DATA--FOUND see 1
                   | like SRAD only shorter and faster
                   +-(GSELNT) GKS
                   |
                   +-(GPL) GKS
```

6.0 RTDTEST MAINTENANCE INFORMATION

6.1 Purpose

This maintenance document describes the RTDTEST software system that manages Tektronix RTD720 Digitizers (RTDs) using Digital Equipment Corporation (DEC) computers and workstations.

6.2 RTDTEST Functions

The functions of the instrument control software RTDTEST allow the user to modify RDT720 setups, modify ICFs, and indicate to the INGRES software to update the data base. RTDTEST allows interactive control to acquire, analyze, and display (plot) dry run data at the terminal. Additional functions RTDTEST provides are to send commands to an RTD720, obtain status information, and calibrate the RTD720.

6.2.1 Set up the RTD from the ICF

This option allows the operator to set up an RTD for Dry Run/Shot Configuration based on information residing in the INGRES data base. It accomplishes this by reading the ICF and sending the appropriate commands (ASCII strings) to an RTD. (Refer to 6.2.2 and 6.2.3)

6.2.2 Modify the ICF

The ICF can be modified by changing the device (RTD) settings according to the options for each function listed below.

Acquire:

a) Internal;
b) Length
c) Mode;
d) Clock;
e) Number of records; and
f) State

Trigger:

a) Source;
b) Coupling;
c) Slope;
d) Type Level;
e) Level;
f) Type position;
g) Position; and
h) Mode

Arming can be set to:

a) External, or
b) Internal

Channel settings can be modified according to the options for each function listed below.

Channel Settings:

a) Range,
b) Offset,
c) Type offset,
d) Coupling,
e) Band-width, and
f) Channel off/on

Sub Channel:

a) Left bound, and
b) Right bound

> Note: This for the first ten (1-10) subrecords in the INGRES data base. The left and right bound are used only by PRELEWD for plotting, and do not affect the RTD.

The ICF is used as the means to transport these changes to the INGRES data base if required. If not transported, the changes are used only for the current RTDTEST session (See 6.2.1 and 6.2.3).

6.2.3 Update INGRES Data Base

At the end of the current RTDTEST session and if an ICF was modified, the operator is asked if the modified ICFs should be sent to the INGRES data base for updating. If the response is Yes, the ICFs are copied from their present directory to an INGRES directory.

6.2.4 Acquire Data from an RTD

If an RTD has acquired data previously (acquisition cycle has completed - an Event code equal to 465), the operator can pull the data from the RTD and plot it on the terminal being used (Refer to 5.5).

6.2.5 Plot the Acquired Data to the Terminal

A two-dimensional plot of X,Y data is available if an RTD has acquired data and if that data has been written into a temporary file. The option in Section 6.2.4 has to be done first. A graphics terminal is required for this function.

6.2.6 Display RTD Set Up Information

This program queries a specific RTD720 for its current internal set up and displays the information on three FMS forms.[1] This program serves the same function as pressing the Utility button on the front panel and then pressing the View button (on the left side of the front panel) three times.

6.2.7 Send Individual Manual Command to an RTD.

The operator can send a command (ASCII string) to the RTD. However, a query command, a command with a "?", cannot be sent.

For example, the operator wants to change the number of channels available for data acquisition. If the current channel configuration is set to one (VMODE = CH1) and the two channel configuration is needed (VMODE = DUAL), the operator will type "VMODE DUAL" and press the return key.

> Note: The only checking done is for a query "?" command.
> However, if the command is typed incorrectly, it can be re-entered.

---

[1] The forms are identical to those shown in *The RTD720 Transient Digitizer Operator Manual* revised February 1991, pp. 3-125 through 3-126.

6.2.8 Calibrate an RTD

The operator can calibrate a specific RTD for a) Time, b) Trigger, c) Vertical, or d) All. In addition, the operator can set the RTD back to factory default settings.

6.2.9 Interface with CONTROL

RTDTEST is accessed through the CONTROL software. A complete description of CONTROL and its interface is discussed in the VAX-available document, RTDTPDWN.HT, by R. J. Isidoro of SNL Department 9321.

6.3 Program Function Descriptions

RTDTEST is a menu-driven system based on the DEC FMS. Only those responses found on the menus (forms) will be accepted. All other responses will be shown on the terminal as an error and give the operator infinite additional chances to respond correctly. Most menus where appropriate will allow the operator to Quit RTDTEST.

RTDTEST_driver. This program is the main module that controls all programs (subroutines) in RTDTEST (Refer to 6.5 Program Calling Tree).

Four major functions are performed. The program 1) obtains the name of the node where RTDTEST is currently executing; 2) creates and opens the log file with a name format of the Node, Date,and Time - (i.e., RTDTEST_RMV10_18JUN92_132421.log); 3) controls the logic flow of the its three subordinate programs a) initialize, b) process, and c) wrap up; and 4) closes the log file and writes the log file name to the terminal for the operators information.

RTDTEST_initialize. This program is executed only once during a session and performs the following four functions: 1) obtains the user name; 2) obtains the process name; 3) sets up the Forms Management System (FMS) and; 4) displays the WELCOME form on the terminal.

RTDTEST_Process. This program has three major functions: 1) the first time in the current session, it calls the program "RTDTEST_node," which in turn calls the program "RTDTEST_Get_Device_address"; 2) it displays the form MAIN on the terminal; and 3) it controls the logic for all further processing.

This is the program that contains the major processing loop and it keeps processing until the operator responds with a Quit.

The following selections can be chosen by this program:

(a) Select a bus (Normally two buses are available),
(b) Select one RTD720 digitizer from the ones available,
(c) ICF Management - With this option, certain fields can be modified in value content.
(d) Calibration - with this option a specific RTD720 can be calibrated,
(e) Status Information (RTD720 set up) can be viewed, and
(f) Miscellaneous Functions.

RTDTEST_wrap_up. This program is executed only once during a session and performs the following three functions: 1) displays the END form on the terminal; 2) closes down the FMS; and 3) de-assigns the channel reference to the GPIB bus interface.

Convert_error_code. This subroutine converts an error code into a readable message using a system call "Lib$sys_getmsg"

DEC_GPIB_QIOW_subs. This file is a collection of several subroutines used to access the DEC GPIB bus and, in general, devices attached to the bus. The subroutines are 1) Initialize the GPIB, 2) Set GPIB timeout, 3) Listen, 4) Unlisten, 5) Talk, 6) Untalk, 7) Clear, 8) Serial poll, 9) Recognize Event, 10) Set Event, 11) Remote enable, 12) Write to a device, 13) Read from a device (ASCII), and 14) Read from a device (Byte).

Extract_string. The purpose of this subroutine is to search an ASCII buffer for a certain string of characters and then extract the characters following that string delimited by ',' or ';' or ' '. The main purpose for this program is to decode queries (ASCII strings) returned from an RTD.

Find_node. The purpose of this subroutine is to find the computer node the program RTDTEST is currently running on by using a system call to "lib$sys_trnlog".

Get_chan_from_icf. The purpose of this subroutine is to read an ICF and copy specific channel information from an RTD720 into a temporary structure for use by subroutines in the RTDTEST system.

Get_device_from_icf. The purpose of this subroutine is to read an ICF and copy specific information from an RTD720 into a temporary structure for use by subroutines in the RTDTEST system.

Get_process. The purpose of this subroutine is to find the process name (ASCII string) using a system call to "lib$getjpi".

Get_subchan_from_icf. The purpose of this subroutine is to read an ICF and copy specific subchannel information from an RTD720 into a temporary structure for use by subroutines in the RTDTEST system.

Get_User. The purpose of this subroutine is to find the user's name (ASCII string) by using a system call to "lib$getjpi".

IOSB_error. This is a standard error handling routine to format and print the I/O status block (IOSB) from an error detected during a QIOW request. In addition, this subroutine calls the system routine to translate the IOSB(1) code into a readable description.

The Queue I/O request and wait (QIOW) service queues an I/O request to a channel associated with a specific device. For a complete definition, refer to the VAX/VMS System Service Reference Manual, April 1986, Software version: 4.4, pages SYS-277 through SYS-282.

The I/O status block (IOSB) receives the final completion status of the I/O operation. It has three fields: 1) condition value, 2) transfer count, and 3) device-specific information.

Modify_chan_on_ICF. The purpose of this subroutine is to put modified channel information back into the ICF (a specific channel record for a specific RTD device address).

Modify_device_on_ICF. The purpose of this subroutine is to put modified device information back into the ICF (for a specific RTD device address).

Modify_subchan_on_ICF. The purpose of this subroutine is to put modified subchannel information back into the ICF (for a specific channel record and a specific RTD device address).

Notebook. Any time the operator has an option (on a form) to make an entry into Notebook, this program 1) displays the Notebook form on the terminal and 2) copies the comments entered by the operator on the form and writes them to the permanent file, Notebook.fil This file can be viewed later by using a source editor, or the file can be printed.

Query_RTD720. This program accepts an ASCII RTD query command and sends it to the specified RTD. Its second function is to read the response from the RTD and send the ASCII message back to the terminal for the operator to view.

RTDTEST_Get_Device_Address. This program has the following three functions: 1) determines which bus form is required and displays that form on the terminal, 2) edits the operator response based on which node or bus was selected, and 3) passes back the GPIB bus address for the RTD in question (i.e., 1, 2, ...30).

RTDTEST_ICF_modification. This program has the following functions: 1) displays the form MOD on the terminal; 2) allows the operator to select the record to modify, that is, device attributes or channel/subchannel information, depending upon the "use"; 3) displays the USE form on the terminal - i) USE = 0 means data, ii) USE = 1 means cable compensation, and iii) USE = 2 means laser calibration; 4) display the SEL form on the terminal allowing the operator to pick a channel for modification and then displays the CHAN form (If a device was selected to be modified the program displays the DEV form); 5) depending upon the selection, the following programs are called: for device modification - Get_device_from_ICF and Modify_device_on_ICF, for channel subchannel modification - Get chan_from_ICF, Modify_chan_ on_ICF and Modify_subchan_on_ICF.

RTDTEST_ICF_management. This program displays the form ICF on the terminal and allows the operator to select the following functions: a) ICF Modification, b) update an RTD from the ICF, c) plot data stored in an RTD if available, and d) make an entry to the Notebook.

RTDTEST_calibration. This program will display the CAL form on the terminal, and allow the operator to calibrate a specific RTD by using the STANDARDIZE commands: a) Time, b) Trigger, c) Vertical, or d) All, and SAFEGUARD (Factory Default) command.

RTDTEST_display_plot. This program generates a two-dimensional plot of X,Y data obtained from an RTD if data was available. A graphics terminal is required for this function.

RTDTEST_manage_plot. This program performs the following functions: 1) calls RTDTEST_get_data (written by R. B. Caudell), 2) calls RTDTEST_THINN (written by P. Kaestner) that thins (compresses) the data to fit the number of terminal rasters, and 3) calls RTDTEST_display_plot (modified version of a plot program written by J. Lee) that displays the data graphically on the terminal.

RTDTEST_manual_command. This program displays the COMM form on the terminal and allows the operator to send a command to a specific RTD. The only editing this program performs on the command is to check for a '?' or query. No queries can be done with this program.

RTDTEST_Node. This program has the following functions:

(1) It displays one of several forms, depending on the node

(a) RMV10,
(b) RMV20, or
(c) RMV30

These forms allow the operator to choose one of two buses (EKA0 or EKA1);

(2) It calls RTDTEST_bus, which passes back to this program the device address (i.e., 1, 2, or .... 30); and

(3) It passes back the GPIB port (bus) either EKA0 or EKA1.

RTDTEST_pull_data. The purpose of this program is to get data from a specific RTD. The functions of this program are to retrieve data from an RTD if it has completed its acquisition cycle and to retrieve the Preamble.

RTDTEST_set_up_RTD. This program handles the communication between the DEC VLC 4000 Workstation (RMV10 and others) and any one of several RTDs attached to its GPIB bus. It takes information passed to it from the ICF. This sets up an RTD for Dry Run/Shot Configuration by translating the information into ASCII strings (the language of the RTD) and sending the commands through the GPIB bus to the RTD.

RTDTEST_set_up_info. This program has the following functions: (1) calls Query_RTD several times to obtain status information from an RTD, (2) calls Extract_string more than one hundred times, and (3) Prints the RTD set up information (three forms are required).

RTDTEST_set_up_communication. This program (1) de_assigns the assigned channel that handles communication the VAX's IEx communication port and the RTDTEST programs, if it is not the first time through in this session, and (2) sets up the necessary communication between the DEC VLC

4000 Workstation/EKx communication port attached and the RTDs. Refer to the source code for additional information.

Send_command_to_RTD. This program sends one (ASCII string) command to a specific RTD and no editing is done. (You get what you type.)

Str_Length. Str_Length.for finds the unpadded length of a character string and returns it as the function value. The unpadded length is not necessarily the same as the length returned by the built-in Len function. (This function was written by Jonathan P. Anspach, EG&G Energy Measurements, Inc., in support of Sandia National Laboratories.)

Table_modified. This integer function first tests to see if it is running on one of the nodes ACEnn, RMVj0, or SANmmm. If not, it returns a status value of 0. No further action occurs.

The code uses system service SYS$FILESCAN to parse FILE_SPEC for the file name, which is stored in FILE_NAME. It then checks directory TABLE_DIR for a file named FILE_NAME.TBL (or named FILE_NAME.DAT if a SANDUS or SANDACE node is used). If there is no such file, the routine returns a status value of 4. No further action occurs.

If FILE_NAME.TBL (FILE_NAME.DAT if a SANDUS or SANDACE node is used) exists, the routine uses run-time library routine LIB$DELETE_SYMBOL to delete any local symbol with the name FILE_NAME. Any error return from LIB$DELETE_SYMBOL is passed back to the caller.

If deletion of any local symbol was successful, the code uses the run-time library routine LIB$SET_SYMBOL to create the global symbol FILE_NAME with a value of FILE_NAME. An error return from LIB$SET_SYMBOL is passed back to the calling program.

6.4 Command Procedures

The following two sections describe the verbatim command procedures for (1) compiling and linking RTDTEST or (2) linking only RTDTEST.

6.4.1 Compile and Link RTDTEST

```
$!
$!          ---------------------------------
$!          File name: RTDTEST_compile_link.com
$!          ---------------------------------
$!
$  set verify
$!
$ For  Convert_error_code
$ For  DEC_GPIB_QIOW_subs
$ For  Extract_string
$ For  Find_node
$ For  Get_chan_from_ICF
$ For  Get_device_from_ICF
$ For  Get_process
$ For  Get_subchan_from_ICF
$ For  Get_user
$ For  IOSB_error
$ For  Modify_chan_on_ICF
$ For  Modify_device_on_ICF
$ For  Modify_subchan_on_ICF
$ For  Notebook
$ For  Query_RTD720
$ For  RTDTEST_Get_Device_Address
$ For  RTDTEST_ICF_modification
$ For  RTDTEST_ICF_management
$ For  RTDTEST_calibration
$ For  RTDTEST_driver
$ For  RTDTEST_display_plot
$ For  RTDTEST_initialize
$ For  RTDTEST_misc_functions
$ For  RTDTEST_manage_plot
$ For  RTDTEST_manual_command
$ For  RTDTEST_node
$ For  RTDTEST_process
$ For  RTDTEST_pull_data
$ For  RTDTEST_set_up_RTD720
$ For  RTDTEST_set_up_info
$ For  RTDTEST_set_up_communication
$ For  RTDTEST_wrap_up
$ For  Send_command_to_RTD720
```

```
$ For  Str_length
$ For  Table_modified
$ For  Thinn
$!
$ Link  RTDTEST_driver,-
      Convert_error_code,-
      DEC_GPIB_QIOW_subs,-
      Extract_string,-
      Find_node,-
      Get_chan_from_ICF,-
      Get_device_from_ICF,-
      Get_process,-
      Get_subchan_from_ICF,-
      Get_user,-
      IOSB_error,-
      Modify_chan_on_ICF,-
      Modify_device_on_ICF,-
      Modify_subchan_on_ICF,-
      Notebook,-
      Query_RTD720,-
      RTDTEST_Get_Device_Address,-
      RTDTEST_ICF_modification,-
      RTDTEST_ICF_management,-
      RTDTEST_calibration,-
      RTDTEST_display_plot,-
      RTDTEST_initialize,-
      RTDTEST_misc_functions,-
      RTDTEST_manage_plot,-
      RTDTEST_manual_command,-
      RTDTEST_node,-
      RTDTEST_process,-
      RTDTEST_pull_data,-
      RTDTEST_set_up_RTD720,-
      RTDTEST_set_up_info,-
      RTDTEST_set_up_communication,-
      RTDTEST_wrap_up,-
      Send_command_to_RTD720,-
      Str_length,-
      Table_modified,-
      Thinn,-
      GKSEXTEND/LIB, -
      sys$input/option
```

```
        gksdir:gksflb.exe/share
        sys$share:vaxcrtl.exe/share,-
        LD:[Utility]utility/lib
$!
$  purge *.obj
$!
$  rename RTDTEST_driver.exe            RTDTEST.exe
$!
$  set noverify
$!
$!                  ---------------
$!                  End of .COM file
$!                  ---------------
```

6.4.2 Link Only RTDTEST

```
$!
$!              ------------------------------
$!              File name: RTDTEST_link_only.com
$!              ------------------------------
$!
$  set verify
$!
$ Link  RTDTEST_driver,-
        Convert_error_code,-
        DEC_GPIB_QIOW_subs,-
        Extract_string,-
        Find_node,-
        Get_chan_from_ICF,-
        Get_device_from_ICF,-
        Get_process,-
        Get_subchan_from_ICF,-
        Get_user,-
        IOSB_error,-
        Modify_chan_on_ICF,-
        Modify_device_on_ICF,-
        Modify_subchan_on_ICF,-
        Notebook,-
        Query_RTD720,-
        RTDTEST_Get_Device_Address,-
        RTDTEST_ICF_modification,-
```

```
        RTDTEST_ICF_management,-
        RTDTEST_calibration,-
        RTDTEST_display_plot,-
        RTDTEST_initialize,-
        RTDTEST_misc_functions,-
        RTDTEST_manage_plot,-
        RTDTEST_manual_command,-
        RTDTEST_node,-
        RTDTEST_process,-
        RTDTEST_pull_data,-
        RTDTEST_set_up_RTD720,-
        RTDTEST_set_up_info,-
        RTDTEST_set_up_communication -
        RTDTEST_wrap_up,-
        Send_command_to_RTD720,-
        Str_length,-
        Table_modified,-
        Thinn,-
        GKSEXTEND/LIB, -
        sys$input/option
        gksdir:gksflb.exe/share
        sys$share:vaxcrtl.exe/share,-
        LD:[Utility]utility/lib
$!


$ purge *.obj
$!
$ rename RTDTEST_driver.exe          RTDTEST.exe
$!
$!
$!              ----------
$!              End of .COM file
$!              ----------
$ set noverify

*1
        ---------------------------
        RTDTEST processing log start
        ---------------------------

(*) RTDTEST_driver start            15-DEC-1992 15:49:28.93
```

```
The logfile is:  ABQVAX::LD:[RTDTEST.LOGS]RTDTEST_RMV10_15DEC92_154928.log

    The logfile_lun:                    119

o RTDTEST_initialize start              15-DEC-1992 15:49:29.31
    User is:                          RCAUDELL
    Node is:                          RMV10
    Process is:                       RCAUDELL
  RTDTEST_initialize stop               15-DEC-1992 15:49:30.85


o RTDTEST_process start                 15-DEC-1992 15:49:30.87
    You selected option:              6 - Query Set Up Info

o RTDTEST_Node start                    15-DEC-1992 15:49:36.68
    You selected option:              3 - Bus 11
    The BUS (Port) ID passed is:        EKA0:
  RTDTEST_Node stop                     15-DEC-1992 15:49:51.55


o RTD720 set up communication start     15-DEC-1992 15:49:51.59
    The BUS (Port) ID passed is:        EKA0:
    The First_time_set_up is:         True
    The "Assign channel" is:          240
    The BUS (Port) ID passed is:        EKA0:
    The "Assign channel" is:          240
    Initialized GPIB bus to address:  0
  RTD720 set up communication stop      15-DEC-1992 15:49:51.63


o RTDTEST_Get_Device_Address start      15-DEC-1992 15:49:51.64
    You selected option:            A - RTD720 address 1
    You selected device address:      1
  RTDTEST_Get_Device_Address stop       15-DEC-1992 15:49:55.87


o RTDTEST_set_up_info start             15-DEC-1992 15:49:55.88
    Assign channel passed is:         240
    Port ID passed is:                EKA0:
    Device address passed is:         1
  RTDTEST_set_up_info stop              15-DEC-1992 15:50:17.15
```

```
      You selected option:              5 - Calibration

   o RTDTEST_calibration start             15-DEC-1992 15:50:26.77
      You selected option               3  Calibrate Time

*1
   o Send_command_to_RTD720 start          15-DEC-1992 15:50:33.46
      Command sent to the RTD720 is:      STANDARDIZE TIME;
    Send_command_to_RTD720 stop           15-DEC-1992 15:50:33.48

 Calibrating TIME complete     PASS                Press Return
      You selected option               2  Previous Menu
    RTDTEST_calibration stop              15-DEC-1992 15:50:49.69

      You selected option:              7 - Miscellaneous Functions

   o RTDTEST_misc_functions start          15-DEC-1992 15:51:09.29
      You selected option               2 - Previous Menu
    RTDTEST_misc_functions stop           15-DEC-1992 15:51:14.17

      You selected option:              4 - ICF Management

   o RTDTEST_ICF_management start          15-DEC-1992 15:51:20.76
      You selected option:              2 - Previous Menu
    RTDTEST_ICF_management stop           15-DEC-1992 15:51:28.96

      You selected option:              1 - Quit

    RTDTEST_process stop                 15-DEC-1992 15:51:40.88


   o RTDTEST_wrap_up start                15-DEC-1992 15:51:40.88
      De_assigned GPIB_channel           240
    RTDTEST_wrap_up stop                  15-DEC-1992 15:51:43.11


(*) RTDTEST_driver stop                  15-DEC-1992 15:51:43.11

                  ----------------------------
                  RTDTEST processing log finish
                  ----------------------------
*1
```

6.5 Program Calling Tree

The following pages illustrate the hierarchical RTDTEST programs relationship to each other. Calls to get date/time, get system assigned logical unit numbers, or any Forms Management System calls are not shown in this Tree.

```
*1
RTDTEST_DRIVER
     |
     +-FIND_NODE
     |
     +-RTDTEST_INITIALIZE
     |              +-GET_USER
     |              |
     |              +-GET_PROCESS
     +-RTDTEST_PROCESS
     |            +-NOTEBOOK
     |            |
     |            +-RTDTEST_NODE
     |            |
     |            +-RTDTEST_SET_UP_COMMUNICATION (A)
     |            |
     |            +-RTDTEST_GET_DEVICE_ADDRESS
     |            |
     |            +-RTDTEST_ICF_MANAGEMENT
     |            |                +-RTDTEST_ICF_MODIFICATION (B)
     |            |                |
     |            |                +-GET_DEVICE_FROM_ICF (C)
     |            |                |
     |            |                +-GET_CHAN_FROM_ICF (D)
     |            |                |
     |            |                +-RTDTEST_SET_UP_RTD720 (E)
     |            |                |
     |            |                +-RTDTEST_MANAGE_PLOT (F)
     |            |                |
     |            |                +-NOTEBOOK
     |            +-RTDTEST_CALIBRATION
     |            |              +-NOTEBOOK
     |            |              |
     |            |              +-SEND_COMMAND_TO_RTD720 (G)
     |            |              |
```

```
|             |                +-QUERY_RTD720 (H)
|             +-RTDTEST_SET_UP_INFO
|             |                +-QUERY_RTD720 (H)
|             |                |
|             |                +-EXTRACT_STRING
|             |
|             +-RTDTEST_MANUAL_COMMAND
|                              +-NOTEBOOK
|                              |
|                              +-SEND_COMMAND_TO_RTD720 (G)
|
+-RTDTEST_WRAP_UP
|             +-TABLE_MODIFIED
```

*1

(A) RTDTEST_SET_UP_COMMUNICATION

```
                    +-IOSB_ERROR-+-CONVERT_ERROR_CODE
                    |
                    +-TIMEOUT
                    |
                    +-UNLISTEN
                    |
                    +-UNTALK
                    |
                    +-REMOTE_ENABLE
                    |
                    +-SET_EVENT
```

(B) RTDTEST_ICF_MODIFICATION

```
                   +-GET_DEVICE_FROM_ICF
                   |                +-CLOSE_TBL
                   |                |
                   |                +-OPEN_TBL
                   |                |
                   |                +-READ_TBL_REC
                   |
                   +-MODIFY_DEVICE_C   'CF
                   |                +-CLOSE_TBL
                   |                |
                   |                +-OPEN_TBL
                   |                |
```

```
|                +-READ_TBL_REC
|                |
|                +-MODIFY_TBL_REC
+-GET_CHAN_FROM_ICF (D)
|
+-GET_SUBCHAN_FROM_ICF
|                +-CLOSE_TBL
|                |
|                +-OPEN_TBL
|                |
|                +-READ_TBL_REC
+-MODIFY_CHAN_ON_ICF
|               +-CLOSE_TBL
|               |
|               +-OPEN_TBL
|               |
|               +-READ_TBL_REC
|               |
|               +-MODIFY_TBL_REC
|
+-MODIFY_SUBCHAN_ON_ICF
                +-CLOSE_TBL
                |
                +-OPEN_TBL
                |
                +-READ_TBL_REC
                |
                +-MODIFY_TBL_REC
```

```
*1
(C) GET_DEVICE_FROM_ICF
                +-CLOSE_TBL
                |
                +-OPEN_TBL
                |
                +-READ_TBL_REC

(D) GET_CHAN_FROM_ICF
              +-CLOSE_TBL
              |
              +-OPEN_TBL
```

```
                |
                +-READ_TBL_REC

(E) RTDTEST_SET_UP_RTD720
                  +-STR_LENGTH
                  |
                  +-LISTEN
                  |
                  +-IOSB_ERROR
                  |
                  +-WRT_RTD720

(F) RTDTEST_MANAGE_PLOT
                +-NOTEBOOK
                |
                +-RTDTEST_PULL_DATA
                |               +-UNLISTEN
                |               |
                |               +-IOSB_ERROR
                |               |
                |               +-UNTALK
                |               |
                |               +-LISTEN
                |               |
                |               +-WRT_RTD720
                |               |
                |               +-TALK
                |               |
                |               +-READ_RTD720
                |               |
                |               +-READ_RTD720_BIG
                |               |
                |               +-QUERY_RTD720 (H)
                |
                +-EXTRACT_STRING
                |
                +-THINN--LOOKN
                |
                +-RTDTEST_DISPLAY_PLOT
                                +-AUTO_SCALE
                                |
                                +-DEFINE_XFORMS
```

```
*1
(G) SEND_COMMAND_TO_RTD720
                    +-UNLISTEN
                    |
                    +-IOSB_ERROR
                    |
                    +-UNTALK
                    |
                    +-LISTEN
                    |
                    +-WRT_RTD720

(H) QUERY_RTD720
              +-LISTEN
              |
              +-IOSB_ERROR
              |
              +-WRT_RTD720
              |
              +-TALK
              |
              +-READ_RTD720
```

APPENDIX A

INSTRUMENT CONTROL FILE FOR TEKTRONIX RTD 720 DIGITIZERS

APPENDIX A
INSTRUMENT CONTROL FILE FOR TEKTRONIX RTD 720 DIGITIZERS

A.1 INTRODUCTION

At the Nevada Test Site (NTS), Sandia National Labs (SNL) utilizes computer control from a network of VAX and microVAX computers to perform Tektronix RTD 720 data acquisition and recording. Each experiment fielded is accompanied by a large, software accessible volume of technical (such as digitizer setup) and administrative (such as name, organization, measurement) information about the digitizer and the experiment. The repository for this information is an INGRES-based instrumentation database on VAX computer node N12DBM. The database format provides a convenient method to define experiments, assign digitizers to experiments, initially configure digitizers, and generate reports throughout the event sequence. However, data-acquisition and reduction programs that execute on other nodes of the network often require database information in a different format.

Instrument Control Files (ICF) are the mechanism by which the instrumentation database, digitizer data-acquisition systems, and data-recording and analysis systems communicate with one another. Digitizer setup information is passed to the database via the ICF. Administrative and technical information is passed from the database to the data recording and analysis programs via the ICF. Each ICF is customized to the unique needs and definition of a particular digitizing/recording system. For the purposes of this document, the discussion will be limited to the Instrument Control Files associated with the Tektronix RTD 720 digitizer systems.

It should be noted that the ICF is a defined interface between the database and data-acquisition/reduction programs. There is nothing unique about the relationship of the ICF to the INGRES database system. On that basis, any database software or other programmatic effort capable of creating and reading files that meet ICF specifications could be used in place of INGRES.

A.2 HISTORY

Instrument Control Files (ICF) have been known by a variety of names during previous field-test events. The most prevalent name has been "Instrumentation Table" and indeed, there is much documentation in existence that will continue to use this name for years to come. Directory and file names throughout the system still reflect the use of the word "table."

Historically, during the early years, the instrumentation table did indeed contain all of the "database" information for a given system. With the introduction of the INGRES-based database into the system and new table definitions [1], the instrumentation table was no longer the "database" for a given system. Definition of this new file consisted of instrument setup information, a limited amount of administrative information, and virtually no information with regard to cableplant measurements, simulation, and signal timing.

Beginning with the Distant Zenith event (1991), the effort to rename "Instrument Table" to "Instrument Control File" began. The goal was to eliminate the word "table" from the file name, because the word "table" in the database world is a totally different concept than the historical use of the word for instrument control files. As with any such effort requiring ingrained thought patterns to change, the effort has met with limited success, and still has a long way to go before achieving universal acceptance. Therefore, while this document will attempt to carefully use the ICF concept, it is quite possible that the ICF will accidently be called by the word "table" in oversight.

A.3 INSTRUMENT CONTROL FILE OVERVIEW

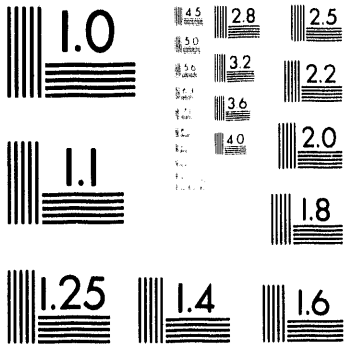
An Instrument Control File (ICF) consists of a collection of records that are applicable to the type of digitizing/recording system. Thus, the ICF for a SANDUS system is different from the ICF for an ACE (Tektronix 7912) system, and these in turn are different from the ICF for a RTD 720 system. Each ICF record structure is unique and defined via an "Include File" by the name of TABLE_STRUCTS.DEF. The ICF is implemented as a standard RMS keyed-access indexed file containing variable length records. This file organization was chosen [1] because it allows both random access and true variable length records.

Currently, TABLE_STRUCTS.DEF is sized to accommodate 40 different structure definitions. However, not all 40 structures are defined at this time. There are a number of embedded spare placeholders, and structures 35 through 40 are available for future expansion. The first four bytes of each record structure are reserved for a numeric key that is used to index into the file.

Definition of these bytes will be discussed in greater detail in Section 4. The following structures are currently defined:

Structure Name	Record Type	Expanded Description
STRUCT_DESC	1	Structure
GEN_DESC	2	General
STREAM_DESC	3	Stream
MM_DESC	4	Mass Memory
HDDR_DESC	5	High Density Digital Recorder
GEAR_DESC	6	General Purpose Interface Bus (GPIB) ethernet Analog Recording
SOURCE_DESC	7	Source
SIMULATOR_DESC	12	Simulator
S_TRIG_DESC	13	SANDUS Trigger
PRI_TYPE_DESC	17	Primary Type
TRIGGER_CHAN_DESC	18	Trigger Channel
S_ALOG_CHAN_DESC	19	SANDUS Analog Channel
S_DIG_CHAN_DESC	20	SANDUS Digital Channel
S_DIG_SUB_DESC	21	SANDUS Digital Subchannel
S_AMUX_CHAN_DESC	22	SANDUS Analog Multiplexer
S_AMUX_SUB_DESC	23	SANDUS Analog Mux Subchannel
T7912_CHAN_DESC	24	Tek 7912 Channel
T7912_SUB_DESC	25	Tek 7912 Subchannel
T7103_CHAN_DESC	28	Tek 7103 Channel
T7103_SUB_DESC	29	Tek 7103 Subchannel
RTD_DEV_DESC	30	Tek RTD Device
RTD_CHAN_DESC	31	Tek RTD Channel
RTD_EXP_SUB_DESC	32	Tek RTD Experiment Subchannel
RTD_LC_SUB_DESC	33	Tek RTD Laser Cal Subchannel
RTD_CC_SUB_DESC	34	Tek RTD Cable Comp Subchannel

In an RTD 720 data acquisition system consisting of three RTD 720 digitizers (devices), with each digitizer configured to record two channels (DUAL mode) and each channel configured to record two (time multiplexed) signals (i.e., two subchannels), the collection of ICF records would be as follows:


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6

3 of 5

One STRUCT_DESC record.

One GEN_DESC record.

One GEAR_DESC record.

One PRI_TYPE_DESC record.

Three RTD_DEV_DESC records; Experiment configuration.
(1 record per device)

Three RTD_DEV_DESC records; Laser Cal configuration.
(1 record per device)

Three RTD_DEV_DESC records; Cable Comp configuration.
(1 record per device)

Six RTD_CHAN_DESC records; Experiment configuration.
(1 record per channel,
2 records per device)

Six RTD_CHAN_DESC records; Laser Cal configuration.
(1 record per channel,
2 records per device)

Six RTD_CHAN_DESC records; Cable Comp configuration.
(1 record per channel,
2 records per device)

Six RTD_EXP_SUB_DESC records; Experiment configuration; Subchannel 00
(1 record per channel,
2 records per device)

Twelve RTD_EXP_SUB_DESC records; Experiment configuration.
(1 record per subchannel,
2 records per channel,
4 records per device)

Six RTD_LC_SUB_DESC records; Laser Cal configuration; Subchannel 00
(1 record per channel,
2 records per device)

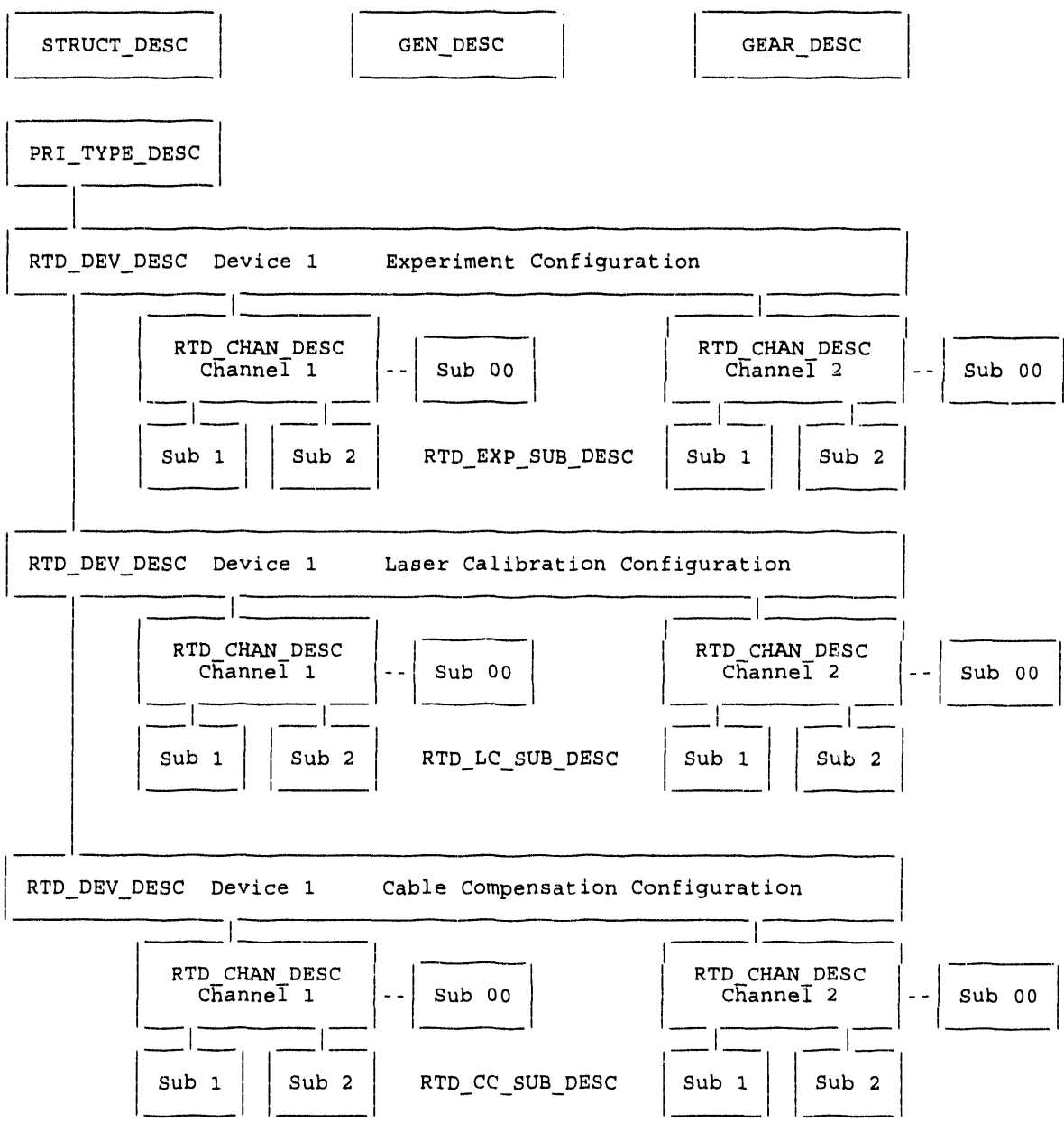
Six RTD_LC_SUB_DESC records; Laser Cal configuration.
(typically only 1 subchannel used,
1 record per subchannel,
1 record per channel,
2 records per device)

Six RTD_CC_SUB_DESC records; Cable Comp configuration; Subchannel 00
(1 record per channel,
2 records per device)

Twelve RTD_CC_SUB_DESC records; Cable Comp configuration.
(1 record per subchannel,
2 records per channel,
4 records per device)

for a total of 79 records for the system described.

In tree form, the ICF would appear as follows:



The records for Device 2 and Device 3 would duplicate those of Device 1.

The total number of records in the ICF for a given system will vary, depending upon system configuration. The number of devices, channels, and subchannels determines the number of each type of record. Under the current method of operation, only the database administrator can modify system configuration, upon direction from the Project Officer.

Subchannel 00 records are special; they direct the data analysis program to create a "CHN" file that contains all of the data recorded for the channel. Thus, when two or more time multiplexed signals are recorded on a given channel, the 00 CHN file will contain all of the signals, and a plot of this file will display all signals in time relationship. This is in contrast to the CHN files for the individual subchannels, which will contain only those data associated with the subchannel signal.

TABLE_STRUCTS.DEF continues to evolve from event to event. Historically, it has been reworked and customized for each UGT event. This accepted practice is necessary to accommodate the changing needs of events, experiments, and measurement techniques.

A complete listing of TABLE_STRUCTS.DEF for the Hunters Trophy event is included at the end of this Appendix.

Instrument Control Files follow a traditional naming convention established a number of years ago. An ICF for SANDUS 505 will be named "SAN505.TBL". An ICF for ACE 21 with Tektronix 7912 digitizers will be named "ACE21T12.TBL". And last, but not least, an ICF for RMV 11 with Tektronix RTD 720 digitizers will be named "RMV11R20.TBL".

A.4 STRUCTURE DETAIL

A.4.1 Record Type 1 /STRUCT_DESC/

This record is used to keep an internal description of the control file within the control file itself. It will normally only be used by the control file access routines.

The structure record number (struct_rec_num, bytes 1 and 2) are set to 0 for STRUCT_DESC, with record type (struct_rec_type, byte 3) set to 1. Byte 4 is an unused dummy field, set to 0.

```
integer*2    struct_rec_num
byte         struct_rec_type
byte         dummy
```

Currently, the structure is defined with space allocated to store up to 40 different record structure names; each name can be up to 24 characters long. This is followed by space to store the length in bytes of the 40 preceding named structures (in the same order). The actual number of record structures defined within TABLE_STRUCTS.DEF appears next.

```
character*24  rec_name(40)
integer*4     rec_len(40)
integer*4     tot_structs
```

The last field is allocated for the date/time of the last revision.

```
character*24  last_edit
```

There is always one, and only one, of this record type in every control file.

A.4.2 Record Type 2 /GEN_DESC/

This record is used to convey general information about the UGT event, software, and characteristics of the data source.

The structure record number (gen_rec_num, bytes 1 and 2) are set to 0 for GEN_DESC, with record type (gen_rec_type, byte 3) set to 2. Byte 4 is an unused dummy field, set to 0.

```
integer*2    gen_rec_num
byte         gen_rec_type
byte         dummy
```

The are eleven fields available for general information. A number of these fields are not applicable to RTD 720 type digitizers, but are included for standardization.

Space has been allocated to designate the test name (UGT event), up to a maximum of 24 alphanumeric characters. For example, the letters "Hunters Trophy" were used for the Hunters Trophy event. This is followed by an alphanumeric test number of up to eight characters, which is available to identify software applicable to the UGT. Sixteen characters are available to

identify the name of the data collection source. For Hunters Trophy names like "RMV 11", "RMV 12" were used. Four characters are available for the source code. For Hunters Trophy the code "R11" accompanied the source name RMV 11, "R12" accompanied the source name RMV 12.

```
character*24  test_name
character*8   test_number
character*16  source_name
character*4   source_code
```

The following fields are not applicable to RTD 720 digitizers and are defaulted to zero.

```
integer*4     num_streams
integer*4     ref_trig
integer*4     ref_trig_tol
integer*4     hddr_offset
integer*4     mm_offset
real*4        sec_per_bit
```

The last field is allocated for the date/time of the last revision.

```
character*24  last_edit
```

There is always one, and only one, of this record type in every control file.

A.4.3 Record Type 6 /GEAR_DESC/

This record is used to describe the GEAR data collection system. The structure record number (struct_rec_num, bytes 1 and 2) are set to 0 for GEAR_DESC, with record type (struct_rec_type, byte 3) set to 6. Byte 4 is an unused dummy field, set to 0.

```
integer*2     struct_rec_num
byte          struct_rec_type
byte          dummy
```

The only field currently in use in this record is the collection point node name. For the Hunters Trophy event, the name used was "GEAR 10".

```
character*16  col_name
```

The remaining character fields are filled with nulls, and the integer fields are defaulted to zeroes. These fields are provided for future expansion.

```
character*16  ether_port(2)
integer*4     %fill(4)
```

The last field is allocated for the date/time of the last revision.

```
character*24  last_edit
```

There is always one, and only one, of this record type in every control file.

A.4.4 Record Type 17 /PRI_TYPE_DESC/

This record identifies which channel numbers are assigned in this ICF and the type of digitizer associated with the channel. Only channel numbers that have been assigned in this record will have descriptive records in the ICF. PRI_TYPE_DESC is short for Primary (Digitizer) Type Description.

Theoretically, one could define a primary type description record with a mix of every type of digitizer for any of the available channels; from the practical viewpoint, ICF files have always been limited to one type of digitizer per file.

The structure record number (pri_rec_num, bytes 1 and 2) are set to 0 for PRI_TYPE_DESC, with record type (pri_rec_type, byte 3) set to 17. Byte 4 is an unused dummy field, set to 0.

```
integer*2     pri_rec_num
byte          pri_rec_type
byte          dummy
```

This is followed by a one-dimension array of 512 bytes indexed 0 through 511. This represents the possible channel numbers of 0 through 511 (512 channels total).

```
byte          pri_chan_type(0:511)
```

The number that is inserted into each of the 512 bytes corresponds to the digitizer type assigned to the channel number. If the number is 0 (default), it is an unused, or dummy channel. If the number is other than 0, it represents the digitizer type and, consequently, the record type description that appears elsewhere in the ICF. The current list of possible digitizer types follows:

0 = Overhead or Dummy.
18 = SANDUS Trigger.
19 = SANDUS Analog.
20 = SANDUS Digital.
22 = SANDUS Analog Mux.
24 = Tektronix 7912.
28 = Tektronix 7103.
30 = RTD 720 Device.
31 = RTD 720 Channel.

The last field is allocated for the date/time of the last revision.

```
character*24  last_edit
```

There is always one, and only one, of this record type in every control file.

A.4.5 Record Type 30 /RTD_DEV_DESC/

This record describes the RTD 720 device. Because the device may be configured differently for the task of taking experiment data, measuring fiber optic laser calibration, or measuring characteristics of the cable plant, each device will have three records of this type in the ICF, one for each task. Byte 4 of the record is used to identify the configuration/task combination.

Within the record structure for RTD_DEV_DESC, bytes 1 and 2 combine to form the integer device number (dev_num), which is set to the decimal number that is the same as the GPIB bus address for the device. Record type (dev_rec_type, byte 3) is set to 30. Record use (rec_use, byte 4) identifies the task; and is set to 0 for Experiment, 1 for Laser Calibration, and 2 for Cable Compensation.

```
integer*2     dev_num
byte          dev_rec_type
byte          rec_use
```

The next two fields are allocated for the date/time of the last revision and identification of the person or program making the change.

```
character*24  last_edit
character*12  editor
```

The next four fields contain general information such as the identification of the rack housing the RTD 720 digitizer, data width, node_name, and port_id. Rack identification was not used during the Hunters Trophy event. When it is used, it is carried through to the PRELEWD program and appears on the data plot. Data width is the width of a data sample in bits and is always set to 8 for RTD 720s. A typical node name for Hunters Trophy was "RMV10". And the accompanying port_id was set to "IXA0:" when the ICF was associated with RMV 11, and "IXA1:" when the ICF was associated with RMV 12.

```
character*8   source_loc
integer*4     data_width
character*8   node_name
character*8   port_id
```

The remainder of this record structure is allocated to the various RTD 720 commands used to configure and control the digitizer device operation. To convey the function and meaning of each command within this document would require the contents of the Tektronix manual to be duplicated. Therefore, the reader is referred to the RTD 720 Operators Manual for further definition of the commands. What is included here is the field name followed by a list of valid entries where applicable.

One field is allocated to the "VERTICAL" command:

```
character*4   vmode       Vertical mode;
                          CH1, DUAL, or QUAD.
```

Six fields are associated with "ACQUIRE" commands:

character*8	acq_mode	Mode; NORMAL or ADVANCE.
character*8	acq_state	State; STOP, RUN, or HLDNXT.
character*8	acq_interval	Acquire interval; 500E-12, 1E-9, 2E-9 or 4E-9.
integer*4	acq_length	Record length; 512 to maximum memory for each channel.
integer*4	acq_nrecord	Number of consecutive records to fill in to fill in ADVANCE mode; 1 to a maximum of 1024.
character*8	acq_clock	Acquire clock source; INTRNAL or EXTERNAL.

Nine fields are associated with "ARM and TRIGGER" commands:

character*8	arm	Arming source; INTRNAL or EXTERNAL.
character*8	trig_mode	Trigger mode; AUTO or NORMAL.
character*8	trig_coupling	Trigger coupling; AC, DC, or HFREJ.
real*4	trig_level	Trigger level in percent of full scale or volts as determined by type.
character*8	trig_typelevel	Trigger level type; PERCENT or VOLTS.
real*4	trig_position	Trigger position in counts, time, or percent of record length as det- ermined by type.

character*8	trig_typepos	Trigger position type; PERCENT, POINT, or SECOND.
character*8	trig_slope	Trigger slope; PLUS or MINUS.
character*8	trig_source	Trigger source; CH1, CH2, CH3, CH4, or EXTERNAL.

Two fields are associated with "WAVEFORM PREAMBLE" commands:

character*8	wftx_mode	Waveform transfer format; DL at this time.
character*8	wftx_intrleave	Waveform transfer interleave; ON or OFF.

Four fields are associated with "DATA and WAVEFORM" commands:

integer*4	data_cnt_rec	Number of records in the waveform transfer.
integer*4	data_count	Number of points in the waveform transfer.
integer*4	data_start	Data transfer starting point.
integer*4	data_strecord	Selects first record to be transferred.

Three fields are associated with "HIGH-SPEED DATA OUTPUT" commands:

character*4	hsdo_state	High speed port selection; ON - Data is sent to HSDO port, OFF - Data is sent to GPIB port.
integer*4	hsdo_txmode	HSDO port's handshake mode; 1 or 2.
character*12	hsdo_dump	HSDO dump mode; CONTINUOUS or OFF.

Thirteen fields are associated with "STATUS and EVENT" commands:

character*4	rqs	Enable/Disable SRQ line; ON or OFF.
character*4	srq_abstouch	Enable/Disable SRQ on the following status or error conditions; ON (Enable) or OFF (Disable).
character*4	srq_cmderr	ON or OFF.
character*4	srq_exerr	ON or OFF.
character*4	srq_exwarn	ON or OFF.
character*4	srq_idprobe	ON or OFF.
character*4	srq_inerr	ON or OFF.
character*4	srq_inwarn	ON or OFF.
character*4	srq_opcmpl	ON or OFF.
character*4	srq_usr1	ON or OFF.
character*4	srq_usr2	ON or OFF.
character*12	uid	A string that assigns a name to the device.
character*80	ident	Space for response to the ID? query.

Ten fields are associated with "GPIB RELATED" commands:

character*8	abstouch	CLEAR or a string representing x,y coordinates of a simulated button push.
character*4	debug_gpib	Sets the state of GPIB debugging; ON or OFF.
character*8	dt	Sets the acquisition state; RUN, STOP, HLDNXT, or OFF.
character*8	init	Initiation source; PANEL, GPIB, or ALL.

character*4	longform	Longform command; ON or OFF.
character*4	path	Path command; ON or OFF.
character*12	user1(2)	Quoted strings to be displayed. (Two fields).
character*12	user2(2)	Quoted strings to be displayed. (Two fields).

Seven fields are associated with "CURSOR" commands:

character*4	crs1_loctn	WINx, where x = {1,2,3,4}.
character*4	crs2_loctn	WINx, where x = {1,2,3,4}.
integer*4	crs1_xpoint	0 to record_length-1.
integer*4	crs2_xpoint	0 to record_length-1.
character*8	cref	FIRST or SAME.
character*8	crsd_typetime	HZ or SECOND.
character*4	cursors	ON or OFF.

Four fields are allocated for preamble data from the device:

integer*4	pt_off	Number of points between the trigger and the first point transmitted.
integer*4	xzero	Horizontal zero point (always 0).
integer*4	yoff	Vertical binary offset (always 128).
real*4	xincr	Horizontal sample interval in seconds.

One field is allocated for waveform data from the device:

real*4	trfract	Fraction of sample interval in which the trigger occurred. Possible values depend upon vertical mode and are as follows; CH1: 0; DUAL: 0., .5; QUAD: 0., .25, .5, .75.

There will be three records of this type for each RTD device, one record with the "record use" byte set to 0 to configure the device for taking Experiment data, one record with the "record use" byte set to 1 to configure the device for taking Laser Calibration data, and one record with the "record use" byte set to 2 to configure the device for taking Cable Compensation data.

A.4.6 Record Type 31 /RTD_CHAN_DESC/

This record describes the RTD 720 channel. Because the channel may be configured differently for the task of taking experiment data, measuring fiber optic laser calibration, or measuring characteristics of the cable plant, each channel may have three records of this type in the ICF, one for each task. Byte 4 of the record is used to identify the configuration/task combination.

If an RTD is operating in "QUAD" mode with four channels (traces), then there may be up to twelve records of this type for the device.

Within the record structure for RTD_CHAN_DESC, bytes 1 and 2 combine to form the integer channel number (chan_num), which is the combination of the GPIB bus address and the RTD 720 channel number. Record type (chan_rec_type, byte 3) is set to 31. Record use (rec_use, byte 4) identifies the task; it is set to 0 for Experiment, 1 for Laser Calibration, and 2 for Cable Compensation.

integer*2	chan_num
byte	chan_rec_type
byte	rec_use

The next three fields are allocated for the channel number in ASCII, date/time of the last revision, and identification of the person or program making the change.

```
character*16  sec_desc
character*24  last_edit
character*12  editor
```

The next field specifies how many subchannels are associated with the channel.

integer*4	num_sub_chans	Number of subchannels; Equal to or greater than 1, equal to or less than 16.

Five fields are associated with "VERTICAL SETUP" commands:

character*8	ch_range	Full scale vertical range setting; 0.25 to 20.0 Volts.
character*8	ch_offset	Input offset in percent of full scale or volts as determined by offset type; 0 to 100 Percent or +/- 1.0 Volts to +/- 10.0 Volts.
character*8	ch_typeoffset	Offset type; PERCENT or VOLTS.
character*4	ch_coupling	Vertical coupling; AC, DC, or OFF.
character*8	ch_bwlim	Bandwidth limiting filter; FULL, HUNDRED, or TWENTY (MHz).

One field is associated with "DATA and WAVEFORM" commands:

character*4	data_ch	Waveform data from this channel; ON or OFF.

Two fields are allocated for preamble data from the channel:

real*4	ymult	Vertical scale factor (volts per count).
real*4	yzero	Vertical offset of the wave form (volts).

Nine fields are associated with setting up the Laser Calibrator:

character*12	sim_ts_node	Terminal server node name.
integer*4	las_cal_cntr	Switch Controller address.
integer*4	las_sig_cal	Gage signal / Laser cal address.
integer*4	las_freq	50 MHz / 1 MHz address.
integer*4	las_ext	Laser input; Internal / External address.
integer*4	las_lcl_rmt	Local / Remote address.
integer*4	las_lckout	Laser lockout address.
integer*4	las_trig	Laser closure trigger address.
integer*4	las_power	Laser Cal ON/OFF power address.

This completes the RTD_CHAN_DESC structure description.

A.4.7 Record Type 32 /RTD_EXP_SUB_DESC/

This record describes the RTD 720 subchannel configuration for Experiment data. Record types 33 and 34 are used to describe subchannel configuration for Laser Calibration and Cable Compensation respectively.

There will be one record of this type present for every RTD 720 subchannel. Each subchannel is associated with one unique event experiment. In addition, there will also be one record of this type designated subchannel 00, which represents the subchannel configuration for the entire channel, i.e., all subchannels.

If an RTD channel is being used to record three subchannels, the ICF will have a total of four records of this type. Each subchannel will have one record, plus the combination subchannel 00.

Within the record structure for RTD_CHAN_DESC, bytes 1 and 2 combine to form the integer channel number (chan_num), which is the combination of the GPIB bus address and the RTD 720 channel number. Record type (sub_rec_type, byte 3) is set to 32. Byte 4 identifies the integer subchannel number.

integer*2	chan_num
byte	sub_rec_type
byte	sub_num

Twenty-four fields are allocated for "GENERAL" information:

character*16	sec_desc	Channel number in ASCII.
character*24	last_edit	Date/time of last edit.
character*12	editor	Name of person or program making the change.
character*12	expmt_id	Sandia Experiment number.
character*20	expmtr_name	Experimenter name.
character*8	expmtr_org	Experimenter organization, or acronym if non-Sandia.
character*16	expmtr_note	Plot annotation string, defined by Experimenter.

character*8	plot_option	Plot option; Acceptable entries are: 'NONE', 'COUNTS', 'LINEAR'. 'NONE' requests no plot, 'COUNTS' requests a plot in percent of full scale. 'LINEAR' requests a plot of engineering units in the linear form "ay+b".
character*12	y_label	Y-axis label for linear plots.
real*4	conv_factor	Gage conversion factor in engineering units per volt.
integer*4	expmt_order	Experiment order of occurrence on the trace.
real*4	expmt_delay	Signal delay inserted to achieve time multiplexing of experiments.
real*4	expmt_atten	Signal attenuation which which results from the inserted signal delay.
integer*4	expmt_left	Experiment left bound in counts.
integer*4	expmt_right	Experiment right bound in counts.
logical*4	use_start_stop	Use the plot start and stop time which follow; .true. means use plot_start and plot_stop, .false. means the plot start and stop time will come from other sources.

real*8	plot_start	Plot start time in seconds; valid only if use_start_stop above is .true.
real*8	plot_stop	Plot stop time in seconds; valid only if use_start_stop above is .true.
logical*4	use_min_max	Use the y-min and y-max values which follow; .true. means use y_min and y_max, .false. means the plot y min and max value will come from other sources.
real*4	y_min	Minimum y value to plot; valid only if use_min_max above is .true., units must agree with y_label.
real*4	y_max	Maximum y value to plot; valid only if use_min_max above is .true., units must agree with y_label.
real*8	relative_delay	Relative delay in seconds, a correction added to time values before plotting.
character*4	zero_ref	Zero reference; used to identify on plots the basis for the relative_delay correction above. 'COMP'(ton), 'FIDU', ' ' are acceptable. Use of this field MAY

		classify the plot.
character*4	srad_file	Indicates whether or not to produce an SRAD file while generating a plot. Acceptable entries are 'NONE', and 'SRAD'.

Four fields are allocated for "K-FACTOR" information:

logical*4	apply_kfactor	Apply K-factor to this experiment; .true. means apply.
real*4	kfactor	Measured K-factor.
real*4	kfactor_atten	Late change affecting K-factor.
character*80	kfactor_method	Description of how the K-factor was measured.

Two fields are allocated for "EQUALIZATION" information.

logical*4	equalize	Apply equalization to this experiment; .true. means equalize.
character*48	equalize_file	Filename of the equalization function.

Two fields are allocated for "NORMALIZATION" information.

logical*4	normalize_time	Normalize the time axis; .true. means normalize.
real*4	normalz_factor	Normalization factor.

Four fields are allocated for "TREND ANALYSIS" information.

real*4	sig_atten	Signal attenuation for TREND.
real*4	sig_term_res	Signal termination resistance.

real*4	sig_prop_time	Total signal propagation time (seconds).
real*4	trg_prop_time	Total trigger propagation time (seconds).

Fifteen fields are allocated for "SIMULATION" information.

logical*4	sim_option	External simulation; .true. means simulation applied.
integer*4	num_simulators	Number of simulators used to generate the simulation signal, four maximum.
character*16	sim_unique_id(4)	Four fields in which to store unique identification for each simulator used.
real*8	sim_pp_volts	Simulated signal voltage, peak-to-peak being inserted for this experiment.
real*4	insert_atten	Attenuator inserted after fanout.
real*4	fanout_atten	Fanout attenuation.
real*4	cable_atten	Sum of all cable attenuation between the simulation signal generator and this experiment.
integer*4	sim_switch	Simulator switch number.
character*16	sim_unique_fo	Fiber Optic ID code.
integer*4	sim_fo_switch	Fiber Optic switch number.
character*8	est_tek_atten	Estimated attenuation needed to avoid over driving the Tek 11801.

integer*4	picosec_atten	Attenuation setting, Picosecond generator.

This completes the RTD_EXP_SUB_DESC structure description.

A.4.8 Record Type 33 /RTD_LC_SUB_DESC/

Currently, this structure is identical to the RTD_EXP_SUB_DESC structure and is defined in the same manner. However, in the future the defined fields and descriptions are subject to change in order to customize.

A.4.9 Record Type 34 /RTD_CC_SUB_DESC/

Currently, this structure is identical to the RTD_EXP_SUB_DESC structure and is defined in the same manner. However, in the future the defined fields and descriptions are subject to change in order to customize.

A.5 UTILITY ROUTINES

There are ten utility routines available for use with ICF files. Recently, [2] some features were added to several of the utilities [1] to enhance their usefulness.

Currently, programs that execute as part of the database application call the following utilities:

1)	OPEN_TBL	Opens an existing control file.
2)	CLOSE_TBL	Closes a control file.
3)	ADD_TBL_REC	Adds a record to the control file.
4)	READ_TBL_REC	Reads a record from the control file.

An additional stand alone utility CREATE_TBL is used to create a new, empty ICF. When a new control file is initially created it contains only one record, the STRUCT_DESC record, and must be populated with other records before being useful.

A.6 CONTROL FILE MOVEMENT

The life cycle of all Instrument Control Files begins and normally ends within the database application. Files are created by one of two methods:

(1) Upon command issued by the database operator, which may be invoked at any time.

(2) Creation occurs when an ICF that has been modified by digitizer operations is received by the database application.

Except for the first-time ICF creation, database operators enter only "administrative" information into the database. This information is associated with items such as experimenter name, organization, measurement descriptions, etc. On the other hand, "technical" information is associated with the setup of the digitizer. Technical information is overlaid into the database by reading an ICF that has been modified.

Each time the database application creates a new ICF, a copy is placed into the directory N12DBM::ID:[INGRES.INST_TABLES]. This directory is then purged to keep the directory clean with only the latest version present.

Files are copied from N12DBM to the appropriate directory on computer node GEAR10 under supervision of the CONTROL program. Each time a RTD system operator logs onto the GEAR10 node in interactive mode, the CONTROL program searches the GEAR10 directory DD:[TABLES] for ICF. For each file encountered, the node N12DBM is searched for a newer ICF with the same name. If a newer file is available from N12DBM, CONTROL initiates a dialogue in which the operator is told which new files are available and given the option to copy each new ICF. If a copy is requested, the new file is pulled from N12DBM. Otherwise, no updated copy of the ICF will be pulled.

In addition to the dialogue at login, the CONTROL menu on GEAR10 is designed to allow the pulling of a new ICF at any time during the CONTROL session, should it become necessary.

The RTD digitizer control and setup software (called RTDTEST) [3] is used to modify an ICF. After the ICF has been modified one or more times, a global symbol is defined to inform the CONTROL program that ICF modifications have been made. When the operator selects either the "EXIT" or "Push modified ICF to database node" command from the CONTROL menu, CONTROL pushes a copy of the modified ICF to directory N12DBM::ID:

[INGRES.CTRL_TABLES], where it is subsequently used to overlay the database.

The database application periodically searches the directory N12DBM::ID:[INGRES.CTRL_TABLES] for receipt of modified ICF. When a file is found, the application:

(1) Copies the ICF into an internal working directory.

(2) Examines the "last_edit" field from each RTD device, channel, and subchannel record to determine whether the date/time that appears in the ICF is different from the date/time currently residing in the database.

(3) For each record with differing date/time information, selected "technical" information (i.e. instrument setup) is extracted from the ICF and used to overlay existing database contents.

(4) The input ICF is then deleted from the directory N12DBM::ID:[INGRES.CTRL_TABLES] ending the life cycle for the file.

(5) Creates a new ICF that has the most current database information, and places the ICF into the directory N12DBM::ID:[INGRES.INST_TABLE].

This procedure assures that the most up-to-date version of any given RTD ICF is always resident on computer node N12DBM. The database application never modifies existing control files. After serving their intended purpose, files are deleted and replaced with totally new files.

A copy of the complete, unabridged version of TABLE_STRUCTS.DEF as defined and used for the Hunter's Trophy event follows.

```
!   TABLE_STRUCTS.DEF     May 12, 1992          R. Aden, 9321

!   Hunters Trophy

!   This file defines the fields for the new control files.  It assumes the
!   following organization:  Each data source (ACE, SANDUS, RMV, etc.)
!   will have one control file associated with it.  The control file will
!   contain information common to all channels of the source as well as
!   channel-specific information.  The common information describes the
!   configurations of the source and collection hardware.  The contents
!   and form of the channel-specific information will vary, depending on
!   the type of channel.
!
!   Unless otherwise noted all values are communicated using the following
!   set of base units:
!
!               Time        is in seconds.
!               Interval    is in seconds.
!               Delay       is in seconds.
!               Resistance  is in ohms.
!               Impedance   is in ohms.
!               Voltage     is in volts.
!               Current     is in amperes.
!               Frequency   is in hertz.
!
! ***************************************************************************
!
! Some notes about the form of this file:

!       (1)     The first four bytes must look like:
!                       integer*2
!                       byte
!                       byte

!       (2)     The file MUST be named TABLE_STRUCTS.DEF and reside in
!               LD:[INCLUDE] else CREATE_TBL will not find it.

!       (3)     The first byte position is the number of the structure
!               in this file.  The structure /STRUCT_DESC/ must remain first
!               and the /UNUSED_nn_DESC/ must be present or subsequent
!               structures must be renumbered.
!
!       (4)     There are some restrictions on the structures this routine
!               can analyze.
!
!               (1) Field names must be declared individually on separate lines.
!                   For example, the routine does not understand this structure:
!
!                       structure /struct/
!                           integer*2 a, b
!                       end structure
!
!                  It must be defined this way:
!
!                       structure /struct/
!                           integer*2 a
!                           integer*2 b
!                       end structure
!
```

```
!               (2) Substructures are allowed, but you may not declare records
!                   that are defined by substructures.
!
!               (3) Only one- and two-dimensional arrays are allowed.


!   The following structure is used to keep an internal description of the
!   table within the table itself.  It will normally only be used by the
!   table access routines.

        structure /STRUCT_DESC/
            integer*2    struct_rec_num          ! Structure record number (always 0).
            byte         struct_rec_type         ! Structure record type (1).
            byte         dummy                   ! Dummy field (not used).

            character*24 rec_nam(40)             ! Record structure name.
            integer*4    rec_len(40)             ! Record length in bytes.
            integer*4    tot_structs             ! Number of record structures.
            character*24 last_edit               ! Date of last edit.
        end structure


!   The following structure describes general information about the table, test,
!   and characteristics of the source.

        structure /GEN_DESC/
            integer*2    gen_rec_num             ! General record number (always 0).
            byte         gen_rec_type            ! General record type (2).
            byte         dummy                   ! Dummy field (not used).

            character*24 test_name               ! DNA Test name.
            character*8  test_number             ! Department 9321 test number for
                                                 !   software version tracking.
            character*16 source_name             ! Source name.
            character*4  source_code             ! Source code.
            integer*4    num_streams             ! Number of streams from this source.
            integer*4    ref_trig                ! Reference trigger number.
            integer*4    ref_trig_tol            ! Maximum difference counts.
            integer*4    hddr_offset             ! Offset for HDDR stream number.
            integer*4    mm_offset               ! Offset for MM stream number.
            real*4       sec_per_bit             ! Bit stream rate.
            character*24 last_edit               ! Date of last edit.
        end structure


!   The following structure describes the streams coming out of the source.
!   There is one record per stream.

        structure /STREAM_DESC/
            integer*2    strm_rec_num            ! Stream record number.
            byte         strm_rec_type           ! Stream record type (3).
            byte         dummy                   ! Dummy field (not used).

            character*12 prime_second            ! Indicates whether this is the
                                                 !   primary or secondary stream.
                                                 !   Acceptable entries are:
                                                 !   'PRIMARY', 'SECONDARY'.
```

```
        character*4  dest_type              ! Stream destination type;
                                            !   acceptable entries are:
                                            !   'UNKN', 'MM', 'HDDR'.
        character*8  dest_node              ! Destination computer node.
        integer*4    dest_rec_num           ! Destination record number.
        integer*4    frame_length           ! Frame length in bytes.
        integer*4    subcom_depth           ! Frame subcommutation depth.
        integer*2    format(200,8)          ! Format description.
        character*24 last_edit              ! Date of last edit.
    end structure

!   The following structure describes the mass memory.  There should be a record
!   for each stream into mass memory.

    structure /MM_DESC/
        integer*2    mm_rec_num             ! Mass memory record number.
        byte         mm_rec_type            ! Mass memory record type (4).
        byte         dummy                  ! Dummy field (not used).

        character*16 col_name               ! Name of collection point.
        integer*4    pointer_addr           ! Address pointer into zero-time data buffer;
                                            !   acceptable entries are 0 or 1.
        integer*4    cal_start_addr         ! Starting address of calibration buffer;
                                            !   entry should always be 2.
        integer*4    cal_end_addr           ! Ending address of calibration buffer.
        integer*4    zt_start_addr          ! Starting address of zero-time data buffer.
        integer*4    zt_end_addr            ! Ending address of zero-time data buffer.
        integer*4    mm_presamples          ! Number of presamples.
        character*8  mm_dev_name            ! Computer device name.
        character*24 last_edit              ! Date of last edit.
    end structure

!   The following structure describes the HDDR.  There should be a record for
!   each stream into an HDDR.

    structure /HDDR_DESC/
        integer*2    hddr_rec_num           ! HDDR record number.
        byte         hddr_rec_type          ! HDDR record type (5).
        byte         dummy                  ! Dummy field (not used).

        character*16 col_name               ! Name of collection point.
        integer*4    hddr_track             ! HDDR track number.
        real*4       record_speed           ! Recording speed in inches per second.
        integer*4    e8330_live_inp         ! 8330 live input port number.
        integer*4    e8330_tape_inp         ! 8330 tape input port number.
        integer*4    ta667_fmt_num          ! TA667 format number.
        integer*4    hardware_path          ! Hardware path taken by live stream;
                                            !   acceptable entries are 1 or 2.
        character*8  ta667_dev(2)           ! Computer device name for TA667.
        character*8  ta653_dev              ! Computer device name for TA653.
        character*8  ta717_dev(2)           ! Computer device name for TA717.
        character*8  e8330_dev(2)           ! Computer device name for 8330.
        integer*4    e8330_sta_no(2)        ! 8330 station number.
        character*24 last_edit              ! Date of last edit.
    end structure
```

```
!   The following structure describes the GEAR collection system.  There should
!   be one record for each GEAR system.  (GPIB ethernet Analog Recording)

        structure /GEAR_DESC/
            integer*2    struct_rec_num          ! Structure record number.
            byte         struct_rec_type         ! Structure record type (6).
            byte         dummy                   ! Dummy field (not used).

            character*16 col_name                ! Collection point node name.
            character*16 ether_port(2)           ! Ethernet port name(s).
            integer*4    %fill(4)                ! For future use.
            character*24 last_edit               ! Date of last edit.
        end structure


!   The following structure describes the data source.  There should always be
!   one of these records present in the table.

        structure /SOURCE_DESC/
            integer*2    source_rec_num          ! Source record number (always 0).
            byte         source_rec_type         ! Source record type (7).
            byte         dummy                   ! Dummy field (not used).

            character*16 col_name                ! Name of collection point.
            character*8  ta666_dev               ! DRQ port connected to TA666.
            character*8  terminal_dev            ! Terminal port connected to ACE.
            character*8  server_dev              ! Terminal server port connected to ACE.
            character*24 last_edit               ! Date of last edit.
        end structure


!   The following structure is presently unused, it is here for future expansion,
!   and it must be present as a placeholder.

        structure /UNUSED_08_DESC/
            integer*2    struct_rec_num          ! Structure record number.
            byte         struct_rec_type         ! Structure record type (8).
            byte         dummy                   ! Dummy field (not used).
        end structure


!   The following structure is presently unused, it is here for future expansion,
!   and it must be present as a placeholder.

        structure /UNUSED_09_DESC/
            integer*2    struct_rec_num          ! Structure record number.
            byte         struct_rec_type         ! Structure record type (9).
            byte         dummy                   ! Dummy field (not used).
        end structure


!   The following structure is presently unused, it is here for future expansion,
!   and it must be present as a placeholder.

        structure /UNUSED_10_DESC/
            integer*2    struct_rec_num          ! Structure record number.
            byte         struct_rec_type         ! Structure record type (10).
            byte         dummy                   ! Dummy field (not used).
        end structure
```

```
!   The following structure is presently unused, it is here for future expansion,
!   and it must be present as a placeholder.

        structure /UNUSED_11_DESC/
            integer*2     struct_rec_num          ! Structure record number.
            byte          struct_rec_type         ! Structure record type (11).
            byte          dummy                   ! Dummy field (not used).
        end structure


!   The following structure describes the family of simulation sources.  There
!   will be one record of this type for every simulator.

        structure /SIMULATOR_DESC/
            integer*2     struct_rec_num          ! Structure record number.
            byte          struct_rec_type         ! Structure record type (12).
            byte          dummy                   ! Dummy field (not used).

            character*16 sim_unique_id            ! Unique identification.
            character*16 sim_model                ! Model description.
            character*12 sim_arm_mode             ! Arming/Control mode.
            character*12 sim_location             ! Simulator location.
            character*12 sim_vax_node             ! VAX Port/Node name.
            character*12 sim_ts_node              ! Terminal server node name.
            character*12 sim_trg_source           ! Trigger source.
            character*12 sim_trg_mode             ! Trigger mode.
            character*12 sim_trg_slope            ! Trigger slope.
            real*4       sim_trg_level            ! Trigger level.
            real*4       sim_trg_delay            ! Trigger delay.
            real*4       sim_thruput              ! Simulator thru-put delay.
            logical*4    sim_term_50              ! Simulator termination;
                                                  !   .true.  indicates 50 ohm termination,
                                                  !   .false. indicates high impedance.
            real*4       sim_baseline             ! Baseline voltage.
            real*4       sim_peak_v               ! Peak voltage.
            real*4       sim_pul_width            ! Pulse width.
            real*4       sim_pul_rise             ! Pulse risetime.
            real*4       sim_pul_fall             ! Pulse falltime.
            real*4       sim_pul_area             ! Pulse area.
            real*4       sim_pul_delay            ! Pulse delay.
            real*4       sim_add_atten            ! Added attenuation.
            real*4       sim_pul_period           ! Pulse period.
            character*12 sim_polarity             ! Pulse polarity.
            real*4       sim_max_volts            ! Maximum programmable voltage.
            real*4       sim_max_width            ! Maximum programmable pulse width.
            real*4       sim_max_delay            ! Maximum programmable pulse delay.
            real*4       sim_max_offset           ! Maximum programmable offset.
            real*4       sim_max_period           ! Maximum programmable period.
            integer*4    sim_x_menu               ! Set-up recall number.
            integer*4    sim_gpib_addr            ! Unique GPIB address.
            integer*4    sim_pwr_switch           ! Power switch number.
            integer*4     num_channels            ! Number of channels which follow.
            character*12 sim_chan(16)             ! Channel array (16 max).
            character*12 sim_tgen_model           ! Trigger Generator model.
            character*12 sim_tgen_loc             ! Trigger Generator location.
            character*12 sim_tgen_node            ! Trigger Generator terminal server node.
            integer*4     sim_tgen_gpib           ! Trigger Generator GPIB address.
            integer*4     sim_tgen_p_sw           ! Trigger Generator power switch number.
            real*4       sim_tgen_volts           ! Trigger Generator output volts.
```

```
        real*4        sim_tgen_width        ! Trigger Generator pulse width.
        real*4        sim_tgen_delay        ! Trigger Generator delay.
        real*4        sim_tgen_rate         ! Trigger Generator pulse period.
        character*24  last_edit             ! Last edit.
    end structure


!   The following structure describes time mapping of SANDUS triggers.  There
!   should be one record of this type for each SANDUS.

    structure /S_TRIG_DESC/
        integer*2     trig_rec_num          ! Trigger record number (always 0).
        byte          trig_rec_type         ! Trigger record type (13).
        byte          dummy                 ! Dummy field (not used).

        integer*4     trig_num(6)           ! Trigger number.
        real*8        trig_time(6)          ! Trigger time.
        real*4        trig_pct_dev(6)       ! Allowable percent deviation.
        integer*4     san_counter(6)        ! SANDUS trigger counter number.
        character*24  last_edit             ! Date of last edit.
    end structure


!   The following structure is presently unused, it is here for future expansion,
!   and it must be present as a placeholder.

    structure /UNUSED_14_DESC/
        integer*2     struct_rec_num        ! Structure record number.
        byte          struct_rec_type       ! Structure record type (14).
        byte          dummy                 ! Dummy field (not used).
    end structure


!   The following structure is presently unused, it is here for future expansion,
!   and it must be present as a placeholder.

    structure /UNUSED_15_DESC/
        integer*2     struct_rec_num        ! Structure record number.
        byte          struct_rec_type       ! Structure record type (15).
        byte          dummy                 ! Dummy field (not used).
    end structure


!   The following structure is presently unused, it is here for future expansion,
!   and it must be present as a placeholder.

    structure /UNUSED_16_DESC/
        integer*2     struct_rec_num        ! Structure record number.
        byte          struct_rec_type       ! Structure record type (16).
        byte          dummy                 ! Dummy field (not used).
    end structure


!   The following structure describes the primary types of all the channels in
!   the source.  We assume a maximum of 512 channels per source.

    structure /PRI_TYPE_DESC/
        integer*2     pri_rec_num           ! Primary record number (always 0).
        byte          pri_rec_type          ! Primary record type (17).
```

```
        byte        dummy                   ! Dummy field (not used).

        byte        pri_chan_type(0:511)    ! Primary channel type description:
                                            !   0  = Overhead or Dummy.
                                            !   18 = Trigger.
                                            !   19 = SANDUS Analog.
                                            !   20 = SANDUS Digital.
                                            !   21 = SANDUS Digital Subchannel.
                                            !   22 = SANDUS Analog Mux.
                                            !   23 = SANDUS AMux Subchannel.
                                            !   24 = Tektronix 7912.
                                            !   25 = Tektronix 7912 Subchannel.
                                            !   26 = Currently unused.
                                            !   27 = Currently unused.
                                            !   28 = Tektronix 7103.
                                            !   29 = Tektronix 7103 Subchannel.
                                            !   30 = RTD 720 Device.
                                            !   31 = RTD 720 Channel.
                                            !   32 = RTD 720 Experiment Subchannel.
                                            !   33 = RTD 720 Laser Cal Subchannel.
                                            !   34 = RTD 720 Cable Comp Subchannel.
        character*24 last_edit              ! Date of last edit.
    end structure

!   The following structure describes trigger counter channels.  There should be
!   a record of this type present for every trigger counter channel.

    structure /TRIGGER_CHAN_DESC/
        integer*2   chan_num                ! Channel number (octal).
        byte        chan_rec_type           ! Channel record type (18).
        byte        dummy                   ! Dummy field (not used).

        integer*4   data_width              ! Width of data sample in bits.
        integer*4   byte_span               ! Number of bytes per sample.
        integer*4   trig_num                ! Trigger number for this channel (1-6).
        integer*4   trig_byte               ! Byte of trigger count this channel occupies (1-4).
        character*24 last_edit              ! Date of last edit.
    end structure

!   The following structure describes the SANDUS analog channels.  There should
!   be one record of this type present for every SANDUS analog channel.

    structure /S_ALOG_CHAN_DESC/
        integer*2   chan_num                ! Channel number (octal).
        byte        chan_rec_type           ! Channel record type (19).
        byte        dummy                   ! Dummy field (not used).

                                            ! "GENERAL" Information follows:

        character*16 sec_desc               ! Channel number in ASCII.
        character*24 last_edit              ! Date of last edit.
        character*12 expmt_id               ! SNL Experiment number.
        character*20 expmtr_name            ! Experimenter name.
        character*8  expmtr_org             ! Experimenter organization, or
                                            !   acronym if non-Sandia.
        character*16 expmtr_note            ! Experimenter plot annotation.
```

```
character*8  plot_option      ! Plot option;
                              !   a character string such as
                              !   'NONE', 'COUNTS', 'LINEAR', 'YT-1D',
                              !   'YT-HD', 'TC-K', 'SLIFER', etc.
                              !   'NONE' requests no plot, 'COUNTS'
                              !   requests a plot in percent of full scale.
                              !   'LINEAR' requests a plot of engineering
                              !   units in the linear form "ay+b".
                              !   Other values request a plot of engineering
                              !   units applying a non-linear, special routine.
character*12 y_label          ! Y-axis label for linear plots.
character*12 alt_y_label      ! Y-axis label for non-linear plots.
integer*4    data_width       ! Width of the data sample in bits.
                              !   Used to determine the UBE,
                              !   the maximum count value.
                              !   Appears in the plot legend.
integer*4    byte_span        ! Number of bytes per sample.
character*8  data_conv        ! Data conversion;
                              !   acceptable entries are:
                              !   'LINEAR', 'EXTENDED'.
                              !   PRELEWD interprets 'EXTENDED' to mean
                              !   data_width = 12 and byte_span = 2.
logical*4    use_start_stop   ! Use the plot start and stop time which follow;
                              !   .true.  means use plot_start and plot_stop,
                              !   .false. means the plot start and stop time
                              !   will come from other sources.
real*8       plot_start       ! Plot start time in seconds;
                              !   valid only if use_start_stop above is .true.
real*8       plot_stop        ! Plot stop time in seconds;
                              !   valid only if use_start_stop above is .true.
logical*4    use_min_max      ! Use the y-min and y-max values which follow;
                              !   .true.  means use y_min and y_max,
                              !   .false. means the plot y min and max value
                              !   will come from other sources.
real*4       y_min            ! Minimum y value to plot;
                              !   valid only if use_min_max above is .true.,
                              !   units must agree with y_label or alt_y_label.
real*4       y_max            ! Maximum y value to plot;
                              !   valid only if use_min_max above is .true.,
                              !   units must agree with y_label or alt_y_label.
real*8       start_time       ! Sample interval start time.
real*8       stop_time        ! Sample interval stop time.
real*8       relative_delay   ! Relative delay in seconds, a correction
                              !   added to time values before plotting.
character*4  zero_ref         ! Zero reference; used to identify on plots the
                              !   basis for the relative_delay correction above.
                              !   Currently 'COMP'(ton), 'FIDU', and ' ' are
                              !   acceptable.  Use of this field MAY classify
                              !   the resulting plot.
logical*4    compress         ! Compress data;
                              !   .true. indicates that hardware compression
                              !   of the data has occurred, while
                              !   .false. indicates NO hardware
                              !   compression has occurred.
integer*4    nom_devtn        ! Acceptable deviation from nominal;
                              !   valid only if compress above
                              !   is .true.  This is a decimal value.
```

```
integer*4    delay_count      ! Number of samples before compression;
                              !   valid only if compress above
                              !   is .true.
logical*4    log_chan         ! Log this channel;
                              !   .true.  means that this is a data logger
                              !   channel and the log_rate which follows is
                              !   to be used for sample interval.
real*8       log_samp_int     ! Log sample rate in seconds;
                              !   valid only if log_chan above is .true.
character*4  srad_file        ! Indicates whether or not to produce
                              !   an SRAD file when generating a plot.
                              !   Acceptable entries are 'NONE', and 'SRAD'.

                              ! "SETUP" information follows:

character*8  circuit_type     ! Circuit type description;
                              !   acceptable entries are:
                              !   'BRIDGE' or 'V-MONITOR'.
character*8  data_mod_type    ! Module identification;
                              !   acceptable entries are:
                              !   'TA591', 'TA592'.
                              !   Appears in the plot legend.
integer*4    num_ad_conv      ! Number of A/D converters;
                              !   acceptable entries are 1 or 2.
character*8  mem_realtime     ! Module type;
                              !   acceptable entries are:
                              !   'MEMORY', 'REALTIME'.
character*8  mem_size         ! Memory size;
                              !   acceptable entries are:
                              !   '4 KB', '8 KB', '16 KB', '1 KB', 'N/A'.
integer*4    sig_cond_gain    ! Signal conditioning gain;
                              !   acceptable entries are:
                              !   1,   2,   5,   10,   20,   50,
                              !   100, 200, 500, 1000, 2000, 5000, 10000.
                              !   Appears in the plot legend.
character*8  input_offset     ! Input offset;
                              !   acceptable entries are:
                              !   '-FS', '-3/4', '-1/2', '-1/4',
                              !   '0',   '+1/4', '+1/2', '+3/4', 'N/A'.
real*4       filter_freq      ! Filter frequency, in Hz;
                              !   acceptable entries are:
                              !   500, 1000, 2000,  5000,
                              !   10000,   20000,   50000,
                              !   100000,  200000,  500000,
                              !   1000000, 2000000, 5000000, 10000000.
                              !   Appears in the plot legend.
character*4  pretrig_bytes    ! Number of pretrigger bytes.
                              !   To convert to pretrigger data samples
                              !   divide pretrig_bytes by byte_span.
integer*4    num_samp_ints    ! Number of sample intervals;
                              !   must be greater than 0 and
                              !   less than or equal to 2.
integer*4    num_samps(2)     ! Number of samples per sample interval.
real*8       samp_int(2)      ! Sample intervals, in seconds.
                              !   The first value appears in the
                              !   plot legend.
```

```
character*8  trig_mode          ! Trigger mode;
                                !   acceptable entries are:
                                !   'INTERNAL', 'EXT A', 'EXT B',
                                !   'MANUAL',   'N/A'.
character*8  trig_level         ! Internal trigger level;
                                !   acceptable entries are:
                                !   '-FS', '-3/4', '-1/2', '-1/4',
                                !   '0',   '+1/4', '+1/2', '+3/4', 'N/A'.
logical*4    trig_arm           ! Trigger arm;
                                !   .true.  means the module is armed,
                                !   .false. means the module is triggered.
logical*4    trig_inhibit       ! Trigger inhibit;
                                !   .true.  means trigger inhibited,
                                !   .false. means trigger not inhibited.
logical*4    trig_manual        ! Manual trigger;
                                !   .true.  means set trigger,
                                !   .false. means clear trigger.
integer*4    trig_number        ! Trigger number for this channel.
                                !   This trigger and the ref_trig are used
                                !   by PROCESS to determine the time of the
                                !   first sample.

                                ! "K-FACTOR,
                                !   NORMALIZATION, and TREND" information follows:

real*4       gage_req_exc       ! Gauge requested excitation.
logical*4    apply_kfactor      ! Apply K-factor to this channel;
                                !   .true. means apply.
real*4       kfactor            ! Measured K-factor.
real*4       kfactor_atten      ! Late change affecting K-factor.
character*80 kfactor_method     ! Description of how K-factor was
                                !   measured.
logical*4    normalize_time     ! Normalize the time axis;
                                !   .true. means normalize.
real*4       normalz_factor     ! Normalization factor.
real*4       sig_atten          ! Signal attenuation for TREND.
real*4       sig_term_res       ! Signal termination resistance.
real*4       sig_prop_time      ! Total signal propogation time.
real*4       trg_prop_time      ! Total trigger propogation time.

                                ! "CALIBRATION" information follows:

logical*4    cal_option         ! Calibrate this channel;
                                !   .true.  means calibrate.
character*12 cal_mode           ! Calibration mode (on TA591/TA592 board);
                                !   acceptable entries are:
                                !   'DATA',  'ZERO',    'SHUNT',
                                !   '+TEST', '-TEST',   'SYS CAL',
                                !   'SPARE', 'LIN CHK', 'N/A'.
integer*4    cal_size           ! Number of calibration samples.
logical*4    cal_use(4)         ! Calibration levels to use;
                                !   .true.  means use this level.
real*8       cal_level(4)       ! Calibrator levels in volts.
real*4       cal_eng_units(4)   ! Calibration levels in engineering units.
real*4       cal_slope          ! Calibration slope.
real*4       cal_slope_dev      ! Calibration slope deviation.
integer*4    cal_top            ! Top cal level to use.
integer*4    cal_bottom         ! Bottom cal level to use.
```

```
        integer*4     cal_diff_check        ! Minimum acceptable difference
                                            !   between cal top and cal bottom
                                            !   in counts.
        integer*4     cal_balance           ! If circuit_type = "BRIDGE", this
                                            !   field indicates which cal level corresponds
                                            !   to the gage balance level.
        character*8   cal_source(4)         ! Calibration source;
                                            !   acceptable entries are:
                                            !   'VS', 'ALT', 'N/A'.
        character*8   cal_type(4)           ! Calibration type;
                                            !   acceptable entries are:
                                            !   'ZERO',  'SHUNT', 'STEP',  '+TEST',
                                            !   '-TEST', 'GAGE',  'S-K1',  'S-K2',
                                            !   'SK1&3', 'S-K3',  'SK1&2', 'SK2&3',
                                            !   'SK1-3', 'EXT',   'INT',   'N/A'.
        character*8   cal_atten             ! Calibration attenuation factor;
                                            !   acceptable entries are:
                                            !   '1:1', '10:1', '100:1', '1K:1', 'N/A'.

                                            ! "SIMULATION" information follows:

        logical*4     sim_option            ! External simulation;
                                            !   .true. means simulation applied.
        integer*4     num_simulators        ! Number of simulators used to
                                            !   generate the simulation signal,
                                            !   four maximum.
        character*16 sim_unique_id(4)       ! Simulator unique identification
                                            !   for each simulator used.
        real*8        sim_pp_volts          ! Simulated signal voltage, peak-to-peak
                                            !   being inserted for this experiment.
        real*4        insert_atten          ! Attenuator inserted after fanout.
        real*4        fanout_atten          ! Fanout attenuation.
        real*4        cable_atten           ! Sum of all cable attenuation between
                                            !   the simulation signal generator and
                                            !   this experiment.
        integer*4     sim_switch            ! Simulator switch number.
        character*16 sim_unique_fo          ! Fiber Optic ID code.
        integer*4     sim_fo_switch         ! Fiber Optic switch number.
        character*8   est_tek_atten         ! Estimated attenuation needed to avoid
                                            !   over driving the Tek 11801.
        integer*4     picosec_atten         ! Attenuation setting, Picosecond generator.
    end structure

!   The following structure describes the SANDUS digital channels.  There should
!   be one record of this type present for every SANDUS digital channel.

    structure /S_DIG_CHAN_DESC/
        integer*2     chan_num              ! Channel number (octal).
        byte          chan_rec_type         ! Channel record type (20).
        byte          dummy                 ! Dummy field (not used).

                                            ! "GENERAL" information follows:

        character*16 sec_desc               ! Channel number in ASCII.
        character*24 last_edit              ! Date of last edit.
        integer*4     data_width            ! Width of data sample in bits.
        integer*4     byte_span             ! Number of bytes per sample.
```

```
character*8  data_conv          ! Data conversion:
                                !   acceptable entries are:
                                !   'LINEAR'.
real*8       start_time         ! Sample interval start time.
real*8       stop_time          ! Sample interval stop time.
integer*4    num_sub_chans      ! Number of subchannels (bit fields);
                                !   less than or equal to 12.
logical*4    sep_dig_subch      ! Process/Plot usage flag;
                                !   .true. use bit fields as described
                                !   in the subchannel structure(s),
                                !   .false. combine all bits (old method).
real*8       relative_delay     ! Relative delay in seconds, a correction
                                !   added to time values before plotting.
character*4  zero_ref           ! Zero reference; used to identify on plots the
                                !   basis for the relative_delay correction above.
                                !   Currently 'COMP'(ton), 'FIDU', and ' ' are
                                !   acceptable.  Use of this field MAY classify
                                !   the resulting plot.
logical*4    compress           ! Compress data;
                                !   .true. indicates that hardware compression
                                !   of the data has occurred, while
                                !   .false. indicates NO hardware
                                !   compression has occurred.
integer*4    compress_mask      ! Compression mask;
                                !   valid only if compress above
                                !   is .true.
integer*4    delay_count        ! Number of samples before compression;
                                !   valid only if compress above
                                !   is .true.
logical*4    log_chan           ! Log this channel;
                                !   .true.  means that this is a data logger
                                !   channel and the log_samp_int which follows is
                                !   to be used for sample interval.
real*8       log_samp_int       ! Log sample rate in seconds;
                                !   valid only if log_chan above is .true.
character*4  srad_file          ! Indicates whether or not to produce
                                !   an SRAD file when generating a plot.
                                !   Acceptable entries are 'NONE', and 'SRAD'.

                                ! "SETUP" information follows:

character*8  data_mod_type      ! Module identification;
                                !   acceptable entries are:
                                !   'TA591', 'TA592'.
                                !   Appears in the plot legend.
integer*4    num_ad_conv        ! Number of A/D converters;
                                !   acceptable entries are 1 or 2.
character*8  mem_realtime       ! Module type;
                                !   acceptable entries are:
                                !   'MEMORY', 'REALTIME'.
character*8  mem_size           ! Memory size;
                                !   acceptable entries are:
                                !   '4 KB', '8 KB', '16 KB', '1 KB', 'N/A'.
character*4  pretrig_bytes      ! Number of pretrigger bytes.
                                !   To convert to pretrigger data samples
                                !   divide pretrig_bytes by byte_span.
integer*4    num_samp_ints      ! Number of sample intervals;
                                !   must be greater than 0 and
                                !   less than or equal to 2.
```

```
        integer*4    num_samps(2)          ! Number of samples per sample interval.
        real*8       samp_int(2)           ! Sample intervals, in seconds.
                                           !   The first value appears in the
                                           !   plot legend.
        character*8  trig_mode             ! Trigger mode;
                                           !   acceptable entries are:
                                           !   'INTERNAL', 'EXT A', 'EXT B',
                                           !   'MANUAL',   'N/A'.
        logical*4    trig_arm              ! Trigger arm;
                                           !   .true.  means the module is armed,
                                           !   .false. means the module is triggered.
        logical*4    trig_inhibit          ! Trigger inhibit;
                                           !   .true.  means trigger inhibited,
                                           !   .false. means trigger not inhibited.
        logical*4    trig_manual           ! Manual trigger;
                                           !   .true.  means set trigger,
                                           !   .false. means clear trigger.
        integer*4    trig_number           ! Trigger number for this channel.
                                           !   This trigger and the ref_trig are used
                                           !   by PROCESS to determine the time of the
                                           !   first sample.

                                           ! "CALIBRATION" information follows:
                                           !   Calibration of SANDUS digital channels
                                           !   is not currently implemented in PRELEWD.

        logical*4    cal_option            ! Calibrate this channel;
                                           !   .true.  means calibrate.
    end structure

!   The following structure describes the SANDUS digital subchannels (bit fields).
!   There should be a record of this type present for every subchannel in a
!   SANDUS digital channel (12 maximum).

    structure /S_DIG_SUB_DESC/
        integer*2    chan_num              ! Main channel number.
        byte         sub_rec_type          ! Bit field record type (21).
        byte         sub_num               ! Subchannel (bit field) number (decimal).

                                           ! "GENERAL" information follows:

        character*16 sec_desc              ! Channel number in ASCII.
        character*24 last_edit             ! Date of last edit.
        character*12 expmt_id              ! SNL Experiment number.
        character*20 expmtr_name           ! Experimenter name.
        character*8  expmtr_org            ! Experimenter organization, or
                                           !   acronym if non-Sandia.
        character*16 expmtr_note           ! Experimenter plot annotation.
        character*8  plot_option           ! Plot option;
                                           !   acceptable entries are:
                                           !   'NONE', 'COUNTS'.
                                           !   'NONE' requests no plot, 'COUNTS'
                                           !   requests a plot in percent of full scale.
        character*12 y_label               ! Y-axis label for linear plots.
                                           !   Ignored for this channel type.
```

```
        logical*4   use_start_stop      ! Use the plot start and stop time which follow;
                                        !   .true.  means use plot_start and plot_stop,
                                        !   .false. means the plot start and stop time
                                        !   will come from other sources.
        real*8      plot_start          ! Plot start time in seconds;
                                        !   valid only if use_start_stop above is .true.
        real*8      plot_stop           ! Plot stop time in seconds;
                                        !   valid only if use_start_stop above is .true.
        logical*4   use_min_max         ! Use the y-min and y-max values which follow;
                                        !   .true.  means use y_min and y_max,
                                        !   .false. means the plot y min and max value
                                        !   will come from other sources.
        real*4      y_min               ! Minimum y value to plot;
                                        !   valid only if use_min_max above is .true.,
                                        !   units must agree with y_label or alt_y_label.
        real*4      y_max               ! Maximum y value to plot;
                                        !   valid only if use_min_max above is .true.,
                                        !   units must agree with y_label or alt_y_label.
        integer*4   field_offset        ! Bit field starting offset counting
                                        !   from the least significant bit.
        integer*4   field_width         ! Bit field length; values other than 1
                                        !   are not currently implemented.

                                        ! "CALIBRATION" information follows:
                                        !   Calibration of SANDUS digital subchannels
                                        !   is not currently implemented in PRELEWD.

        logical*4   cal_option          ! Calibrate this subchannel;
                                        !   .true.  means calibrate.
        integer*4   cal_size            ! Number of calibration samples.
        logical*4   cal_use(4)          ! Calibration levels to use;
                                        !   .true.  means use this level.
        real*8      cal_level(4)        ! Calibrator levels in volts.
        real*4      cal_eng_units(4)    ! Calibration levels in engineering units.
        real*4      cal_slope           ! Calibration slope.
        real*4      cal_slope_dev       ! Calibration slope deviation.
        integer*4   cal_top             ! Top cal level to use.
        integer*4   cal_bottom          ! Bottom cal level to use.
        integer*4   cal_diff_check      ! Minimum acceptable difference
                                        !   between cal top and cal bottom
                                        !   in counts.
    end structure

!   The following structure describes the SANDUS analog multiplexer channels.
!   There should be one record of this type present for every SANDUS analog
!   multiplexer channel.

    structure /S_AMUX_CHAN_DESC/
        integer*2   chan_num            ! Channel number (octal).
        byte        chan_rec_type       ! Channel record type (22).
        byte        dummy               ! Dummy field (not used).

                                        ! "GENERAL" information follows:

        character*16 sec_desc           ! Channel number in ASCII.
        character*24 last_edit          ! Date of last edit.
```

```
integer*4     data_width          ! Width of the data sample in bits.
                                  !   Used to determine the UBE,
                                  !   the maximum count value.
                                  !   Appears in the plot legend.
integer*4     byte_span           ! Number of bytes per sample.
character*8   data_conv           ! Data conversion;
                                  !   acceptable entries are:
                                  !   'LINEAR', 'EXTENDED'.
                                  !   PRELEWD interprets 'EXTENDED' to mean
                                  !   data_width = 12 and byte_span = 2.
real*8        start_time          ! Sample interval start time.
real*8        stop_time           ! Sample interval stop time.
integer*4     num_sub_chans       ! Number of subchannels;
                                  !   less than or equal to 32.
real*8        relative_delay      ! Relative delay in seconds, a correction
                                  !   added to time values before plotting.
character*4   zero_ref            ! Zero reference; used to identify on plots the
                                  !   basis for the relative_delay correction above.
                                  !   Currently 'COMP'(ton), 'FIDU', and ' ' are
                                  !   acceptable.  Use of this field MAY classify
                                  !   the resulting plot.
logical*4     compress            ! Compress data;
                                  !   .true. indicates that hardware compression
                                  !   of the data has occurred, while
                                  !   .false. indicates NO hardware
                                  !   compression has occurred.
integer*4     nom_devtn           ! Acceptable deviation from nominal;
                                  !   valid only if compress above
                                  !   is .true.  This is a decimal value.
integer*4     delay_count         ! Number of samples before compression;
                                  !   valid only if compress above
                                  !   is .true.
logical*4     log_chan            ! Log this channel;
                                  !   .true.  means that this is a data logger
                                  !   channel and the log_samp_int which follows is
                                  !   to be used for sample interval.
real*8        log_samp_int        ! Log sample rate in seconds;
                                  !   valid only if log_chan above is .true.
character*4   srad_file           ! Indicates whether or not to produce
                                  !   an SRAD file when generating a plot.
                                  !   Acceptable entries are 'NONE', and 'SRAD'.

                                  ! "SETUP" Information follows:

character*8   data_mod_type       ! Module identification;
                                  !   acceptable entries are:
                                  !   'TA591', 'TA592'.
                                  !   Appears in the plot legend.
integer*4     num_ad_conv         ! Number of A/D converters;
                                  !   acceptable entries are 1 or 2.
character*8   mem_realtime        ! Module type;
                                  !   acceptable entries are:
                                  !   'MEMORY', 'REALTIME'.
character*8   mem_size            ! Memory size;
                                  !   acceptable entries are:
                                  !   '4 KB', '8 KB', '16 KB', '1 KB', 'N/A'.
character*4   pretrig_bytes       ! Number of pretrigger bytes.
                                  !   To convert to pretrigger data samples
                                  !   divide pretrig_bytes by byte_span.
```

```
        integer*4    num_samp_ints          ! Number of sample intervals;
                                            !   must be greater than 0 and
                                            !   less than or equal to 2.
        integer*4    num_samps(2)           ! Number of samples per sample interval.
        real*8       samp_int(2)            ! Sample intervals, in seconds.
                                            !   The first value appears in the
                                            !   plot legend.
        character*8  trig_mode              ! Trigger mode;
                                            !   acceptable entries are:
                                            !   'INTERNAL', 'EXT A', 'EXT B',
                                            !   'MANUAL',   'N/A'.
        logical*4    trig_arm               ! Trigger arm;
                                            !   .true.  means the module is armed,
                                            !   .false. means the module is triggered.
        logical*4    trig_inhibit           ! Trigger inhibit;
                                            !   .true.  means trigger inhibited,
                                            !   .false. means trigger not inhibited.
        logical*4    trig_manual            ! Manual trigger;
                                            !   .true.  means set trigger,
                                            !   .false. means clear trigger.
        integer*4    trig_number            ! Trigger number for this channel.
                                            !   This trigger and the ref_trig are used
                                            !   by PROCESS to determine the time of the
                                            !   first sample.

                                            ! "CALIBRATION" Information follows:
                                            !   Calibration of SANDUS analog multiplexer channels
                                            !   is not currently implemented in PRELEWD.
        logical*4    cal_option             ! Calibrate this channel;
                                            !   .true.  means calibrate.
        character*4  cal_object             ! Calibration object;
                                            !   'MAIN' means use main channel cal values
                                            !   and calibrate as an aggregate,
                                            !   'SUB' means use the cal information contained
                                            !   in the subchannel structure(s).
        integer*4    cal_size               ! Number of calibration samples.
        logical*4    cal_use(4)             ! Calibration levels to use;
                                            !   .true.  means use this level.
        real*8       cal_level(4)           ! Calibrator levels in volts.
        real*4       cal_slope              ! Calibration slope.
        real*4       cal_slope_dev          ! Calibration slope deviation.
        integer*4    cal_top                ! Top cal level to use.
        integer*4    cal_bottom             ! Bottom cal level to use.
        integer*4    cal_diff_check         ! Minimum acceptable difference
                                            !   between cal top and cal bottom
                                            !   in counts.

    end structure


!   The following structure describes the SANDUS analog multiplexer subchannels.
!   There should be a record of this type present for every subchannel in a
!   SANDUS analog multiplexer channel (32 maximum).

    structure /S_AMUX_SUB_DESC/
        integer*2    chan_num               ! Main channel number.
        byte         sub_rec_type           ! Subchannel record type (23).
        byte         sub_num                ! Subchannel number (decimal).
```

```
                                    ! "GENERAL" Information follows:

character*16 sec_desc               ! Channel number in ASCII.
character*24 last_edit              ! Date of last edit.
character*12 expmt_id               ! SNL Experiment number.
character*20 expmtr_name            ! Experimenter name.
character*8  expmtr_org             ! Experimenter organization, or
                                    !   acronym if non-Sandia.
character*16 expmtr_note            ! Experimenter plot annotation.
character*8  plot_option            ! Plot option;
                                    !   a character string such as
                                    !   'NONE', 'COUNTS', 'LINEAR', 'YT-1D',
                                    !   'YT-HD', 'TC-K', 'SLIFER', etc.
                                    !   'NONE' requests no plot, 'COUNTS'
                                    !   requests a plot in percent of full scale.
                                    !   'LINEAR' requests a plot of engineering
                                    !   units in the linear form "ay+b".
                                    !   Other values request a plot of engineering
                                    !   units applying a non-linear, special routine.
character*12 y_label                ! Y-axis label for linear plots.
character*12 alt_y_label            ! Y-axis label for non-linear plots.
logical*4    use_start_stop         ! Use the plot start and stop time which follow;
                                    !   .true.  means use plot_start and plot_stop,
                                    !   .false. means the plot start and stop time
                                    !   will come from other sources.
real*8       plot_start             ! Plot start time in seconds;
                                    !   valid only if use_start_stop above is .true.
real*8       plot_stop              ! Plot stop time in seconds;
                                    !   valid only if use_start_stop above is .true.
logical*4    use_min_max            ! Use the y-min and y-max values which follow;
                                    !   .true.  means use y_min and y_max,
                                    !   .false. means the plot y min and max value
                                    !   will come from other sources.
real*4       y_min                  ! Minimum y value to plot;
                                    !   valid only if use_min_max above is .true.,
                                    !   units must agree with y_label or alt_y_label.
real*4       y_max                  ! Maximum y value to plot;
                                    !   valid only if use_min_max above is .true.,
                                    !   units must agree with y_label or alt_y_label.
integer*4    word_disp              ! Word displacement from sync pattern.

                                    ! "CALIBRATION" Information follows:
                                    !   Calibration of SANDUS analog multiplexer
                                    !   subchannels is not currently implemented in PRELEWD.

logical*4    cal_option             ! Calibrate this channel;
                                    !   .true.  means calibrate.
integer*4    cal_size               ! Number of calibration samples.
logical*4    cal_use(4)             ! Calibration levels to use;
                                    !   .true.  means use this level.
real*8       cal_level(4)           ! Calibrator levels in volts.
real*4       cal_eng_units(4)       ! Calibration levels in engineering units.
real*4       cal_slope              ! Calibration slope.
real*4       cal_slope_dev          ! Calibration slope deviation.
integer*4    cal_top                ! Top cal level to use.
integer*4    cal_bottom             ! Bottom cal level to use.
```

```
        integer*4    cal_diff_check          ! Minimum acceptable difference
                                             !   between cal top and cal bottom
                                             !   in counts.
    end structure

!   The following structure describes the Tektronix 7912 channels.  There should
!   be one record of this type present for every 7912 channel.

    structure /T7912_CHAN_DESC/
        integer*2    chan_num                ! Channel number (octal).
        byte         chan_rec_type           ! Channel record type (24).
        byte         rec_use                 ! Record use;
                                             !   0 - For experiment data.
                                             !   1 - For laser calibration.
                                             !   2 - For cable compensation.

                                             ! "GENERAL" Information follows:

        character*16 sec_desc                ! Channel number in ASCII.
        character*24 last_edit               ! Date of last edit.
        character*8  source_loc              ! Information for plot header to
                                             !   enable use of TA668 i.e. '7912-1'.
        integer*4    data_width              ! Width of data sample in bits.
        real*8       start_time              ! Sample interval start time.
        real*8       stop_time               ! Sample interval stop time.
        integer*4    num_sub_chans           ! Number of subchannels.

                                             ! "SETUP" Information follows:

        integer*4    drc_op_code             ! DRC operation code.
        integer*4    gpib_address            ! GPIB address.
        character*4  dt_get                  ! Interface message digitize.
        character*8  intensity               ! Main intensity.
        character*4  focus                   ! Focus.
        character*12 time_div                ! Time per division.
        character*8  h_position              ! Horizontal sweep position.
        character*4  magnifier               ! Sweep magnifier.
        character*4  trig_mode               ! Trigger mode.
        character*4  trig_coupling           ! Trigger signal coupling.
        character*4  trig_slope              ! Trigger slope.
        character*4  trig_source             ! Trigger source.
        character*4  trig_holdoff            ! Trigger holdoff period.
        character*8  trig_level              ! Trigger level.
        character*4  v_coupling              ! Vertical coupling.
        character*12 v_volts_div             ! Vertical volts per division.
        character*4  v_var_sens              ! Vertical variable sensitivity.
        character*4  v_polarity              ! Vertical polarity.
        character*8  v_position              ! Baseline vertical position.
        character*8  tbase_plugin            ! Time Base Plug-In type.
        character*8  vert_plugin             ! Vertical Amplifier Plug-In type.

                                             ! "CALIBRATION" Information follows:

        logical*4    cal_option              ! Calibrate this channel;
                                             !   .true. means calibrate.
        integer*4    trace_width             ! Optimum trace width.
        integer*4    time_tolerance          ! Timebase tolerance.
        integer*4    vert_tolerance          ! Vertical tolerance.
```

```
real*4        baseline_loc        ! Baseline location.
real*4        trace_start         ! Trace start position.
integer*4     num_cal_levels      ! Number of DC calibration levels.
integer*4     cal_size            ! Number of bytes per trace.
logical*4     cal_ext_trig        ! External trigger;
                                  !   .true. means use external trigger.
logical*4     cal_take(8)         ! Cal levels to take; .true. means take.
logical*4     cal_use(8)          ! Cal levels to use; .true. means use.
logical*4     cal_brief(8)        ! Cal levels to take at t-2 hours;
                                  !   (max of 3 can be .true.)
real*8        cal_level(8)        ! Calibrator levels in volts.
integer*4     cal_top             ! Top cal level to use.
integer*4     cal_bottom          ! Bottom cal level to use.
integer*4     cal_diff_check      ! Minimum acceptable difference
                                  !   between cal top and cal bottom
                                  !   in counts.
integer*4     cal_order(8)        ! Order of calibration levels.
integer*4     num_cal_pulses      ! Number of calibration pulses.
logical*4     pulse_take(4)       ! Pulses to take; .true. means take.
logical*4     pulse_use(4)        ! Pulses to use; .true. means use.
logical*4     pulse_brief(4)      ! Pulses to take at t-2 hours;
                                  !   (max of 2 can be .true.).
real*4        pulse_period(4)     ! Pulse period.
real*4        pulse_delay(4)      ! Pulse delay.
real*4        trig_delay(4)       ! Trigger delay.
real*4        pulse_width(4)      ! Pulse width.
real*4        pulse_rise(4)       ! Pulse rise time.
real*4        pulse_fall(4)       ! Pulse fall time.
real*4        pulse_v_high(4)     ! Pulse high volts.
real*4        pulse_v_low(4)      ! Pulse low volts.
logical*4     pulse_double(4)     ! Double pulse; .true. means double pulse.
logical*4     pulse_invert(4)     ! Invert pulse; .true. means invert pulse.
real*4        mid_sine_freq       ! Midscreen sine frequency.
real*4        base_sine_freq      ! Baseline sine frequency.
real*4        mid_sine_amp        ! Midscreen sine amplitude.
real*4        base_sine_amp       ! Baseline sine amplitude.
integer*4     min_cycles          ! Minimum number of complete
                                  !   sine cycles.
character*4   sig_in_spdt         !
character*4   sig_in_sp8t         ! Should be identical to sig_in_spdt.
character*4   sig_dist_sp4t       !
character*4   src_sel_sp4t(4)     !
character*4   dc_atten_spdt       ! DC attenuator address
character*4   rf_atten_spdt       ! RF attenuator address
character*4   pu_atten_spdt       ! Pulse attenuator address
character*4   trig_sw_spdt        !
integer*4     dc_gpib_addr        ! DC voltage source GPIB address.
integer*4     rf_gpib_addr        ! RF sinewave source GPIB address.
integer*4     pu_gpib_addr        ! Pulse source GPIB address.
integer*4     hp_gpib_addr        ! HP3852A GPIB address.
integer*4     tr_gpib_addr        ! Trigger source address.
integer*4     vax_channel         ! VAX GPIB channel number.
character*8   mid_vert_pos        ! Setting for midscreen vertical position.

                                  ! "LASER" Calibration information follows:

character*12  sim_ts_node         ! Terminal server node name.
integer*4     las_cal_cntr        ! Switch Controller address.
integer*4     las_sig_cal         ! Gage signal / Laser cal address.
```

```
        integer*4    las_freq            ! 50 MHz / 1 MHz address.
        integer*4    las_ext             ! Laser input; Internal / External address.
        integer*4    las_lcl_rmt         ! Local / Remote address.
        integer*4    las_lckout          ! Laser lockout address.
        integer*4    las_trig            ! Laser closure trigger address.
        integer*4    las_power           ! Laser Cal ON/OFF power address.
    end structure


!   The following structure describes the Tektronix 7912 subchannels.  There
!   will be one record of this type present for every Tek 7912 subchannel.
!   Each Tek 7912 subchannel is associated with one unique event experiment.
!   There will also be one record of this type designated subchannel 0, which
!   represents the raw data for the entire channel.

    structure /T7912_SUB_DESC/
        integer*2    chan_num            ! Main channel number.
        byte         sub_rec_type        ! Subchannel record type (25).
        byte         sub_num             ! Subchannel number.

                                         ! "GENERAL" Information follows:

        character*16 sec_desc            ! Channel number in ASCII.
        character*24 last_edit           ! Date of last edit.
        character*12 expmt_id            ! SNL Experiment number.
        character*20 expmtr_name         ! Experimenter name.
        character*8  expmtr_org          ! Experimenter organization, or
                                         !   acronym if non-Sandia.
        character*16 expmtr_note         ! Experimenter plot annotation.
        character*8  plot_option         ! Plot option;
                                         !   Acceptable entries are:
                                         !   'NONE', 'COUNTS', 'LINEAR'.
                                         !   'NONE' requests no plot, 'COUNTS'
                                         !   requests a plot in percent of full scale.
                                         !   'LINEAR' requests a plot of engineering
                                         !   units in the linear form "ay+b".
        character*12 y_label             ! Y-axis label for linear plots.
        real*4       conv_factor         ! Conversion factor in
                                         !   engineering units per volt.
        integer*4    expmt_order         ! Experiment order of occurrence.
        real*4       expmt_delay         ! Signal delay inserted for multiplexing.
        real*4       expmt_atten         ! Signal attenuation which results from
                                         !   the inserted signal delay.
        integer*4    expmt_left          ! Experiment left bound in counts.
        integer*4    expmt_right         ! Experiment right bound in counts.
        logical*4    use_start_stop      ! Use the plot start and stop time which follow;
                                         !   .true.  means use plot_start and plot_stop,
                                         !   .false. means the plot start and stop time
                                         !   will come from other sources.
        real*8       plot_start          ! Plot start time in seconds;
                                         !   valid only if use_start_stop above is .true.
        real*8       plot_stop           ! Plot stop time in seconds;
                                         !   valid only if use_start_stop above is .true.
        logical*4    use_min_max         ! Use the y-min and y-max values which follow;
                                         !   .true.  means use y_min and y_max,
                                         !   .false. means the plot y min and max value
                                         !   will come from other sources.
```

```
real*4        y_min              ! Minimum y value to plot;
                                 !   valid only if use_min_max above is .true.,
                                 !   units must agree with y_label or alt_y_label.
real*4        y_max              ! Maximum y value to plot;
                                 !   valid only if use_min_max above is .true.,
                                 !   units must agree with y_label or alt_y_label.
real*8        relative_delay     ! Relative delay in seconds, a correction
                                 !   added to time values before plotting.
character*4   zero_ref           ! Zero reference; used to identify on plots the
                                 !   basis for the relative_delay correction above.
                                 !   Currently 'COMP'(ton), 'FIDU', ' ' are
                                 !   acceptable.  Use of this field MAY classify
                                 !   the resulting plot.
character*4   srad_file          ! Indicates whether or not to produce
                                 !   an SRAD file while generating a plot.
                                 !   Acceptable entries are 'NONE', and 'SRAD'.

                                 ! "K-FACTOR, EQUALIZATION,
                                 !   NORMALIZATION, and TREND" Information follows:

logical*4     apply_kfactor      ! Apply K-factor to this channel;
                                 !   .true. means apply.
real*4        kfactor            ! Measured K-factor.
real*4        kfactor_atten      ! Late change affecting K-factor.
character*80  kfactor_method     ! Description of how K-factor was
                                 !   measured.
logical*4     equalize           ! Apply equalization to this channel;
                                 !   .true. means equalize.
character*48  equalize_file      ! Filename of the equalization
                                 !   function.
logical*4     normalize_time     ! Normalize the time axis;
                                 !   .true. means normalize.
real*4        normalz_factor     ! Normalization factor.
real*4        sig_atten          ! Signal attenuation for TREND.
real*4        sig_term_res       ! Signal termination resistance.
real*4        sig_prop_time      ! Total signal propogation time.
real*4        trg_prop_time      ! Total trigger propogation time.

                                 ! "SIMULATION" Information follows:

logical*4     sim_option         ! External simulation;
                                 !   .true. means simulation applied.
integer*4     num_simulators     ! Number of simulators used to
                                 !   generate the simulation signal,
                                 !   four maximum.
character*16  sim_unique_id(4)   ! Simulator unique identification
                                 !   for each simulator used.
real*8        sim_pp_volts       ! Simulated signal voltage, peak-to-peak
                                 !   being inserted for this experiment.
real*4        insert_atten       ! Attenuator inserted after fanout.
real*4        fanout_atten       ! Fanout attenuation.
real*4        cable_atten        ! Sum of all cable attenuation between
                                 !   the simulation signal generator and
                                 !   this experiment.
integer*4     sim_switch         ! Simulator switch number.
character*16  sim_unique_fo      ! Fiber Optic ID code.
integer*4     sim_fo_switch      ! Fiber Optic switch number.
```

```
        character*8  est_tek_atten          ! Estimated attenuation needed to avoid
                                            !   over driving the Tek 11801.
        integer*4    picosec_atten          ! Attenuation setting, Picosecond generator.
    end structure


!   The following structure is presently unused, it is here for future expansion,
!   and it must be present as a placeholder.

    structure /UNUSED_26_DESC/
        integer*2    struct_rec_num         ! Structure record number.
        byte         struct_rec_type        ! Structure record type (26).
        byte         dummy                  ! Dummy field (not used).
    end structure


!   The following structure is presently unused, it is here for future expansion,
!   and it must be present as a placeholder.

    structure /UNUSED_27_DESC/
        integer*2    struct_rec_num         ! Structure record number.
        byte         struct_rec_type        ! Structure record type (27).
        byte         dummy                  ! Dummy field (not used).
    end structure


!   The following structure describes the Tektronix 7103 channels.  There should
!   be one record of this type present for every 7103 channel.

    structure /T7103_CHAN_DESC/
        integer*2    chan_num               ! Channel number (octal).
        byte         chan_rec_type          ! Channel record type (28).
        byte         rec_use                ! Record use;
                                            !   0 - For experiment data.
                                            !   1 - For laser calibration.
                                            !   2 - For cable compensation.

                                            ! "GENERAL" Information follows:

        character*16 sec_desc               ! Channel number in ASCII.
        character*24 last_edit              ! Date of last edit.
        character*8  source_loc             ! Information for plot header to
                                            !   enable use of TA668 i.e. '7103-1'.
        integer*4    data_width             ! Width of data sample in bits.
        real*8       start_time             ! Sample interval start time.
        real*8       stop_time              ! Sample interval stop time.
        integer*4    num_sub_chans          ! Number of subchannels.

                                            ! "SETUP" Information follows:

        character*8  intensity              ! Main intensity.
        character*4  focus                  ! Focus.
        character*12 time_div               ! Time per division.
        character*8  h_position             ! Horizontal sweep position.
        character*4  magnifier              ! Sweep magnifier.
        character*4  trig_mode              ! Trigger mode.
        character*4  trig_coupling          ! Trigger signal coupling.
        character*4  trig_slope             ! Trigger slope.
        character*4  trig_source            ! Trigger source.
```

```
character*4  trig_holdoff        ! Trigger holdoff period.
character*8  trig_level          ! Trigger level.
character*4  v_coupling          ! Vertical coupling.
character*12 v_volts_div         ! Vertical volts per division.
character*4  v_var_sens          ! Vertical variable sensitivity.
character*4  v_polarity          ! Vertical polarity.
character*8  v_position          ! Baseline vertical position.
character*8  tbase_plugin        ! Time Base Plug-In type.
character*8  vert_plugin         ! Vertical Amplifier Plug-In type.
character*4  readout             !
character*4  readout_gate        !
character*4  grat_gate           !
character*4  grat_illum          !
character*4  mf_trig_src         !
character*4  mf_vert_mode        !

                                 ! "CALIBRATION" Information follows:

logical*4    cal_option          ! Calibrate this channel;
                                 !   .true. means calibrate.
integer*4    trace_width         ! Optimum trace width.
integer*4    time_tolerance      ! Timebase tolerance.
integer*4    vert_tolerance      ! Vertical tolerance.
real*4       baseline_loc        ! Baseline location.
real*4       trace_start         ! Trace start position.
integer*4    num_cal_levels      ! Number of DC calibration levels.
integer*4    cal_size            ! Number of bytes per trace.
logical*4    cal_ext_trig        ! External trigger;
                                 !   .true. means use external trigger.
logical*4    cal_take(8)         ! Cal levels to take; .true. means take.
logical*4    cal_use(8)          ! Cal levels to use; .true. means use.
logical*4    cal_brief(8)        ! Cal levels to take at t-2 hours;
                                 !   (max of 3 can be .true.)
real*8       cal_level(8)        ! Calibrator levels in volts.
integer*4    cal_top             ! Top cal level to use.
integer*4    cal_bottom          ! Bottom cal level to use.
integer*4    cal_diff_check      ! Minimum acceptable difference
                                 !   between cal top and cal bottom
                                 !   in counts.
integer*4    cal_order(8)        ! Order of calibration levels.
integer*4    num_cal_pulses      ! Number of calibration pulses.
logical*4    pulse_take(4)       ! Pulses to take; .true. means take.
logical*4    pulse_use(4)        ! Pulses to use; .true. means use.
logical*4    pulse_brief(4)      ! Pulses to take at t-2 hours;
                                 !   (max of 2 can be .true.).
real*4       pulse_period(4)     ! Pulse period.
real*4       pulse_delay(4)      ! Pulse delay.
real*4       trig_delay(4)       ! Trigger delay.
real*4       pulse_width(4)      ! Pulse width.
real*4       pulse_rise(4)       ! Pulse rise time.
real*4       pulse_fall(4)       ! Pulse fall time.
real*4       pulse_v_high(4)     ! Pulse high volts.
real*4       pulse_v_low(4)      ! Pulse low volts.
logical*4    pulse_double(4)     ! Double pulse; .true. means double pulse.
logical*4    pulse_invert(4)     ! Invert pulse; .true. means invert pulse.
real*4       mid_sine_freq       ! Midscreen sine frequency.
real*4       base_sine_freq      ! Baseline sine frequency.
real*4       mid_sine_amp        ! Midscreen sine amplitude.
real*4       base_sine_amp       ! Baseline sine amplitude.
```

```
        integer*4     min_cycles            ! Minimum number of complete
                                            !   sine cycles.
        character*4   sig_in_spdt           !
        character*4   sig_in_sp8t           ! Should be identical to sig_in_spdt.
        character*4   sig_dist_sp4t         !
        character*4   src_sel_sp4t(4)       !
        character*4   dc_atten_spdt         ! DC attenuator address
        character*4   rf_atten_spdt         ! RF attenuator address
        character*4   pu_atten_spdt         ! Pulse attenuator address
        character*4   trig_sw_spdt          !
        integer*4     dc_gpib_addr          ! DC voltage source GPIB address.
        integer*4     rf_gpib_addr          ! RF sinewave source GPIB address.
        integer*4     pu_gpib_addr          ! Pulse source GPIB address.
        integer*4     hp_gpib_addr          ! HP3852A GPIB address.
        integer*4     tr_gpib_addr          ! Trigger source address.
        integer*4     vax_channel           ! VAX GPIB channel number.
        character*8   mid_vert_pos          ! Setting for midscreen vertical position.

                                            ! "LASER" Calibration information follows:

        character*12 sim_ts_node            ! Terminal server node name.
        integer*4     las_cal_cntr          ! Switch Controller address.
        integer*4     las_sig_cal           ! Gage signal / Laser cal address.
        integer*4     las_freq              ! 50 MHz / 1 MHz address.
        integer*4     las_ext               ! Laser input; Internal / External address.
        integer*4     las_lcl_rmt           ! Local / Remote address.
        integer*4     las_lckout            ! Laser lockout address.
        integer*4     las_trig              ! Laser closure trigger address.
        integer*4     las_power             ! Laser Cal ON/OFF power address.
    end structure

!   The following structure describes the Tektronix 7103 subchannels.  There
!   will be one record of this type present for every Tek 7103 subchannel.
!   Each Tek 7103 subchannel is associated with one unique event experiment.
!   There will also be one record of this type designated subchannel 0, which
!   represents the raw data for the entire channel.

    structure /17103_SUB_DESC/
        integer*2     chan_num              ! Main channel number.
        byte          sub_rec_type          ! Subchannel record type (29).
        byte          sub_num               ! Subchannel number.

                                            ! "GENERAL" Information follows:

        character*16 sec_desc               ! Channel number in ASCII.
        character*24 last_edit              ! Date of last edit.
        character*12 expmt_id               ! SNL Experiment number.
        character*20 expmtr_name            ! Experimenter name.
        character*8   expmtr_org            ! Experimenter organization, or
                                            !   acronym if non-Sandia.
        character*16 expmtr_note            ! Experimenter plot annotation.
        character*8   plot_option           ! Plot option;
                                            !   Acceptable entries are:
                                            !   'NONE', 'COUNTS', 'LINEAR'.
                                            !   'NONE' requests no plot, 'COUNTS'
                                            !   requests a plot in percent of full scale.
                                            !   'LINEAR' requests a plot of engineering
                                            !   units in the linear form "ay+b".
```

```
character*12 y_label          ! Y-axis label for linear plots.
real*4       conv_factor      ! Conversion factor in
                              !   engineering units per volt.
integer*4    expmt_order      ! Experiment order of occurrence.
real*4       expmt_delay      ! Signal delay inserted for multiplexing.
real*4       expmt_atten      ! Signal attenuation which results from
                              !   the inserted signal delay.
integer*4    expmt_left       ! Experiment left bound in counts.
integer*4    expmt_right      ! Experiment right bound in counts.
logical*4    use_start_stop   ! Use the plot start and stop time which follow;
                              !   .true.  means use plot_start and plot_stop,
                              !   .false. means the plot start and stop time
                              !   will come from other sources.
real*8       plot_start       ! Plot start time in seconds;
                              !   valid only if use_start_stop above is .true.
real*8       plot_stop        ! Plot stop time in seconds;
                              !   valid only if use_start_stop above is .true.
logical*4    use_min_max      ! Use the y-min and y-max values which follow;
                              !   .true.  means use y_min and y_max,
                              !   .false. means the plot y min and max value
                              !   will come from other sources.
real*4       y_min            ! Minimum y value to plot;
                              !   valid only if use_min_max above is .true.,
                              !   units must agree with y_label or alt_y_label.
real*4       y_max            ! Maximum y value to plot;
                              !   valid only if use_min_max above is .true.,
                              !   units must agree with y_label or alt_y_label.
real*8       relative_delay   ! Relative delay in seconds, a correction
                              !   added to time values before plotting.
character*4  zero_ref         ! Zero reference; used to identify on plots the
                              !   basis for the relative_delay correction above.
                              !   Currently 'COMP'(ton), 'FIDU', ' ' are
                              !   acceptable.  Use of this field MAY classify
                              !   the resulting plot.
character*4  srad_file        ! Indicates whether or not to produce
                              !   an SRAD file while generating a plot.
                              !   Acceptable entries are 'NONE', and 'SRAD'.

                              ! "K-FACTOR, EQUALIZATION,
                              !   NORMALIZATION, and TREND" information follows:

logical*4    apply_kfactor    ! Apply K-factor to this channel;
                              !   .true. means apply.
real*4       kfactor          ! Measured K-factor.
real*4       kfactor_atten    ! Late change affecting K-factor.
character*80 kfactor_method   ! Description of how K-factor was
                              !   measured.
logical*4    equalize         ! Apply equalization to this channel;
                              !   .true. means equalize.
character*48 equalize_file    ! Filename of the equalization
                              !   function.
logical*4    normalize_time   ! Normalize the time axis;
                              !   .true. means normalize.
real*4       normalz_factor   ! Normalization factor.
real*4       sig_atten        ! Signal attenuation for TREND.
real*4       sig_term_res     ! Signal termination resistance.
real*4       sig_prop_time    ! Total signal propogation time.
real*4       trg_prop_time    ! Total trigger propogation time.
```

```
                                        ! "SIMULATION" information follows:

        logical*4     sim_option        ! External simulation;
                                        !   .true. means simulation applied.
        integer*4     num_simulators    ! Number of simulators used to
                                        !   generate the simulation signal,
                                        !   four maximum.
        character*16 sim_unique_id(4)   ! Simulator unique identification
                                        !   for each simulator used.
        real*8        sim_pp_volts      ! Simulated signal voltage, peak-to-peak
                                        !   being inserted for this experiment.
        real*4        insert_atten      ! Attenuator inserted after fanout.
        real*4        fanout_atten      ! Fanout attenuation.
        real*4        cable_atten       ! Sum of all cable attenuation between
                                        !   the simulation signal generator and
                                        !   this experiment.
        integer*4     sim_switch        ! Simulator switch number.
        character*16 sim_unique_fo      ! Fiber Optic ID code.
        integer*4     sim_fo_switch     ! Fiber Optic switch number.
        character*8   est_tek_atten     ! Estimated attenuation needed to avoid
                                        !   over driving the Tek 11801.
        integer*4     picosec_atten     ! Attenuation setting, Picosecond generator.
    end structure

!   The following structure describes the RTD 720 device.  There should be one
!   record of this type for every RTD 720 device.

    structure /RTD_DEV_DESC/
        integer*2     dev_num           ! Device number (a decimal number
                                        !   that is identical to the GPIB
                                        !   bus address).
        byte          dev_rec_type      ! RTD 720 device record type (30).
        byte          rec_use           ! Record use;
                                        !   0 - For experiment data.
                                        !   1 - For laser calibration.
                                        !   2 - For cable compensation.

                                        ! "GENERAL" device information follows:

        character*24 last_edit          ! Date of last edit.
        character*12 editor             ! Name of RTDTEST editor.
        character*8   source_loc        ! Rack identification which appears on
                                        !   the plot.
        integer*4     data_width        ! Width of data sample in bits.
        character*8    node_name        ! GPIB translator Node name.
        character*8   port_id           ! GPIB translator GPIB port ID.

                                        ! "VERTICAL" commands follow:

        character*4   vmode             ! Vertical mode;
                                        !   CH1, DUAL, or QUAD for
                                        !   1, 2, or 4 hardware channels.

                                        ! "ACQUIRE" commands follow:

        character*8   acq_mode          ! Mode;
                                        !   NORMAL or ADVANCE.
```

```
character*8  acq_state          ! State;
                                !   STOP, RUN, or HLDNXT.
character*8  acq_interval       ! Acquire interval in seconds;
                                !   500E-12, 1E-9, 2E-9 or 4E-9.
integer*4    acq_length         ! Record length;
                                !   512 to maximum memory for each
                                !   channel.
integer*4    acq_nrecord        ! Number of consecutive records
                                !   to fill in ADVANCE mode;
                                !   1 to a maximum of 1024.
character*8  acq_clock          ! Acquire clock source;
                                !   INTRNAL or EXTERNAL.

                                ! "ARM and TRIGGER" commands follow:

character*8  arm                ! Arming source;
                                !   INTRNAL or EXTERNAL.
character*8  trig_mode          ! Trigger mode;
                                !   AUTO or NORMAL.
character*8  trig_coupling      ! Trigger coupling;
                                !   AC, DC, or HFREJ.
real*4       trig_level         ! Trigger level in percent of full
                                !   scale or volts as determined
                                !   by type.
character*8  trig_typelevel     ! Trigger level type;
                                !   PERCENT or VOLTS.
real*4       trig_position      ! Trigger position in counts,
                                !   time, or percent of record
                                !   length as determined by type.
character*8  trig_typepos       ! Trigger position type;
                                !   PERCENT, POINT, or SECOND.
character*8  trig_slope         ! Trigger slope;
                                !   PLUS or MINUS.
character*8  trig_source        ! Trigger source;
                                !   CH1, CH2, CH3, CH4, or EXTERNAL.

                                ! "WAVEFORM PREAMBLE" commands follow:

character*8  wftx_mode          ! Waveform transfer format;
                                !   DL at this time.
character*8  wftx_intrleave     ! Waveform transfer interleave;
                                !   ON or OFF.

                                ! "DATA and WAVEFORM" commands follow:

integer*4    data_cnt_rec       ! Number of records in waveform transfer.
integer*4    data_count         ! Number of points in waveform transfer.
integer*4    data_start         ! Data transfer starting point.
integer*4    data_strecord      ! Selects first record to be transferred.

                                ! "HIGH-SPEED DATA OUTPUT" commands follow:

character*4  hsdo_state         ! High speed port selection;
                                !   ON  - Data is sent to HSDO port,
                                !   OFF - Data is sent to GPIB port.
integer*4    hsdo_txmode        ! HSDO port's handshake mode;
                                !   1 or 2.
```

```
character*12 hsdo_dump          ! HSDO dump mode;
                                !   CONTINUOUS or OFF.

                                ! "STATUS and EVENT" commands follow:

character*4  rqs                ! Enable/Disable SRQ line;
                                !   ON or OFF.
character*4  srq_abstouch       ! Enable/Disable SRQ on the
                                !   following status or error
                                !   conditions;
                                !   ON (Enable) or OFF (Disable).
character*4  srq_cmderr         !   ON or OFF.
character*4  srq_exerr          !   ON or OFF.
character*4  srq_exwarn         !   ON or OFF.
character*4  srq_idprobe        !   ON or OFF.
character*4  srq_inerr          !   ON or OFF.
character*4  srq_inwarn         !   ON or OFF.
character*4  srq_opcmpl         !   ON or OFF.
character*4  srq_usr1           !   ON or OFF.
character*4  srq_usr2           !   ON or OFF.
character*12 uid                ! A string that assigns a name to
                                !   device.
character*80 ident              ! Space for response to the ID? query.

                                ! "GPIB RELATED" commands follow:

character*8  abstouch           ! CLEAR or a string representing
                                !   x,y coordinates of a simulated
                                !   button push.
character*4  debug_gpib         ! Sets the state of GPIB debugging;
                                !   ON or OFF.
character*8  dt                 ! Sets the acquisition state;
                                !   RUN, STOP, HLDNXT, or OFF.
character*8  init               ! Initiation source;
                                !   PANEL, GPIB, or ALL.
character*4   longform          ! Longform command;
                                !   ON or OFF.
character*4  path               ! Path command;
                                !   ON or OFF.
character*12 user1(2)           ! Quoted strings to be displayed.
character*12 user2(2)           ! Quoted strings to be displayed.

                                ! "CURSOR" commands follow:

character*4  crs1_loctn         ! WINx, where x = (1,2,3,4).
character*4  crs2_loctn         ! WINx, where x = (1,2,3,4).
integer*4    crs1_xpoint        ! 0 to record_length-1.
integer*4    crs2_xpoint        ! 0 to record_length-1.
character*8  cref               ! FIRST or SAME.
character*8  crsd_typetime      ! HZ or SECOND.
character*4  cursors            ! ON or OFF.

                                ! Preamble data from the device:

integer*4    pt_off             ! Number of points between trigger
                                !   and first point transmitted.
integer*4    xzero              ! Horizontal zero point (always 0).
integer*4    yoff               ! Vertical binary offset (always 128).
```

```
        real*4      xincr               ! Horizontal sample interval (seconds).

                                        ! Waveform data from the device:

        real*4      trfract             ! Fraction of sample interval in which trigger
                                        ! occurred.  Possible values are as follows;
                                        ! CH1: 0; DUAL: 0., .5; QUAD: 0., .25, .5, .75.
    end structure


!   The following structure describes the RTD 720 channels.  There should
!   be one record of this type present for every RTD 720 channel.  If a
!   RTD 720 digitizer is operating in "QUAD" mode with four channels (traces),
!   then there will be four of these records for that digitizer.

    structure /RTD_CHAN_DESC/
        integer*2   chan_num            ! Channel number (combination of the
                                        !   decimal GPIB bus address and the
                                        !   RTD 720 channel number).
        byte        chan_rec_type       ! Channel record type (31).
        byte        rec_use             ! Record use;
                                        !   0 - For experiment data.
                                        !   1 - For laser calibration.
                                        !   2 - For cable compensation.

                                        ! "GENERAL" information follows:

        character*16 sec_desc           ! Channel number in ASCII.
        character*24 last_edit          ! Date of last edit.
        character*12 editor             ! Name of RTDTEST editor.
        integer*4   num_sub_chans       ! Number of subchannels for this
                                        !   hardware channel (>=1).

                                        ! "VERTICAL SETUP" commands follow:

        character*8 ch_range            ! Full scale vertical range setting;
                                        !   0.25 to 20.0 V.
        character*8 ch_offset           ! Input offset in percent of full
                                        !   scale or volts as determined
                                        !   by type;
                                        !   +/- 1.0 V to +/- 10.0 V.
        character*8 ch_typeoffset       ! Offset type;
                                        !   PERCENT or VOLTS.
        character*4 ch_coupling         ! Vertical coupling;
                                        !   AC, DC, or OFF.
        character*8 ch_bwlim            ! Bandwidth limiting filter;
                                        !   FULL, HUNDRED, or TWENTY.

                                        ! "DATA and WAVEFORM" commands follow:

        character*4 data_ch             ! Waveform data from this channel;
                                        !   ON or OFF.

                                        ! Preamble data from the channel:

        real*4      ymult               ! Vertical scale factor (volts per count).
        real*4      yzero               ! Vertical offset of the wave form (volts).
```

```
                                        ! "LASER" Calibration information follows:

        character*12 sim_ts_node        ! Terminal server node name.
        integer*4    las_cal_cntr       ! Switch Controller address.
        integer*4    las_sig_cal        ! Gage signal / Laser cal address.
        integer*4    las_freq           ! 50 MHz / 1 MHz address.
        integer*4    las_ext            ! Laser input; Internal / External address.
        integer*4    las_lcl_rmt        ! Local / Remote address.
        integer*4    las_lckout         ! Laser lockout address.
        integer*4    las_trig           ! Laser closure trigger address.
        integer*4    las_power          ! Laser Cal ON/OFF power address.
    end structure


!   The following structure describes the RTD 720 Experiment subchannels.
!   There will be one record of this type present for every RTD 720 subchannel.
!   Each RTD 720 subchannel is associated with one unique event experiment.
!   There will also be one record of this type designated subchannel 0, which
!   represents the raw data for the entire channel.

    structure /RTD_EXP_SUB_DESC/
        integer*2    chan_num           ! Channel number (combination of the
                                        !   decimal GPIB bus address and the
                                        !   RTD 720 channel number).
        byte         sub_rec_type       ! Subchannel record type (32).
        byte         sub_num            ! Subchannel number.

                                        ! "GENERAL" Information follows:

        character*16 sec_desc           ! Channel number in ASCII.
        character*24 last_edit          ! Date of last edit.
        character*12 editor             ! Name of RTDTEST editor.
        character*12 expmt_id           ! SNL Experiment number.
        character*20 expmtr_name        ! Experimenter name.
        character*8  expmtr_org         ! Experimenter organization, or
                                        !   acronym if non-Sandia.
        character*16 expmtr_note        ! Experimenter plot annotation.
        character*8  plot_option        ! Plot option;
                                        !   Acceptable entries are:
                                        !   'NONE', 'COUNTS', 'LINEAR'.
                                        !   'NONE' requests no plot, 'COUNTS'
                                        !   requests a plot in percent of full scale.
                                        !   'LINEAR' requests a plot of engineering
                                        !   units in the linear form "ay+b".
        character*12 y_label            ! Y-axis label for linear plots.
        real*4       conv_factor        ! Gage conversion factor in
                                        !   engineering units per volt.
        integer*4    expmt_order        ! Experiment order of occurrence.
        real*4       expmt_delay        ! Signal delay inserted for multiplexing.
        real*4       expmt_atten        ! Signal attenuation which results from
                                        !   the inserted signal delay.
        integer*4    expmt_left         ! Experiment left bound in counts.
        integer*4    expmt_right        ! Experiment right bound in counts.
        logical*4    use_start_stop     ! Use the plot start and stop time which follow;
                                        !   .true.  means use plot_start and plot_stop,
                                        !   .false. means the plot start and stop time
                                        !   will come from other sources.
        real*8       plot_start         ! Plot start time in seconds;
                                        !   valid only if use_start_stop above is .true.
```

```
real*8        plot_stop              ! Plot stop time in seconds;
                                     !   valid only if use_start_stop above is .true.
logical*4     use_min_max            ! Use the y-min and y-max values which follow;
                                     !   .true.  means use y_min and y_max,
                                     !   .false. means the plot y min and max value
                                     !   will come from other sources.
real*4        y_min                  ! Minimum y value to plot;
                                     !   valid only if use_min_max above is .true.,
                                     !   units must agree with y_label or alt_y_label.
real*4        y_max                  ! Maximum y value to plot;
                                     !   valid only if use_min_max above is .true.,
                                     !   units must agree with y_label or alt_y_label.
real*8        relative_delay         ! Relative delay in seconds, a correction
                                     !   added to time values before plotting.
character*4   zero_ref               ! Zero reference; used to identify on plots the
                                     !   basis for the relative_delay correction above.
                                     !   Currently 'COMP'(ton), 'FIDU', ' ' are
                                     !   acceptable.  Use of this field MAY classify
                                     !   the resulting plot.
character*4   srad_file              ! Indicates whether or not to produce
                                     !   an SRAD file while generating a plot.
                                     !   Acceptable entries are 'NONE', and 'SRAD'.

                                     ! "K-FACTOR, EQUALIZATION,
                                     !   NORMALIZATION, and TREND" Information follows:

logical*4     apply_kfactor          ! Apply K-factor to this channel;
                                     !   .true. means apply.
real*4        kfactor                ! Measured K-factor.
real*4        kfactor_atten          ! Late change affecting K-factor.
character*80  kfactor_method         ! Description of how K-factor was
                                     !   measured.
logical*4     equalize               ! Apply equalization to this channel;
                                     !   .true. means equalize.
character*48  equalize_file          ! Filename of the equalization
                                     !   function.
logical*4     normalize_time         ! Normalize the time axis;
                                     !   .true. means normalize.
real*4        normalz_factor         ! Normalization factor.
real*4        sig_atten               ! Signal attenuation for TREND.
real*4        sig_term_res           ! Signal termination resistance.
real*4        sig_prop_time          ! Total signal propogation time.
real*4        trg_prop_time          ! Total trigger propogation time.

                                     ! "SIMULATION" Information follows:

logica.*4     sim_option             ! External simulation;
                                     !   .true. means simulation applied.
integer*4     num_simulators         ! Number of simulators used to
                                     !   generate the simulation signal,
                                     !   four maximum.
character*16 sim_unique_id(4)        ! Simulator unique identification
                                     !   for each simulator used.
real*8        sim_pp_volts           ! Simulated signal voltage, peak-to-peak
                                     !   being inserted for this experiment.
real*4        insert_atten           ! Attenuator inserted after fanout.
real*4        fanout_atten           ! Fanout attenuation.
```

```
        real*4       cable_atten          ! Sum of all cable attenuation between
                                          !   the simulation signal generator and
                                          !   this experiment.
        integer*4    sim_switch           ! Simulator switch number.
        character*16 sim_unique_fo        ! Fiber Optic ID code.
        integer*4    sim_fo_switch        ! Fiber Optic switch number.
        character*8  est_tek_atten        ! Estimated attenuation needed to avoid
                                          !   over driving the Tek 11801.
        integer*4    picosec_atten        ! Attenuation setting, Picosecond generator.
    end structure


!   The following structure describes the RTD 720 Laser Calibration subchannels.
!   There will be one record of this type present for every RTD 720 subchannel.
!   There will also be one record of this type designated subchannel 0, which
!   represents the raw data for the entire channel.

    structure /RTD_LC_SUB_DESC/
        integer*2    chan_num             ! Channel number (combination of the
                                          !   decimal GPIB bus address and the
                                          !   RTD 720 channel number).
        byte         sub_rec_type         ! Subchannel record type (33).
        byte         sub_num              ! Subchannel number.

                                          ! "GENERAL" Information follows:

        character*16 sec_desc             ! Channel number in ASCII.
        character*24 last_edit            ! Date of last edit.
        character*12 editor               ! Name of RTDTEST editor.
        character*12 expmt_id             ! SNL Experiment number.
        character*20 expmtr_name          ! Experimenter name.
        character*8  expmtr_org           ! Experimenter organization, or
                                          !   acronym if non-Sandia.
        character*16 expmtr_note          ! Experimenter plot annotation.
        character*8  plot_option          ! Plot option;
                                          !   Acceptable entries are:
                                          !   'NONE', 'COUNTS', 'LINEAR'.
                                          !   'NONE' requests no plot, 'COUNTS'
                                          !   requests a plot in percent of full scale.
                                          !   'LINEAR' requests a plot of engineering
                                          !   units in the linear form "ay+b".
        character*12 y_label              ! Y-axis label for linear plots.
        real*4       conv_factor          ! Gage conversion factor in
                                          !   engineering units per volt.
        integer*4    expmt_order          ! Experiment order of occurrence.
        real*4       expmt_delay          ! Signal delay inserted for multiplexing.
        real*4       expmt_atten          ! Signal attenuation which results from
                                          !   the inserted signal delay.
        integer*4    expmt_left           ! Experiment left bound in counts.
        integer*4    expmt_right          ! Experiment right bound in counts.
        logical*4    use_start_stop       ! Use the plot start and stop time which follow;
                                          !   .true.  means use plot_start and plot_stop,
                                          !   .false. means the plot start and stop time
                                          !   will come from other sources.
        real*8       plot_start           ! Plot start time in seconds;
                                          !   valid only if use_start_stop above is .true.
        real*8       plot_stop            ! Plot stop time in seconds;
                                          !   valid only if use_start_stop above is .true.
```

```
logical*4     use_min_max          ! Use the y-min and y-max values which follow;
                                   !   .true.  means use y_min and y_max,
                                   !   .false. means the plot y min and max value
                                   !   will come from other sources.
real*4        y_min                ! Minimum y value to plot;
                                   !   valid only if use_min_max above is .true.,
                                   !   units must agree with y_label or alt_y_label.
real*4        y_max                ! Maximum y value to plot;
                                   !   valid only if use_min_max above is .true.,
                                   !   units must agree with y_label or alt_y_label.
real*8        relative_delay       ! Relative delay in seconds, a correction
                                   !   added to time values before plotting.
character*4   zero_ref             ! Zero reference; used to identify on plots the
                                   !   basis for the relative_delay correction above.
                                   !   Currently 'COMP'(ton), 'FIDU', ' ' are
                                   !   acceptable.  Use of this field MAY classify
                                   !   the resulting plot.
character*4   srad_file            ! Indicates whether or not to produce
                                   !   an SRAD file while generating a plot.
                                   !   Acceptable entries are 'NONE', and 'SRAD'.

                                   ! "K-FACTOR, EQUALIZATION,
                                   !   NORMALIZATION, and TREND" Information follows:

logical*4     apply_kfactor         ! Apply K-factor to this channel;
                                   !   .true. means apply.
real*4        kfactor              ! Measured K-factor.
real*4        kfactor_atten        ! Late change affecting K-factor.
character*80  kfactor_method       ! Description of how K-factor was
                                   !   measured.
logical*4     equalize             ! Apply equalization to this channel;
                                   !   .true. means equalize.
character*48  equalize_file        ! Filename of the equalization
                                   !   function.
logical*4     normalize_time       ! Normalize the time axis;
                                   !   .true. means normalize.
real*4        normalz_factor       ! Normalization factor.
real*4        sig_atten             ! Signal attenuation for TREND.
real*4        sig_term_res         ! Signal termination resistance.
real*4        sig_prop_time        ! Total signal propogation time.
real*4        trg_prop_time        ! Total trigger propogation time.

                                   ! "SIMULATION" Information follows:

logical*4     sim_option           ! External simulation;
                                   !   .true. means simulation applied.
integer*4     num_simulators       ! Number of simulators used to
                                   !   generate the simulation signal,
                                   !   four maximum.
character*16  sim_unique_id(4)     ! Simulator unique identification
                                   !   for each simulator used.
real*8         sim_pp_volts        ! Simulated signal voltage, peak-to-peak
                                   !   being inserted for this experiment.
real*4         insert_atten        ! Attenuator inserted after fanout.
real*4        fanout_atten         ! Fanout attenuation.
real*4        cable_atten          ! Sum of all cable attenuation between
                                   !   the simulation signal generator and
                                   !   this experiment.
integer*4     sim_switch           ! Simulator switch number.
```

```
        character*16 sim_unique_fo          ! Fiber Optic ID code.
        integer*4    sim_fo_switch          ! Fiber Optic switch number.
        character*8  est_tek_atten          ! Estimated attenuation needed to avoid
                                            !   over driving the Tek 11801.
        integer*4    picosec_atten          ! Attenuation setting, Picosecond generator.
    end structure


!   The following structure describes the RTD 720 Cable Compensation subchannels.
!   There will be one record of this type present for every RTD 720 subchannel.
!   There will also be one record of this type designated subchannel 0, which
!   represents the raw data for the entire channel.

    structure /RTD_CC_SUB_DESC/
        integer*2    chan_num               ! Channel number (combination of the
                                            !   decimal GPIB bus address and the
                                            !   RTD 720 channel number).
        byte         sub_rec_type           ! Subchannel record type (34).
        byte         sub_num                ! Subchannel number.

                                            ! "GENERAL" Information follows:

        character*16 sec_desc               ! Channel number in ASCII.
        character*24 last_edit              ! Date of last edit.
        character*12 editor                 ! Name of RTDTEST editor.
        character*12 expmt_id               ! SNL Experiment number.
        character*20 expmtr_name            ! Experimenter name.
        character*8  expmtr_org             ! Experimenter organization, or
                                            !   acronym if non-Sandia.
        character*16 expmtr_note            ! Experimenter plot annotation.
        character*8  plot_option            ! Plot option;
                                            !   Acceptable entries are:
                                            !   'NONE', 'COUNTS', 'LINEAR'.
                                            !   'NONE' requests no plot, 'COUNTS'
                                            !   requests a plot in percent of full scale.
                                            !   'LINEAR' requests a plot of engineering
                                            !   units in the linear form "ay+b".
        character*12 y_label                ! Y-axis label for linear plots.
        real*4       conv_factor            ! Gage conversion factor in
                                            !   engineering units per volt.
        integer*4    expmt_order            ! Experiment order of occurrence.
        real*4       expmt_delay            ! Signal delay inserted for multiplexing.
        real*4       expmt_atten            ! Signal attenuation which results from
                                            !   the inserted signal delay.
        integer*4    expmt_left             ! Experiment left bound in counts.
        integer*4    expmt_right            ! Experiment right bound in counts.
        logical*4    use_start_stop         ! Use the plot start and stop time which follow;
                                            !   .true.  means use plot_start and plot_stop,
                                            !   .false. means the plot start and stop time
                                            !   will come from other sources.
        real*8       plot_start             ! Plot start time in seconds;
                                            !   valid only if use_start_stop above is .true.
        real*8       plot_stop              ! Plot stop time in seconds;
                                            !   valid only if use_start_stop above is .true.
        logical*4    use_min_max            ! Use the y-min and y-max values which follow;
                                            !   .true.  means use y_min and y_max,
                                            !   .false. means the plot y min and max value
                                            !   will come from other sources.
```

```
real*4        y_min                ! Minimum y value to plot;
                                   !   valid only if use_min_max above is .true.,
                                   !   units must agree with y_label or alt_y_label.
real*4        y_max                ! Maximum y value to plot;
                                   !   valid only if use_min_max above is .true.,
                                   !   units must agree with y_label or alt_y_label.
real*8        relative_delay       ! Relative delay in seconds, a correction
                                   !   added to time values before plotting.
character*4   zero_ref             ! Zero reference; used to identify on plots the
                                   !   basis for the relative_delay correction above.
                                   !   Currently 'COMP'(ton), 'FIDU', ' ' are
                                   !   acceptable.  Use of this field MAY classify
                                   !   the resulting plot.
character*4   srad_file            ! Indicates whether or not to produce
                                   !   an SRAD file while generating a plot.
                                   !   Acceptable entries are 'NONE', and 'SRAD'.

                                   ! "K-FACTOR, EQUALIZATION,
                                   !   NORMALIZATION, and TREND" Information follows:

logical*4     apply_kfactor         ! Apply K-factor to this channel;
                                   !   .true. means apply.
real*4        kfactor              ! Measured K-factor.
real*4        kfactor_atten        ! Late change affecting K-factor.
character*80  kfactor_method       ! Description of how K-factor was
                                   !   measured.
logical*4     equalize             ! Apply equalization to this channel;
                                   !   .true. means equalize.
character*48  equalize_file        ! Filename of the equalization
                                   !   function.
logical*4     normalize_time       ! Normalize the time axis;
                                   !   .true. means normalize.
real*4        normalz_factor       ! Normalization factor.
real*4        sig_atten             ! Signal attenuation for TREND.
real*4        sig_term_res         ! Signal termination resistance.
real*4        sig_prop_time        ! Total signal propogation time.
real*4        trg_prop_time        ! Total trigger propogation time.

                                   ! "SIMULATION" Information follows:

logical*4     sim_option           ! External simulation;
                                   !   .true. means simulation applied.
integer*4     num_simulators       ! Number of simulators used to
                                   !   generate the simulation signal,
                                   !   four maximum.
character*16  sim_unique_id(4)     ! Simulator unique identification
                                   !   for each simulator used.
real*8         sim_pp_volts        ! Simulated signal voltage, peak-to-peak
                                   !   being inserted for this experiment.
real*4         insert_atten        ! Attenuator inserted after fanout.
real*4        fanout_atten         ! Fanout attenuation.
real*4        cable_atten          ! Sum of all cable attenuation between
                                   !   the simulation signal generator and
                                   !   this experiment.
integer*4     sim_switch           ! Simulator switch number.
character*16  sim_unique_fo        ! Fiber Optic ID code.
integer*4     sim_fo_switch        ! Fiber Optic switch number.
```

```
        character*8  est_tek_atten          ! Estimated attenuation needed to avoid
                                            !   over driving the Tek 11801.
        integer*4    picosec_atten          ! Attenuation setting, Picosecond generator.
      end structure

!     End of TABLE_STRUCTS Listing.
```

APPENDIX B

INSTRUMENT CONTROL FILE (TABLE) UTILITY ROUTINES

APPENDIX B
INSTRUMENT CONTROL FILE (TABLE) UTILITY ROUTINES

B.1 INTRODUCTION

The files that were called tables are now called Instrument Control Files or ICFs.[1] The ICFs are implemented as standard RMS keyed-access indexed files containing variable length records. This file organization was chosen because it allows both random access and true variable length records. The first four bytes of each record are reserved for a numeric key that is used to index into the file. These routines require that the ICF exist. To create the file, which is then expanded into an ICF, use the utility program CREATE_TBL, described in B.4.9.

B.2 ICF UTILITY ROUTINES

There are ten routines available for use with the ICFs. Only five of them are intended for general use by application programs. They are:

(1) **OPEN_TBL** - opens an existing ICF.
(2) **READ_TBL_REC** - reads a record from an ICF.
(3) **MODIFY_TBL_REC** - rewrites a record to an ICF.
(4) **TBL_CHAN_DIR** - returns a directory of the channels in this ICF.
(5) **CLOSE_TBL** - closes an ICF.

They are described in this section. Three other routines are used to add records to an ICF. They are:

(6) **ADD_TBL_REC** - adds a new record to an ICF.
(7) **DELETE_TBL_REC** - deletes an existing record from an ICF.
(8) **GET_TBL_INFO** - returns information about fields and records in the ICF.

---

[1] Much of this document and all of the routines described herein were written by Jon Anspach when he worked for Organization 9321. These routines have been modified when necessary and documentation updated by Peter Kaestner of Department 9321. The files that were called tables are now Instrument Control Files (ICFs).

The remaining two routines are SANDUS specific and are not described in this document. They are:

(9) **CNV_STATUS_TBL** - converts SANDUS status bytes to ICF entries.
(10) **CNV_TBL_STATUS** - converts ICF entries to SANDUS status bytes.

All of the routines are implemented as integer*4 functions that return status values to a longword (integer*4) in the calling program. The status values indicate whether the functions succeeded or failed. For more information on return status values, see the Implementation paragraph, B.5

B.3 STRUCTURE NAMES

When a structure name is requested as an argument, it must be a character variable (or literal) whose value is a valid structure name. Valid structure names come from the file TABLE_STRUCTS.DEF. The names of the record types are the same as the structure names in TABLE_STRUCTS.DEF. Structure names that start with UNUSED... are place holders and are for future use only. Currently valid structure names are:

(1) **STRUCT_DESC** - keeps an internal description of the ICF in the ICF.
(2) **GEN_DESC** - contains general information about the table, test, and source.
(3) **STREAM_DESC** - describes a stream coming from the source.
(4) **MM_DESC** - describes a Mass Memory input stream.
(5) **HDDR_DESC** - describes an HDDR input stream.
(6) **GEAR_DESC** - describes a GEAR collection system.
(7) **SOURCE_DESC** - describes the data source.
(8) **UNUSED_08_DESC** - currently unused, exists as a placeholder.
(9) **UNUSED_09_DESC** - currently unused, exists as a placeholder.
(10) **UNUSED_10_DESC** - currently unused, exists as a placeholder.
(11) **UNUSED_11_DESC** - currently unused, exists as a placeholder.
(12) **SIMULATOR_DESC** - describes a family of simulation sources.
(13) **S_TRIG_DESC** - contains the time mapping of the trigger sources.
(14) **UNUSED_14_DESC** - currently unused, exists as a placeholder.
(15) **UNUSED_15_DESC** - currently unused, exists as a placeholder.
(16) **UNUSED_16_DESC** - currently unused, exists as a placeholder.
(17) **PRI_TYPE_DESC** - names the channel type for all the source channels.
(18) **TRIGGER_CHAN_DESC** - describes a trigger counter channel.
(19) **S_ALOG_CHAN_DESC** - describes a SANDUS ANALOG channel.

(20) **S_DIG_CHAN_DESC** - describes a SANDUS DIGITAL channel.
(21) **S_DIG_SUB_DESC** - describes a SANDUS DIGITAL subchannel.
(22) **S_AMUX_CHAN_DESC** - describes a SANDUS ANALOG MUX channel.
(23) **S_AMUX_SUB_DESC** - describes a SANDUS ANALOG MUX subchannel.
(24) **T7912_CHAN_DESC** - describes a Tektronix 7912 channel.
(25) **T7912_SUB_DESC** - describes a Tektronix 7912 subchannel.
(26) **UNUSED_26_DESC** - currently unused, exists as a placeholder.
(27) **UNUSED_27_DESC** - currently unused, exists as a placeholder.
(28) **T7103_CHAN_DESC** - describes a Tektronix 7103 channel.
(29) **T7103_SUB_DESC** - describes a Tektronix 7103 subchannel.
(30) **RTD_DEV_DESC** - describes a Tektronix RTD720 device.
(31) **RTD_CHAN_DESC** - describes a RTD720 channel.
(32) **RTD_EXP_DESC** - describes a RTD720 data subchannel.
(33) **RTD_LS_SUB_DESC** - describes a RTD720 laser calibration subchannel.
(34) **RTD_CC_SUB_DESC** - describes a RTD720 cable compensation subchannel.

For example, in order to read a Tektronix 7912 channel record from an ICF, the user must supply the string 'T7912_CHAN_DESC' as the 'rec_name' argument.

B.4 USER CALLABLE ROUTINES

B.4.1 Open_Tbl

Open_Tbl is an integer*4 function that opens an ICF. It will not create a new ICF, so the ICF must already exist. Open_Tbl returns as the function value a status code that indicates success or failure. Open_Tbl and the variable to receive the return status must be declared in the calling program as integer*4 variables.

The calling format for Open_Tbl is:

```
status = Open_Tbl ( file_spec,
                    lun,
                    access)
```

where the arguments are:

(1) file_spec

name: file_spec
type: character string
access: read only

File specification of the ICF. If the file type is not specified, it defaults to .TBL. There is no default for the file name.

(2) lun

name: lun
type: integer longword
access: read only

Logical unit number to assign to the ICF. The logical unit number is used by the other routines.

(3) access

name: access
type: character string
access: read only

File access mode. If the user wants to write to the ICF, the access argument must be 'WRITE.' Specify 'READ' to only read the file. The ICFs are read-shareable; that is, they can be read by more than one program simultaneously. However, they are not write-shareable, so only one program at a time can write to an ICF.

B.4.2 Read_Tbl_Rec

Read_Tbl_Rec is an integer*4 function that reads a record from the ICF. The user's FORTRAN record variable name must be one of the record structures in the file TABLE_STRUCTS.DEF. Include the file TABLE_STRUCTS.DEF file in the program and declare a record variable for

each type of record to be written from the ICF. For example, to read 7912 channel records, use the following lines (substituting the appropriate record variable name for t7912_rec):

```
include 'INCLUDE$INCLUDE:TABLE_STRUCTS.DEF'
record /T7912_CHAN_DESC/ t7912_rec
```

where INCLUDE$INCLUDE has been defined as:

$ DEFINE INCLUDE$INCLUDE LD:[INCLUDE] (or wherever the files are maintained)

Only SANDUS ANALOG channels have NO subchannels associated with them. Each type of subchannel has its own record structure, and each subchannel has its own record in the ICF. In order to read a subchannel record, the user must specify a subchannel number in the call to Read_Tbl_Rec.

Finally, since there may be more than one record in the ICF that matches the record type to be read, the user must also supply an index number that indicates which record of that type is to be read. In the case of a channel description record, the index is simply the channel number. Read_Tbl_Rec returns as the function value a status code that indicates success or failure. Read_Tbl_Rec and the variable to receive the return status must be declared in the calling program as integer*4 variables.

The calling format for Read_Tbl_Rec is:

```
status = Read_Tbl_Rec (  lun,
                         rec_name,
                         rec_index,
                         sub_rec,
                         rec_var)
```

where the arguments are:

(1) lun

name: lun
type: integer longword
access: read only

Logical unit number of the ICF. This is the same logical unit number supplied to Open_Tbl.

(2) rec_name

name: rec_name
type: character string
access: read only

Name of the record (or structure) type to read. This name must be the name of a structure definition in TABLE_STRUCTS.DEF or Read_Tbl_Rec will return an error code.

(3) rec_index

name: rec_index
type: integer longword
access: read only

Record index to use. If there are multiple occurrences of some record types in an ICF, Read_Tbl_Rec uses rec_index to find the desired record. If there can only be one occurrence of the record in the ICF, rec_index is ignored.

(4) sub_rec

name: sub_rec
type: integer longword
access: read only

Subchannel record number. This argument is applicable to SANDUS DIGITAL and ANALOG MUX channels, and to Tektronix 7912, and 7103, and the RTD 720 channels. It does NOT apply to SANDUS ANALOG channels. For all other records, this argument should be zero.

(5) rec_var

name: rec_var
type: record variable
access: write only

FORTRAN record variable to receive the ICF record. 'rec_var' must be declared using one of the structures defined in TABLE_STRUCTS.DEF.

B.4.3 Modify_Tbl_Rec

Modify_Tbl_Rec is an integer*4 function that writes a modified record into the ICF. The user's FORTRAN record variable must be declared, using the name of one of the record structures in the file TABLE_STRUCTS.DEF. Include the TABLE_STRUCTS.DEF file in the program and declare a record variable for each record type to be written to the ICF. For example, to write SANDUS channel records, use the following lines (substituting the appropriate record variable name for sandus_rec):

```
include 'INCLUDE$INCLUDE:TABLE_STRUCTS.DEF'
record /S_ALOG_CHAN_DESC/ sandus_rec
```

To use Modify_Tbl_Rec, supply the name of the record type (or structure) to be written. The names of the record types are the same as the structure names in TABLE_STRUCTS.DEF. For example, in order to read a SANDUS ANALOG channel record, supply the string 'S_ALOG_CHAN_DESC' as an argument. Certain channels have subchannels associated with them. Each type of subchannel has its own record structure, and each subchannel has its own record in the ICF. In order to read a subchannel record, specify a subchannel number in the call to Read_Tbl_Rec.

Finally, because there may be more than one record in the ICF that matches the record type needed, supply an index number that indicates which record of that type to write. In the case of a channel description record the index is simply the channel number.

Modify_Tbl_Rec returns as the function value a status code that indicates success or failure. Modify_Tbl_Rec and the variable to receive the return status must be declared in the calling program as integer*4 variables.

The calling format for Modify_Tbl_Rec is:

```
status = Modify_Tbl_Rec ( lun,
                          rec_name,
                          rec_index,
                          sub_rec,
                          rec_var,
                          audit)
```

where the arguments are:

(1) lun

name: lun
type: integer longword
access read only

Logical unit number of the ICF. This is the same logical unit number supplied to Open_Tbl.

(2) rec_name

name: rec_name
type: character string
access: read only

Name of the record (or structure) type to write. This name must be the name of a structure defined in TABLE_STRUCTS.DEF or Modify_Tbl_Rec will return an error code.

(3) rec_index

name: rec_index
type: integer longword
access: read only

Record index to use. If there are multiple occurrences of a certain record type in an ICF, Modify_Tbl_Rec uses rec_index to find the desired record. If there can only be one occurrence of the record in the ICF, rec_index is ignored.

(4) sub_rec

name: sub_rec
type: integer longword
access: read only

Subchannel record number. This argument is applicable to SANDUS DIGITAL and ANALOG MUX channels, and to Tektronix 7912, and 7103, and the RTD 720 channels. It does NOT apply to SANDUS ANALOG channels. For all other records, this argument should be zero.

(5) rec_var

name: rec_var
type: record variable
access: read only

FORTRAN record variable to receive the ICF record. rec_var must be declared, using one of the structures defined in TABLE_STRUCTS.DEF.

(6) audit

name: audit
type: logical longword
access: read only

Flag that indicates whether or not to write audit trail information to the ICF. Under most circumstances, this argument should have the value TRUE.

B.4.4 Tbl_Chan_Dir

Tbl_Chan_Dir is an integer*4 function that returns a directory of channels contained in the ICF. The directory is in the form of an integer array with values that are channel numbers existing in the ICF. There may be up to 512 values returned in the directory array; one for every possible channel record in the ICF.

The user may specify a channel type in the argument list. Tbl_Chan_Dir will return only the channels that match that channel type. If the user does not specify a channel type, all channels in the directory will be returned. The channel types are defined in TABLE_STRUCTS.DEF within the PRI_TYPE_DESC record structure. The following types are currently defined: overhead, trigger, dummy, SANDUS ANALOG, SANDUS DIGITAL, SANDUS ANALOG MUX, Tektronix 7912, Tektronix 7103, and RTD 720.

Tbl_Chan_Dir returns as the function value a status code that indicates success or failure. Tbl_Chan_Dir and the variable to receive the return status must be declared in the calling program as integer*4 variables.

The calling format for Tbl_Chan_Dir is:

```
status = Tbl_Chan_Dir (   lun,
                          chan_type,
                          dir,
                          dir_size,
                          num_chans)
```

where the arguments are:

(1) lun

name: lun
type: integer longword
access: read only

Logical unit number of the ICF. This is the same logical unit number supplied to Open_Tbl.

(2) chan_type

name: chan_type
type: integer longword
access: read only

Channel type to match. If the value is negative, Tbl_Chan_Dir returns all existing channels without matching channel types.

(3) dir

name: dir
type: longword array
access: write only

Channel directory to return. Each element contains a channel number that exists in the ICF. If a nonnegative chan_type argument is specified, then the channels match that channel type.

(4) dir_size

name: dir_size
type: integer longword
access: read only

Size of channel directory array. An ICF can contain as many as 512 channels, so this argument should be at least 512. Tbl_Chan_Dir does not return an error if the directory is too small to hold all the channels; it only returns as many channels as fit in the directory.

(5) num_chans

name: num_chans
type: integer longword
access: write only

Number of channels Tbl_Chan_Dir returned in dir. This number will never be more than dir_size.

B.4.5 Close_Tbl

Close_Tbl is an integer*4 function that closes the ICF. Close_Tbl returns as the function value a status code that indicates success or failure. Close_Tbl and the variable to receive the return status must be declared in the calling program as integer*4 variables.

The calling format for Close_Tbl is:

```
status = Close_Tbl (lun)
```

where the argument is:

(1) lun

name: lun
type: integer longword
access: read only

Logical unit number of the ICF. This is the same logical unit number supplied to Open_Tbl.

B.4.6 Add_Tbl_Rec

Add_Tbl_Rec is an integer*4 function that is used to add a record to an existing ICF. It is used only by a routine that is building an ICF.

The calling format would be:

```
status = Add_Tbl_Rec    (lun,
                         rec_name,
                         rec_index,
                         sub_rec,
                         byte_array)
```

where the arguments are:

(1) lun

name: lun
type: integer longword
access: read only

Logical unit number of the ICF. This is the same logical unit number supplied to Open_Tbl.

(2) rec_name

name: rec_name
type: character string
access: read only

Name of the record (or structure) type to write. This name must be the name of a structure definition in TABLE_STRUCTS.DEF or Add_Tbl_Rec will return an error code.

(3) rec_index

name: rec_index
type: integer longword
access: read only

Record index to use. If there are multiple occurrences of a certain record type in an ICF, Add_Tbl_Rec uses rec_index to identify the desired record. If there can only be one occurrence of the record in the ICF, rec_index is ignored.

(4) sub_rec

name: sub_rec
type: integer longword
access: read only

Subchannel record number. This argument is applicable to SANDUS DIGITAL and ANALOG MUX channels, and to Tektronix 7912, and 7103, and the RTD 720 channels. It does NOT apply to SANDUS ANALOG channels. For all other records, this argument should be zero.

(5) byte_array

name: byte_array
type: byte array
access: read only

This is the record to be entered into the ICF. Note that it is a byte array.

B.4.7 Delete_Tbl_Rec

Delete_Tbl_Rec is an integer*4 function that deletes an existing record from an ICF. It would only be used by a routine building an ICF. To change an existing record, use Modify_Tbl_Rec.

The call to Delete_Tbl_Rec is:

```
status = Delete_Tbl_Rec ( lun,
                          rec_name,
                          rec_index,
                          sub_rec)
```

where the arguments are as follows:

(1) lun

name: lun
type: integer longword
access: read only

Logical unit number of the ICF. This is the same logical unit number supplied to Open_Tbl.

(2) rec_name

name: rec_name
type: character string
access: read only

Name of the record (or structure) type to write. This name must be the name of a structure definition in TABLE_STRUCTS.DEF or Delete_Tbl_Rec will return an error code.

(3) rec_index

name: rec_index
type: integer longword
access: read only

Record index to use. If there are multiple occurrences of a certain record type in an ICF, Delete_Tbl_Rec uses rec_index to identify the desired record. If there can only be one occurrence of the record in the ICF, rec_index is ignored.

(4) sub_rec

name: sub_rec
type: integer longword
access: read only

Subchannel record number. This argument is applicable to SANDUS DIGITAL and ANALOG MUX channels, and to Tektronix 7912, and 7103, and the RTD 720 channels. It does NOT apply to SANDUS ANALOG channels. For all other records, this argument should be zero.

B.4.8 Get_Tbl_Info

Get_Tbl_Info is an integer*4 function that returns information about the fields and records in an ICF. All of the information except the record key is stored in the ICF itself, in a record described in STRUCT_DESC.

The call to Get_Tbl_Info is:

```
status = Get_Tbl_Info   ( rec_name,
                          rec_index,
                          sub_rec,
                          rec_key,
                          rec_length)
```

where the arguments are:

(1) rec_name

name: rec_name
type: character string
access: read only

Name of the record (or structure) type to write. This name must be the name of a structure definition in TABLE_STRUCTS.DEF or Get_Tbl_Info will return an error code.

(2) rec_index

name: rec_index
type: integer longword
access: read only

Record index to use. If there are multiple occurrences of a certain record type in an ICF, Get_Tbl_Info uses rec_index to identify the desired record. If there can only be one occurrence of the record in the ICF, rec_index is ignored.

(3) sub_rec

name: sub_rec
type: integer longword
access: read only

Subchannel record number. This argument is applicable to SANDUS DIGITAL and ANALOG MUX channels, and to Tektronix 7912, and 7103, and the RTD 720 channels. It does NOT apply to SANDUS ANALOG channels. For all other records this argument should be zero.

(4) rec_key

name: rec_key
type: integer longword
access: write

This is the RMS record key. The structure of the key is defined in Get_Tbl_Info and is similar to the first four bytes of each record.

(5) rec_length

name: rec_length
type: integer longword
access: write

This is the record length in bytes.

B.4.9 CREATE_TBL.FOR

Create_Tbl is a stand alone utility used to create an empty ICF named WORK.TBL. The source file is in TOOLS$LIBRARY:CREATE_TBL.FOR, where the symbol TOOLS$LIBRARY is defined:

$ DEFINE TOOLS$LIBRARY LD:[TOOLS] (or wherever the file is maintained)

WORK.TBL is either renamed as required, or copied to the desired directory with a new file name. It must be populated with the necessary records before being useful, but it does contain record lengths and record names in its only record STRUCT_DESC.

CREATE_TBL must be run, as the OPEN_TBL routine verifies that the named ICF exists. An OPEN_TBL call is executed by the INGRES database administrator. CREATE_TBL.FOR must be recompiled when any change is made to TABLE_STRUCTS.DEF. The structure

STRUCT_DESC contains an upper bound on the number of unique structures that are currently allowed in TABLE_STRUCTS. This is currently 40, and is easily changed. CREATE_TBL.COM exists, and may be used to compile and link CREATE_TBL.EXE.

B.4.10 ANALYZE_STRUCTS.FOR

CREATE_TBL.FOR contains a subroutine ANALYZE_STRUCTS that has been modified and used to prepare the DUMP....FOR subroutines in REPAIR$LIBRARY. The program is called DUMPREP, and is documented in Appendix I. This subroutine has also been modified to create a program that will take a CHN file from test A, modify the header, and create a CHN file for test B.

B.5 IMPLEMENTATION

As mentioned above, all of the ICF utility routines return status values to the calling program that indicate the success or failure of the routines. The status codes are defined in such a way that success codes are odd values and failure codes are even values. In VAX FORTRAN numeric values can be tested as logical expressions; odd values test TRUE and even values test FALSE. The return status should always be tested and appropriate actions should be taken. If the routine fails, the user should at least write out an error message. There is a routine named Bad_Status that will convert a status code to an ASCII error message and print it out on the terminal. It can be called as a subroutine with three arguments. The arguments are:

(1) calling_name

name: calling_name
type: character string
access: read only

Name of the calling routine. This name gets printed out in order to show which routine the program was in when it tried to call the ICF utility routine.

(2) called_name

name: called_name
type: character string
access: read only

Name of the called routine. This name gets printed out in order to show which utility routine returned the error status.

(3) status

name: status
type: integer longword
access: read only

Status code returned by the called routine. The status code is converted to a system-supplied error message and the error message is printed out.

A typical example is:

```
program Main
integer*4 Open_Tbl, status
      .
      .
      .
status = Open_Tbl ( ... )
if (status) then        ! Success
      .
      .
      .
else                    ! Failure
   call Bad_Status ('Main', 'Open_Tbl', status)
      .
      .
      .
end if
```

Bad_Status and the ICF routines (along with other useful routines) are in the object library UTILITY.OLB, or its companion for use while debugging, UTILITY.DBG_OLB. Before you can run a program that calls these routines, you must link them with your program. To do that, use the following command (or your own variation):

$ DEFINE UTILITY$LIBRARY LD:[UTILITY] (or "directory containing UTILITY.OLB")

and either

$ LINK YOUR_PROG, UTILITY$LIBRARY:UTILITY/LIBRARY

or

$LINK/DEBUG PROG,UTILITY$LIBRARY:UTILITY.DBG_OLB

APPENDIX C

BIG FILE FORMAT

APPENDIX C
BIG FILE FORMAT

C.1 INTRODUCTION

This document describes a BIG file. BIG File is a generic phrase describing file types that contain data from one or more channels, that are created by FETCH, DECOM, or REALIZE, and that are processed by either PROCESS or ANALYZE into CHN files. Certain other software tools, such as AUTOSIM, RTDTEST, and DHTEST, create BIG files.

BIG files contain information about their channels, and contain data from a specific recording device, i.e., ACE data channels, or data of a specific type, i.e., realtime data. The BIG file is in a very cryptic format and is best used with a suite of utility routines that handle all aspects of the BIG files creation and use.

C.1.1 BIG File Creation by FETCH/DECOM and REALIZE

The software programs FETCH and DECOM, among other tasks, pull the data from a collection point, demodulate it as required, and pack the data, along with records from the Instrument Control File (ICF), into a BIG file. Documentation for FETCH and DECOM exists, but not in this Manual.

REALIZE is a program created for the RTD 720 device that collects data from one or more RTDs, prefixes the data along with information from the ICF, and packages the data into a BIG file. For additional information on REALIZE, see Section 4 of the Operators Manual, and Section 2 of the Maintenance Manual.

C.1.2 BIG File Use by PROCESS and ANALYZE

The program PROCESS processes the BIG files created by FETCH/DECOM into CHN files. ANALYZE serves the same function for BIG files created by REALIZE. The BIG file, with calibration information from the CHNCAL file, contains all the information necessary to create a CHN file. Documentation for PROCESS exists, but not in this manual. For further information on ANALYZE see Section 5 of the Operator's Manual and Section 4 of the Maintenance Manual.

C.1.3 Working with BIG Files

Because of the very specialized nature of BIG files, there is a special set of utility routines that all programs use. Appendix E describes this set of utility routines. These utility routines are:

(a) OPEN_BIG - opens an existing file or creates a new file.
(b) BIG_DIR - returns a directory of the channels in a file.
(c) READ_BIG - reads channel data from a file.
(d) READ_BIG_HDR - retrieves a channel header from a file.
(e) WRITE_BIG - writes channel data to a file.
(f) CLOSE_BIG - closes a file.

All of the above routines are implemented as integer functions that return status values to a longword (integer*4) in the calling program. The status values indicate whether the functions succeeded or failed.

C.1.4 File Format

The BIG data files do not use standard VMS Record Management Services (RMS) file formats or control information. They appear to the operating system to be simply a stream of bytes without any structure or organization. They do have an organization, but it is meaningful only to the BIG routines listed in paragraph C.1.3.

Each BIG file contains a file header, channel headers, and data. The file header appears at the beginning of the file and is followed by the first channel header, then the data for the first channel, then the second channel header and channel data, and so on. Every header and every data section starts on a block boundary. There are no terminators to separate, or control fields to distinguish one channel from another. The only way to tell where a channel begins and ends is with the information stored in the file header.

There may be up to 512 channels in each BIG file. The channel numbers must be in the range 0 to 511, and a channel may only occur once in the file. Each channel may contain up to 2,147,483,646 bytes of information.

C.2 BIG FILE NAMING

BIG file names have the following form:

sss**BIG**tnnx**.DAT**, where:

(a) **sss** - indicates the type of data in the file, where:

(1) BAS - for baseline data (not currently used)
(2) CAL - for calibration data
(3) FST - for data from HDDR "fast" channels
(4) MEM - for Mass Memory channel data
(5) REL - for realtime data
(6) RTD - for RTD720 data
(7) SLW - for data from HDDR "slow" channels
(8) T48 - for Tektronix 7912 early calibrations

The data may be further qualified by the BIG file header variables DATA_TYPE and USE_INDEX.

(b) **tnn** - indicates the specific recording device type and number, i.e., R12 for GPIB bus number 2 on RMV 10, S03 for SANDUS 503, and A21 for ACE 21.

(c) **x** - indicates the stream source (L for live, T for tape, S for source).

BIG File Name Prefixes from a Source and/or a Collection Point

	Data Source			
		Tektronix		
Data Collection Point	SANDUS	7912	7103	RTD720
HDDR	BAS, CAL, FST, SLW, REL	FST	FST	None
MASS MEM	BAS, CAL, MEM, REL	MEM	MEM	None
SOURCE	BAS, CAL, FST, SLW	CAL, T48	None	RTD

C.3 REFERENCES AND SUGGESTED READING

The following three paragraphs list some related documentation, and some very important files which contain source code with a documentation. Users are advised to become familiar with this documentation.

C.3.1 Documentation

(1) REALIZE - creates the RTDBIG file with RTD720 data and is documented in Section 5 of the User's Manual and Section 2 of this Manual.

(2) ANALYZE - reads the RTDBIG file written by REALIZE and creates the individual CHN files used by DSP and PRELEWD. ANALYZE is described in Section 5 of the User's Manual and Section 4 of this Manual.

(3) BIG File Utility Routines are describe in Appendix E of this Manual.

(4) The CHN Data File Format is described in Appendix D of this Manual.

C.3.2 INCLUDE Files

INCLUDE files are so named because they are captured into the source code by the use of the FORTRAN INCLUDE statement. These files are maintained in INCLUDE$INCLUDE, which is defined as:

$ DEFINE INCLUDE$INCLUDE LD:[INCLUDE] (or wherever these files are maintained)

(1) BIG_STRUCTS.DEF

This file defines the structures and fields for BIG files. Each BIG file contains one BIG file header and a channel header for each channel in the file. The header will consist of records extracted from the ICF combined with supplemental information. This file includes TABLE_STRUCTS.DEF and BIG_STRUCTS.PRM.

(2) TABLE_STRUCTS.DEF

This file defines the structures and fields for the ICF, BIG, and CHN file headers. It assumes that each data source will have one control file associated with it which will contain information common to all channels of the source, and will have channel-specific information. This file is included in BIG_STRUCTS.DEF.

(3) BIG_STRUCTS.PRM

This file defines parameters that determine the maximum number of subchannel records in the BIG file. These parameters are named xxx_SUB_MAX, where XX is a source device name, (i.e., T7912 or S_ALOG).

C.3.3 A Brief Explanation of Records and Structures

For a detailed description of the DEC's nonstandard implementation of records and structures, see both the VAX FORTRAN Language Reference Manual Order Number AA-D034E-TE, dated June 1988, and the VAX FORTRAN User Manual Order Number AA-D035E-TE, dated June 1988. Later editions exist.

Those who choose not to delve deeply into records and structures can think of a structure as a description of a collection of variables, a plan, and a record as the structure's realization in memory. The structure is the architect's plan, while the record is the construction company's building; other buildings may be built using the same set of plans. The variables described in the structure may be any legal FORTRAN type in any order. In a record, the variables are stored in structure order with no blank space. A record statement assigns a variable name to a structure. A structure description may contain record statements.

The variable name defined in a record statement can be used in much the same way that usual variable names are used, but the name refers to all the elements contained in that record. The structure defined by a record statement may be dimensioned. An individual element in a record is referenced by prefixing its name with the name of each record it is a member of, working outward. Thus the variable name: "A.B.C(i).X" is element "X" of the ith record "C" in record "B" in record "A". There is a difference between a structure and a record; however, authors tend to use the two words interchangeably.

C.4 BIG FILE HEADER DESCRIPTION RECORD

The file header contains information that describes the file to the utility routines. It currently occupies nine blocks. It is defined by the following structure in the file BIG_STRUCTS.DEF.

```
structure /BIG_FILE_HDR_DESC/
        integer*4 start_block(0:511)
        integer*4 byte_count(0:511)
        character*24 date
        character*16 col_name
        integer*4 use_index
        integer*4 data_type
        integer*4 subcom_depth
        integer*4 frame_length
        integer*4 trig_cnts(6)
        integer*4 %fill(3)
        record /GEN_DESC/ gd
        record /PRI_TYPE_DESC/ pt
        record /S_TRIG_DESC/ st
end structure
```

With the exception of the start_block field, the contents of the file header may be used by the calling program. Several structure elements must be supplied by the program when creating a new file.

(1) START_BLOCK

The start_block field is an array that contains the starting block number for each channel. This field is intended to be used only by the BIG file access routines. A channel header always starts on a block boundary.

(2) BYTE_COUNT

The byte_count field is an array the contains the number of data bytes for each channel. This field will need to be used by the calling program when reading data from the file. A user's program may not change this field.

(3) DATE

The date field specifies the date and time that the file was created. The calling program may read, but not write, this field.

(4) COL_NAME

The col_name field gives the name of the collection point or subsystem that collected the data. The program that creates the data file must supply a value for this field.

(5) USE_INDEX

The use_index field specifies the purpose of the data. Acceptable values are 0, which implies normal gauge data, 1, which implies laser calibration data, and 2, which implies cable compensation data. This field must be supplied by the program creating the BIG file.

(6) DATA_TYPE

The data_type field specifies the type of data that is contained in the file from the following categories: memory data, realtime data, standard or late calibration data, and early calibration data. The program that creates the data file must supply a value for this field.

(7) SUBCOM_DEPTH

The subcom_depth field specifies the maximum subcommutation depth of the data stream format. The program that creates the data file must supply a value for this field.

(8) FRAME_LENGTH

The frame_length field specifies the length, in bytes, of the data stream's major frame. The program that creates the data file must supply a value for this field.

(9) TRIG_CNTS(6)

The trig_cnts field is an array of trigger counts. The program that creates the data file must supply values for this field. It is meaningful only for the SANDUS.

(10) %FILL(3)

The %fill field hasn't been defined yet. %fill reserves space in the structure, but does not create a variable name for that space.

(11) RECORD /GEN_DESC/ GD

The GD record is simply a copy of the GEN_DESC record from the ICF, and has the same structure. It supplies information such as the test name, source code, trigger channels, etc. The GEN_DESC record structure is defined in TABLE_STRUCTS.DEF.

(12) RECORD /PRI_TYPE_DESC/ PT

The PT record is simply a copy of the PRI_TYPE_DESC record from the control file, and has the same structure. It contains values that specify the primary types of all the channels. The PRI_TYPE_DESC record structure is defined in TABLE_STRUCTS.DEF.

This byte array enables the utility program BIG_DIR to return the primary channel type description value of each channel in the BIG file. This value is equal to the record type, which appears in the third byte of all structures defined in TABLE_STRUCTS.DEF. These values, described in the following list, are used in virtually every program in the NTS Instrumentation System, and should not be changed without careful thought. The values are listed in the form Value/Description/Corresponding Structure Name.

(a) 0 = Overhead or Dummy not applicable
(b) 18 = Trigger /TRIGGER_CHAN_DESC/
(c) 19 = SANDUS Analog /S_ALOG_CHAN_DESC/
(d) 20 = SANDUS Digital /S_DIG_CHAN_DESC/
(e) 22 = SANDUS Analog Mux /S_AMUX_CHAN_DESC/
(f) 24 = Tektronix 7912 /T7912_CHAN_DESC/
(g) 28 = Tektronix 7103 /T7103_CHAN_DESC/
(h) 30 = RTD 720 Device /RTD_DEV_DESC/
(i) 31 = RTD 720 Channel /RTD_CHAN_DESC/

(13) RECORD /S_TRIG_DESC/ st

The ST record is simply a copy of the S_TRIG_DESC record from the control file, and has the same structure. It contains values that define triggers for SANDUS operations. The record structure is defined in TABLE_STRUCTS.DEF.

C.5 BIG FILE CHANNEL DESCRIPTION RECORDS

The following structures define the BIG file channel headers. Each channel header consists of a channel description record from the ICF, plus various additional fields, depending on the channel type. There is one channel header for each channel, and it is this record that is returned by the utility routine READ_BIG_HDR.

For all channel headers except the RTD channel header, FETCH/DECOM will derive the value for the COM_RATE field from the stream format file. The COM_RATE field is not meaningful for the RTD. The NUM_RT_CHANS and RT_CHANS fields are only applicable to the SANDUS. They tell PROCESS how many realtime channels are grouped together and what the individual realtime channel numbers are. The order of the realtime channel data within the group is the same as the order of the channel numbers. The TAPE_RATIO is only meaningful for data (hence sources) processed by FETCH.

C.5.1 Current Source Devices

A BIG file may contain data from any one of four types of source device. The short name of the device is shown parenthetically. They are:

(A) SANDUS - which has three possible configurations,
  (a) SANDUS ANALOG (S_ALOG)
  (b) SANDUS DIGITAL (S_DIG), with S_DIG_SUB_MAX data subchannels/ channel
  (c) SANDUS ANALOG MUX (S_AMUX), with S_AMUX_SUB_MAX data subchannels/channel (configuration probably obsolete)

(B) Tektronix 7912 (T7912)

(C) Tektronix 7103, with attached CCD camera (T7103) (obsolete)

(D) RTD 720. The RTD may be configured with one or more channels, which can depend on what is being recorded. A description of its header is more complex.

C.5.2 SANDUS ANALOG Channel Header

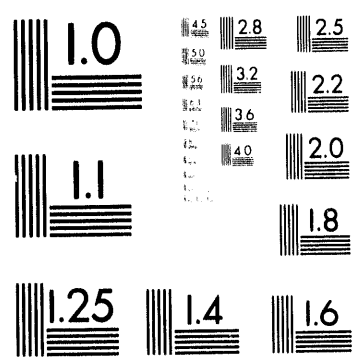
This structure is used for SANDUS ANALOG channels. The variables NUM_RT_CHANS and RT_CHANS are only used if this channel is a realtime channel; they indicate how many channels are grouped with this channel and the channel numbers.

```
structure /S_ALOG_HDR_DESC/
        real*4 com_rate                     ! Commutation rate
        integer*4 num_rt_chans              ! Number of realtime channels
        integer*4 rt_chans(8)               ! Realtime channel numbers in order
        real*4 tape_ratio                   ! Playback speed / record speed
        integer*4 %fill(4)                  ! Unused (to be defined)
        record /S_ALOG_CHAN_DESC/ sa        ! Defined in Table_Structs.def
end structure
```

C.5.3 SANDUS DIGITAL Channel Header

This structure is used for SANDUS DIGITAL channels. There are no realtime digital channels. These variables have the same meaning as before. Note that there are always S_DIG_SUB_MAX sds records in this header, even though they may not all be used.

```
structure /S_DIG_HDR_DESC/
        real*4 com_rate                     ! Commutation rate
        integer*4 num_rt_chans              ! Number of realtime channels
        integer*4 rt_chans(8)               ! Realtime channel numbers
        real*4 tape_ratio                   ! Playback speed / record speed
        integer*4 %fill(4)                  ! Unused (to be defined
        record /S_DIG_CHAN_DESC/ sd         ! Defined in Table_Structs.def
        record /S_DIG_SUB_DESC/ sds(1:S_DIG_SUB_MAX)
                                            ! Defined in Table_Structs.def
                                            ! Parameter in Big_Structs.prm
end structure
```


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0

4 of 5

C.5.4 SANDUS ANALOG MUX Channel Header

This structure is used for SANDUS ANALOG MUX channels. There are no realtime analog mux channels. This particular channel type has not been used often, and may never be used again. The variables have the same meaning as above. Note that there are always S_AMUX_SUB_MAX sms records in this header, even though not all may be used.

```
structure /S_AMUX_HDR_DESC/
        real*4 com_rate                          ! Commutation rate
        integer*4 num_rt_chans                   ! Number of realtime channels
        integer*4 rt_chans(8)                    ! Realtime channel numbers
        real*4 tape_ratio                        ! Playback speed / record speed
        integer*4 %fill(4)                       ! Unused (to be defined)
        record /S_AMUX_CHAN_DESC/ sm             ! Defined in Table_Structs.def
        record /S_AMUX_SUB_DESC/ sms(1:S_AMUX_SUB_MAX)
                                                 ! Defined in Table_Structs.def
                                                 ! Parameter in Big_Structs.prm
end structure
```

C.5.5 T7912 Channel Header

This structure is used for Tektronix 7912 channels. 7912 channels are connected to the ACE recording devices. The variables have the same meaning as before. Note that in contrast to the SANDUS subchannels, there will be not more than T7912_SUB_MAX+1 t7912s records in this header. Only those actually needed will be here.

```
structure /T7912_HDR_DESC/
   real*4 com_rate                               ! Commutation rate
   real*4 tape_ratio                             ! Playback speed / record speed
   integer*4 %fill(4)                            ! Unused (to be defined)
   record /T7912_CHAN_DESC/ t7912
                                                 ! Defined in Table_Structs.def
   record /T7912_SUB_DESC/ t7912s(0:T7912_SUB_MAX)
                                                 ! Defined in Table_Structs.def
                                                 ! Parameter in Big_Structs.prm
end structure
```

C.5.6 T7103 CHANNEL HEADER

This structure is used for Tektronix 7103 channels. These devices will probably never be used again. The variables have the same meaning as before. Note that in contrast to the SANDUS subchannels, there will be not more than t7103_sub_max+1 t7912s records in this header. Only those actually needed will be here.

```
structure /T7103_HDR_DESC/
        real*4 com_rate                          ! Commutation rate
        real*4 tape_ratio                        ! Playback speed / record speed
        integer*4 %fill(4)                       ! Unused (to be defined)
        record /T7103_CHAN_DESC/ t7103
                                                 ! Defined in Table_Structs.def
        record /T7103_SUB_DESC/ t7103s(0:t7103_sub_max)
                                                 ! Defined in Table_Structs.def
                                                 ! Parameter in Big_Structs.prm
end structure
```

C.5.7 RTD720 Channel Header

This structure is used for an RTD720 channel. The RTD720 device can have either one, two, or four channels. Each channel will have a header, which includes the device record; the device record is repeated for each of the device's channels. The appearance of this structure is further complicated by the intent of the data, which is determined by the variable USE_INDEX in the BIG file header. The "UNION" operator acts as an "or" for each map/end map group, either the map/end map group for USE_INDEX =0, or for USE_INDEX=1, or for USE_INDEX=2. Again, note that only ...SUB_DESC+1 rec records actually required are present and that there cannot be more than the number indicated.

```
structure /RTD_HDR_DESC/
        integer*4 %fill(5)              ! Unused (to be defined)
        integer*4 fill                  ! =1 ==> TBL_STREAM_MIS=bad (F)
                                        ! =0 ==> TBL_STREAM_MIS=good (T)
        record /RTD_DEV_DESC/ rtd
                                        ! Defined in Table_Structs.def
        record /RTD_CHAN_DESC/ rtdc
                                        ! Defined in Table_Structs.def
```

```
        union
            map                         ! USE_INDEX = 0
            record /RTD_EXP_SUB_DESC/ rtdes(0:RTD_EXP_SUB_MAX)
                                        ! Parameter in Big_Structs.prm
            end map                     ! Defined in Table_Structs.def

            map                         ! USE_INDEX = 1
            record /RTD_LC_SUB_DESC/ rtdls(0:RTD_LC_SUB_MAX)
                                        ! Parameter in Big_Structs.prm
            end map                     ! Defined in Table_Structs.def

            map                         ! USE_INDEX = 2
            record /RTD_CC_SUB_DESC/ rtdcs(0:RTD_CC_SUB_MAX)
                                        ! Parameter in Big_Structs.prm
            end map                     ! Defined in Table_Structs.def

        end union
    end structure
```

C.6 DATA DESCRIPTION

The data format of the RTDBIG file is quite different than the original format developed for data streams that are processed by FETCH/DECOM (SANDUS, Tektronix 7912, and 7103).

C.6.1 SANDUS/T7912 Data Description

Each Big file data word is a two byte integer. Each 16 bit word consists of an op-code (bits 12 to 15) and a data value (bits 0 to 11). The first ten words contain information to assist in data processing. The actual data starts with the eleventh word.

(a) Op-code 15 (octal) denoting start of data, with the data portion of the word containing the channel number. (1st word)

(b) Op-code 11 (octal) word containing error messages. (2nd word)

(c) Three sync words (3rd through 5th words)

(d) Five status words containing set-up parameters applicable to the run. (6th through 10th words). These can be translated to ICF entries by the CONVERSION routines. They are not documented here.

After the final data value for a channel, an op-code 14 data word is inserted to denote end of channel data. The op-code 11 error messages are also repeated at the end of channel data. Various other op-codes may be includes within the channel data stream to indicate one of the following:

(a) Loss of data - generally only needed for realtime data and by the 7912. This is different from data at one of the band edges. It shows as a message on the plot and as a break in the graph line.

(b) Data compression - hardware data compression only; all software data compression is done in PROCESS. See the discussion of data compression in Appendix D.

(c) Loss of context - FETCH and DECOM must decommutate the data that is sent out in the stream; FETCH uses hardware and DECOM uses software. For twelve bit data, DECOM must piece together two consecutive bytes. If a byte is missing or an extra byte has been inserted, DECOM recognizes the omission or addition, but it will not repair the data. This, very briefly, is a context error.

The content and interpretation of the data depends both on the source of the data and the type of file. T48 files only occur for Tektronix 7912 sources; in the SANDUS family, CAL files occur only for the SANDUS ANALOG sources. CAL files also occur for the Tektronix 7912. These are described in Appendix D.

For a complete description of all SANDUS op-codes and their information content, please refer to **SANDUS (MA164) COMMAND SYSTEM MESSAGE AND DATA FORMAT**, SAND86-1398, UC-37 (Ebinger, 1988).

C.6.2 Tektronix 7103 Data Description

This digitizer is obsolete, and will be deleted.

C.6.3 RTD720 Data Description

This Big file is in byte format. Each 8 bit data value retrieved from the RTD720 occupies a byte of storage. The ANALYZE program moves this data into an integer*2 array in CHN files for input to the PRELEWD plotting program. The length of the data record is a function of the device setup values.

APPENDIX D

CHANNEL DATA FILE FORMAT

APPENDIX D
CHANNEL DATA FILE FORMAT

D.1 INTRODUCTION

This document describes a CHN file. CHN File is a generic phrase describing the three types of channel data files created from a BIG file by either PROCESS or ANALYZE. It is a stand-alone file that, when used with PRELEWD or DSP, contains sufficient information to generate a fully annotated time/answer plot.

D.2 CHN FILE NAMING

A CHN file name is inherited, in part, from the BIG file name that fathered it. Hunters Trophy BIG files are described in Appendix C. CHN file names have the following form:

nnnxxxyzzz-ww.DAT, where:

nnn is a prefix describing the type of data in the file; it is one of the three strings:

(a) *CHN*, which implies dry run or event data,
(b) *CAL*, which implies dry run or event calibrations, or
(c) *T48*, which implies early calibrations (only for a Tektronix 7912).

xxx is a source device code, e.g., A32 for ACE 32, S04 for SANDUS 504, or R12 for GPIB bus number 2 on RMV 10. Source means the primary recording station.

y is a collection device code, e.g., M for Mass Memory, L for HDDR (live), and S for Source. Collection device is that piece of hardware where the data is stored for further processing. An RTD is both a source and collection device. Many of the modern digitizers do both tasks.

zzz is the channel number, e.g., 000, or 177. For the SANDUS and ACE, channel numbers are in octal. For the RTD, channel numbers are decimal.

-ww identifies the subchannel; -00 is for the entire channel. A subchannel may be all or part of a channel. A subchannel is always used, and only meaningful, for non-SANDUS devices.

The sequence *xxxyzzz-ww* is the historical source-collection-channel number sequence by which the provenance of the data is known. The source and collection codes and channel numbers vary for each event, e.g., S03M001, A32T011-00, A61L032-03, or R11S012-01.

D.3 CHN FILE DEVELOPMENT

D.3.1 REALIZE/FETCH/DECOM

REALIZE, a program created for the RTD 720 devices, collects data from one or more RTDs and packages the data into a BIG file. For a detailed description of REALIZE, see Section 4 of the Operators Manual, and Section 2 of this manual.

The software programs FETCH and DECOM, among other tasks, pull the data from a collection point, demodulate it as required, and pack the channel data and information from the Instrument Control File (ICF) into a BIG file. FETCH and DECOM are documented elsewhere and are not necessary to understanding the CHN file.

D.3.2 ANALYZE/PROCESS

The program ANALYZE processes the BIG files created by REALIZE into CHN files, prefixing the data with records both from the BIG file and from ANALYZE. PROCESS serves the same function, but for BIG files created by either FETCH or DECOM. In either case, the resulting CHN file contains all the information necessary to decode the data into a signal versus time format. For further information on ANALYZE, see Section 4 of this manual, and Section 5 of the Operator's Manual. PROCESS is documented elsewhere and is not necessary to this discussion.

D.4 EXPLAINING RECORDS AND STRUCTURES

For a detailed description of the DEC's nonstandard implementation of records and structures, see both the VAX FORTRAN Language Reference Manual Order Number AA-D034E-TE and the VAX FORTRAN User Manual Order Number AA-D035E-TE, both dated June 1988. Later editions exist.

Those who choose not to delve deeply into records and structures can think of a structure as a description of a collection of variables, a plan, and a record as the structure's realization in

memory. The structure is the architect's plan, while the record is the construction company's building; other buildings may be built using the same set of plans. The variables described in the structure may be any legal FORTRAN type in any order. In a record, the variables are stored in structure order with no blank space. A record statement assigns a variable name to a structure. A structure description may contain record statements.

The variable name defined in a record statement can be used in much the same way that usual variable names are used, but the name refers to all the elements contained in that record. The structure defined by a record statement may be dimensioned. An individual element in a record is referenced by prefixing its name with the name of each record it is a member of, working outward. Thus the variable name *A.B.C(i).X* is element *X* of the *i*th record *C* in record *B* in record *A*. There is a difference between a structure and a record; however, authors tend to use the two words interchangeably.

D.4.1 CHN HEADER Structure

The following is taken from CHN_HEADER.DEF; it is the structure that describes the CHN file header. The user who examines and understands this structure will have no problem with any other structure. The lines have been numbered to facilitate explanation.

```
01    structure / CHN_FILE_HDR_DESC /
02       union
03            map
04               record / GENERAL_CHAN / gc
05               record / GEN_DESC / g
06               union
07                    map
08                       record / S_ALOG_CHAN_DESC / sa
09                       record / CAL_S_ALOG / sa_cal
10                    end map
11                    map
12                       record / S_DIG_CHAN_DESC / sd
13                       record / S_DIG_SUB_DESC / sds(1:S_DIG_SUB_MAX)
14                       record / CAL_S_DIG / sd_cal
15                    end map
16                    map
17                       record / S_AMUX_CHAN_DESC / sm
18                       record / S_AMUX_SUB_DESC /  sms(1:S_AMUX_SUB_MAX)
19                       record / CAL_S_AMUX / sm_cal
20                    end map
21                    map
22                       record / T7912_CHAN_DESC / t
23                       record / T7912_SUB_DESC / ts
24                       record / CAL_7912 / t_cal
25                    end map
26                    map
```

```
27                         record / T7103_CHAN_DESC / c
28                         record / T7103_SUB_DESC / cs
29                         record / CAL_T7103 / c_cal
30                      end map
31                      map
32                         record / RTD_DEV_DESC / r
33                         record / RTD_CHAN_DESC / rc
34                         union
35                            map
36                              record / RTD_EXP_SUB_DESC / res
37                            end map
38                            map
39                              record / RTD_LC_SUB_DESC / rls
40                            end map
41                            map
42                              record / RTD_CC_SUB_DESC / rcs
43                            end map
44                         end union
45                         record / CAL_RTD / r_cal
46                      end map
47                end union
48             end map
49             map
50                byte Z(512,6)
51             end map
52       end union
53    end structure
```

Note that this structure has both variables (Z, line 50) and records as elements. Some of the records are dimensioned (lines 13 and 18). The parameters S_DIG_SUB_MAX and S_AMUX_SUB_MAX give the upper bounds of these dimensions, found in BIG_STRUCTS.PRM. For every STRUCTURE, there is an END STRUCTURE; for every UNION, an END UNION; and for every MAP, an END MAP.

Starting at the right-most indentation levels, the UNION/END UNION of lines 34-44 includes three MAP/END MAP groups, lines 35-37, 38-40, and 41-43. A UNION/END UNION is similar to a FORTRAN equivalence statement, and the MAP/END MAP groups define the elements that are equivalent. The records named RES, RLS, and RCS will start at the same location; the longest will define the space taken by this UNION/END UNION group.

Now consider the UNION/END UNION group, lines 6-47. It has the 6 MAP/END MAP groups, lines 7-10, 11-15, 16-20, 21-25, 26-30, and 31-46. Each of these 6 groups is specific to a specific source device, and each defines the layout of the CHN file header for that source device. Lines 31-46 define the RTD720 specific part of the header.

Now look at the UNION/END UNION group, lines 2-52. It covers two MAP/END MAP

groups, lines 3-48, and 49-51. Lines 4 and 5 define the two records that are PHYSICALLY and LOGICALLY first in the header (and in ALL headers). The MAP/END MAP group at line 49-51 defines a byte variable Z dimensioned (512,6). Z and the MAP/END MAP group, lines 3-48, occupy the same address space. 6 is the length in physical records of the longest device header and 512 is the byte length of a physical record. This construction allows the user to read the first physical record from the file, using an ordinary binary file read, and then to use the variable CHN.GC.LEN_HEADER in the record GC to determine how many more physical records need to be read from this CHN file. See module HEADER_READ for details.

The STRUCTURE/END STRUCTURE statements, lines 1-53, complete the structure definition. CHNHEAD.CMN declares this structure to be a record named CHN. There may be several record statements, each declaring this structure to have a locally unique name.

The structure definition files TABLE_STRUCTS.DEF, CHN_HEADER.DEF, and the parameter file BIG_STRUCTS.PRM are all in INCLUDE$INCLUDE, where that symbol is defined:

> $ DEFINE INCLUDE$INCLUDE LD:[INCLUDE] (or wherever the files are maintained)

D.5 CHN FILE FORMAT

All CHN files are similar in format. They are unformatted direct access files where each of the file's physical records is exactly 512 bytes (one VAX/VMS page) long. They consist of two parts, a header and data, each part beginning a new physical record. A vocabulary problem exists here because of the necessity to use the word "record" both in connection with structures and in connection with files. It should be clear from the context which meaning is intended, however; "record" without a nearby "file" or "physical" always ties to structures.

D.6 CHN FILE HEADER

D.6.1 Type of Information Contained

The header is one record consisting of several physical records; it contains the following three distinct types of information:

(1) General file information about the test, the source, and about the entire file.

(2) Source-specific information about this device and channel from the ICF.

(3) Source-specific calibration information gathered by PROCESS and used to calibrate the raw data. ANALYZE does not have this information available, but passes along a dummy structure.

D.6.2 Current Source Devices

There are currently four types of source devices:

(1) SANDUS, which has three possible configurations:

(a) SANDUS ANALOG (S_ALOG)

(b) SANDUS DIGITAL (S_DIG), with S_DIG_SUB_MAX data channels/channel

(c) SANDUS ANALOG MUX (S_AMUX), with S_AMUX_SUB_MAX data channels/channel (configuration probably obsolete)

(2) Tektronix 7912 (T7912)

(3) Tektronix 7103, with attached CCD camera (T7103) (probably obsolete)

(4) RTD 720. The RTD may be configured with one or more channels, depending on what is being recorded. A description of its header is more complex.

D.6.3 Header Structures Table

This is an expansion of the terse header structure described in paragraph D.6.1.

The Names of Structures in the Header for SANDUS

	SOURCE DEVICE		
Device	S_ALOG	S_DIG	S_AMUX
General (same for all)	----------	GENERAL_CHAN* GEN_DESC**	----------
Source Specific	S_ALOG_CHAN_DESC**	S_DIG_CHAN_DESC** S_DIG_SUB_DESC(n)**	S_AMUX_CHAN_DESC** S_AMUX_SUB_DESC(n)**
Calibration Source Specific	CAL_S_ALOG*	CAL_S_DIG*	CAL_S_AMUX*

The Names of Structures in the Header for Other Sources

	SOURCE DEVICE		
Device	T7912	T7103	RTD 720
General (same for all)	----------	GENERAL_CHAN* GEN_DESC**	---------- ----------
Source Specific	T7912_CHAN_DESC** T7912_SUB_DESC**	T7103_CHAN_DESC** T7103_SUB_DESC**	RTD_DEV_DESC** RTD_CHAN_DESC** and one of the following: RTD_EXP_SUB_DESC** RTD_LC_SUB_DESC** RTD_CC_SUB_DESC**
Calibration Source Specific	CAL_7912*	CAL_T7103*	CAL_RTD*

NOTE

The first two records in the header are common to all source devices; they supply the length of the header in physical records. This allows for a bootstrap file reading technique. These records supply the source device type, because the rest of the header is source-specific.

* For the current definition of this structure, see the ASCII file: "CHN_HEADER.DEF".

** For the current definition of this structure, see the ASCII file: "TABLE_STRUCTS.DEF".

D.6.4 A Short Description of the Structures

The files containing these structures are heavily documented and should be understood before any changes are made. A very brief description of the contents of each record follows:

(1) GENERAL_CHAN

This record is generated by PROCESS or ANALYZE; it contains necessary information not otherwise available from any record in the header, such as error flags, and information captured by PROCESS or ANALYZE. The error flags are logical variables, and if an error occurs, it is noted on the first title line of the plot. It is identical for all sources.

(2) GEN_DESC

This record comes from the ICF via the BIG file; it contains the source name and code, the test name, and the reference trigger number. It is identical for all sources.

(3) RTD_DEV_DESC

This record comes from the BIG file and is specific to the RTD 720. It contains the RTD device setup values, such as the VMODE command and the acquire commands. The original comes from the ICF.

(4) xxxx_CHAN_DESC

This record comes with the BIG file and contains values for the source equipment. It includes instrument settings, calibration information, plot specifications, and channel specific data. The original comes from the ICF.

(5) S_xxxx_SUB_DESC

This record ONLY occurs for SANDUS DIGITAL and ANALOG MUX channels, with one structure for every possible subchannel, whether or not they are used. It contains data that is specific to the subchannel. If this channel is a DIG channel, then all S_DIG_SUB_MAX records are present, whether used or not. If this channel is a MUX channel, then all S_AMUX_SUB_MAX records are present, whether used or not. These parameters are defined in BIG_STRUCTS.PRM

(6) xxxx_SUB_DESC

This record describes the subchannel information, such as where in the data the subchannel record lies. It is specific to the 7912 and 7103.

(7) RTD_EXP_SUB_DESC

This record holds dry run and experiment information about the specific subchannel, i.e., plotting information.

(8) RTD_LC_SUB_DESC

This record holds laser calibration information about the specific subchannel, i.e., plotting information.

(9) RTD_CC_SUB_DESC

This record holds cable compensation information about the specific subchannel, i.e., plotting information.

(10) CAL_xxxx

This record is created by PROCESS working with CAL BIG file; it is passed on in the header of all CHN files. It contains source-specific flags and calibration information. ANALYZE creates this record for the RTD720s, but it is not meaningful, and exists only for future expansion.

D.7 DATA DESCRIPTION

The data consists of 1 or more physical records, each containing 256 sixteen bit words. A sixteen bit word is further broken into an op-code, the high order 4 bits, and the data, the remaining low order twelve bits, which depend on the op-code for interpretation. If the low order twelve bits are meaningful, they are assumed to be a positive integer; thus, the largest integer that can be sent is (2**12)-1, or 4095. If the data can have negative values, a bias is added to all data values so that all values are positive, and that bias appears in the CAL_xxx record; the bias must be subtracted from all data before use. Currently, the Tektronix 7103 is the only device with a data offset. Data values are bounded between 0 and 4095 inclusive.

D.7.1 Op-Codes

There are currently four meaningful op-codes. They are:

(1) '00' octal ('0' hex) signals ordinary data.

This op-code indicates that the low order twelve bits contain a nonnegative twelve bit integer data value.

(2) '07' octal ('7' hex) signals compression repeat factor.

This op-code indicates that the low order twelve bits are a repeat factor. To understand "repeat factor," it is necessary to understand data compression. There can be both hardware and software data compression, which PROCESS incorporates into this op-code. REALIZE/ANALYZE does NOT use any kind of data compression.

In an effort to make the channel files as small as possible, PROCESS puts the last data value into reg A and compares each following data value (in reg B) with A until abs(A-B) .ge. compression factor, while counting the number of values put into B, and including A. This count becomes the "repeat factor" in the low order twelve bits. Both the hardware and software compression factors are in the header record. The compression factor is normally 1. Compression factors greater than 1 are possible, but very unusual. Data compression occurs only on SANDUS signals.

Consider the following string of data values:

... 24 23 23 23 23 25 25 26 27 30 ... becomes

... 24 23 '7'4 25 '7'2 26 27 30 ... (compression factor=1), or

... 24 '7'7 26 '7'2 30 ... (compression factor=2).

A compression factor of 1 will not lose any information. It is not meaningful for the first word of data to have a '7' op-code. Because of the twelve bit limitation on the magnitude of repeat factors, several consecutive '7' op-codes may occur. If op-code '7' is repeated, then the repeat factors are summed to obtain the correct repeat factor. It is meaningful for the last data word to have a '7' op-code. A repeat factor of 0 is not valid.

(3) '11' octal ('B' hex) signals end-of-data.

This op-code indicates that the previous word was the last valid word. The first word of data should not have an op-code 'B'. If the data ends on a block boundary requiring an additional block be written just to signal end-of-data, the end-of-data is omitted. This is specially true for 7912s, which always write 512 pieces of uncompressed data. The low order twelve bits are not used.

(4) '16' octal ('E' hex) signals loss of data.

This op-code indicates that the source instrument did not have data and the low order twelve bits may contain a repeat factor indicating the total number of missing data values. Here, a repeat factor of 0 has the same meaning as a repeat factor of 1. Loss of data is different from a 0 or full scale value, both of which are legal values. It is common in 7912 data. The first data word may have an 'E' op-code. If op-codes are repeated, then the repeat factors are summed to obtain the correct repeat factor.

D.7.2 Data by Device and File Type

The content and interpretation of the data depends on both the source of the data and the type of file. T48 files only occur for Tektronix 7912 sources; in the SANDUS family CAL files occur only for the SANDUS ANALOG sources. CAL files also occur for the Tektronix 7912. The data always starts with the first word of the first physical record following the header record. The header contains a count of the number of words written to the file, as well as the number of words of data before compression.

The Sandus Family

Currently the bulk of the data recorded during any event is from a SANDUS ANALOG source. The SANDUS has a number of documents characterizing it. A large number of variables are necessary to properly interpret the data; they are all contained in the record header. The number of data values is a function of the channel and is generally either 8K (K=1024) or 16K; however, real time channels may have thousands of physical blocks of data at 256 samples/block.

(a) SANDUS ANALOG source (S_ALOG)

(1) CHN file type

S_ALOG data is always compressed, usually with a compression factor of 1. There is only one waveform in the data part of the file.

(2) CAL file type

Calibration data consists of four DC levels, each level beginning a new physical record. The header contains the number of the physical record where each level's data starts. An integer array in the header establishes a correspondence between the levels as they occur in the file, and in the header.

(b) SANDUS DIGITAL source (S_DIG)

Data from this source device is often called binary data. Though the header variables allow for data widths of more that 1 bit, this option has never been used. In this document, the data is referred to as binary data.

(1) CHN file type

Twelve bits are available for data, of which S_DIG_SUB_MAX are used. The bit position and data width (1 bit) for each subchannel is available from the record header. Data compression can occur. Only one waveform is in this file type. There is no correspondence between the sub-channel number and the bit position in the data word. That information is called out in the subchannel record.

(2) CAL file type

A CAL file does not exist for this SANDUS configuration.

(c) SANDUS ANALOG MUX source (S_AMUX)

This source configuration resembles a SANDUS ANALOG source, except that it may have up to 32 subchannels multiplexed together. The number of subchannels is obtained from the header record. Because this source has not been used for several tests, the current maximum number of subchannels is 1, and is

defined by the parameter S_AMUX_SUB_MAX. PROCESS demultiplexes the channel into subchannels, writing the data for the subchannels in a serial manner, each starting a new physical record. The record header contains both the number of samples in each subchannel and the number of the physical record where the data for the subchannel starts.

(1) CHN file type

The header record supplies the number of subchannels in this "channel." There are a maximum of S_AMUX_SUB_MAX waveforms in this file type.

(2) CAL file type

A CAL file does not exist for this SANDUS configuration.

Tektronix 7912 (T7912)

A 7912 waveform consists of 512 upper trace Y values in two physical records, followed by 512 lower trace Y values in two physical records. The upper Y trace and the lower Y trace each start a physical record. An end-of-data op-code will not appear in 7912 CHN files. 7912 data is never compressed.

(a) CHN file type

There is only one waveform in this file type. The CHN file from a 7912 is always two blocks longer than expected, because of a readout of the instrument, which is passed along with the data. If serious questions about the instrument setup arise, this is the final authority. These extra two blocks do not appear on T48 or CAL files.

(b) CAL file type

The CAL file type contains the following eight waveforms in the order given (AUTOCAL creates these files):

(1) 1 baseline for "baseline" sine wave,

(2) 1 "baseline" sine wave,

(3) three DC levels,

(4) two pulses, and

(5) 1 experiment system baseline.

These eight waveforms are a subset of the set of 17 waveforms done for the T48 file type (also created by AUTOCAL).

(c) T48 file type

This file type contains the following 17 calibration waveforms, recorded about 48 hours prior to the event, in the order given here.

(1) 1 baseline for "midscreen" sine wave,

(2) 1 "midscreen" sine wave,

(3) 1 baseline for "baseline" sine wave,

(4) 1 "baseline" sine wave,

(5) eight DC levels,

(6) four pulses, and

(7) 1 experiment system baseline.

Tektronix 7103 source (T7103)

This is a new source type. The number of data points/waveform is 10240, where the first, and last, 64 data points are not used. The data is not compressed and, because there are exactly 40 blocks of data/waveform, there is no end-of-data op-code.

(a) CHN file type

There is one waveform/file.

(b) CAL file type

A CAL file for this device does not exist.

RTD 720 source type

This device is one of the newer digitizers; it has a great many programmable features. The number of samples can vary between 512 and 512K, depending on how the device and channel are configured. The data is not compressed. It generally does not contain missing data.

(a) CHN file type

The header contains one waveform, which may consist of one or more segments, with each segment having its own trigger. The subchannel records must direct ANALYZE to the correct segment.

(b) CAL file type

Currently a CAL file for this device does not exist.

APPENDIX E

BIG FILE UTILITY ROUTINES

APPENDIX E
BIG FILE UTILITY ROUTINES

E.1 INTRODUCTION

The data acquisition and processing operations performed on SANDUS, ACE, and RTD channels involve fetching raw data from hardware devices and writing it out to data files, then reading the files and processing the data into a graphical form. The raw data files are known as BIG files because they can contain data for many channels.[1]

There is a set of utility routines available that are general enough to be used by any program that needs to work with the files. They are:

(1) **OPEN_BIG** - opens an existing file or creates a new file.
(2) **BIG_DIR** - returns a directory of channel numbers from a file.
(3) **READ_BIG** - reads data for a particular channel from a file.
(4) **READ_BIG_HDR** - retrieves a channel header from a file.
(5) **WRITE_BIG** - writes data for a particular channel to a file.
(6) **CLOSE_BIG** - closes a file.

All of the routines are implemented as functions that return status values to a longword (integer*4) in the calling program. The status values indicate whether the functions succeeded or failed. For more information on return status values, see the Implementation section, E.4.

The headers used by the BIG files, along with some of the arguments for these routines, use FORTRAN record structures. For a detailed description of the DEC's nonstandard implementation structures, see both the VAX FORTRAN Language Reference Manual Order Number AA-D034E-TE and the VAX FORTRAN User Manual Order Number AA-D035E-TE, both dated June 1988. Later editions exist.

---

[1] Much of this document and all of the routines described herein were written by Jon Anspach when he worked for Organization 9321. These routines have been modified when necessary and documentation updated by Peter Kaestner of Department 9321. The files that were called tables are now Instrument Control Files (ICFs).

E.2 FILE ORGANIZATION

The BIG files do not use standard RMS file formats or control information. They appear to the operating system to be simply a stream of bytes without any structure or organization. They do have an organization, but it is meaningful only to the routines described in this document.

Each BIG file contains a file header, channel headers, and data. The file header appears at the beginning of the file and is followed by the first channel header, then the data for the first channel, then the second channel header and channel data, and so on. Every header and every data section starts on a block boundary. There are no terminators to separate, or control fields to distinguish one channel from another. The only way to tell where a channel begins and ends is with the information stored in the file header.

There may be up to 512 channels in each BIG file. The channel numbers must be in the range 0 to 511, and a channel may only occur once in the file. Each channel may contain up to 2,147,483,646 bytes of information.

The file header contains information that describes the file as a whole. It currently occupies nine blocks. It is defined by the following record structure, which is taken from the file BIG_STRUCTS.DEF.

```
structure /BIG_FILE_HDR_DESC/
        integer*4 start_block(0:511)
        integer*4 byte_count(0:511)
        character*24 date
        character*16 col_name
        integer*4 use_index
        integer*4 data_type
        integer*4 subcom_depth
        integer*4 frame_length
        integer*4 trig_cnts(6)
        integer*4 %fill(3)
        record /GEN_DESC/ gd
        record /PRI_TYPE_DESC/ pt
        record /S_TRIG_DESC/ st
end structure
```

With the exception of the start_block field, the contents of the file header may be used by the calling program. Several structure elements must be supplied by the program when creating a new file.

(1) START_BLOCK

The start_block field is an array that contains the channel's starting block number. This field is intended to be used only by the BIG file access routines.

(2) BYTE_COUNT

The byte_count field is an array the contains the number of data bytes for each channel. This field will need to be used by the calling program when reading data from the file. The program may not change this field.

(3) DATE

The date field specifies the date and time that the file was created. The calling program may read but not write this field.

(4) COL_NAME

The col_name field gives the name of the collection point or subsystem that collected the data. The program that creates the data file must supply a value for this field.

(5) USE_INDEX

The use_index field specifies the purpose of the data. Acceptable values are: 0 implies normal gauge data; 1 implies laser calibration data; and 2 implies cable compensation data. This field must be supplied by the program creating the BIG file

(6) DATA_TYPE

The data_type field specifies the type of data that is contained in the file from the following categories: memory data, realtime data, standard or late calibration data, and early calibration data. The program that creates the data file must supply a value for this field.

(7) SUBCOM_DEPTH

The subcom_depth field specifies the maximum subcommutation depth of the data stream format. The program that creates the data file must supply a value for this field. It is not meaningful for RTD BIG files

(8) FRAME_LENGTH

The frame_length field specifies the length, in bytes, of the data stream's major frame. The program that creates the data file must supply a value for this field. It is not meaningful for RTD BIG files.

(9) TRIG_CNTS(6)

The trig_cnts field is an array of trigger counts. The program that creates the data file must supply values for this field. This field in only meaningful for BIG files containing SANDUS data.

(10) %FILL(3)

The %fill field hasn't been defined yet. %fill reserves space in the structure, but does not create a variable name for that space.

(11) RECORD /GEN_DESC/ GD

The GD record is simply a copy of the GEN_DESC record from the ICF. It supplies information such as the test name, source code, trigger channels, etc. The GEN_DESC structure is defined in TABLE_STRUCTS.DEF.

(12) RECORD /PRI_TYPE_DESC/ PT

The PT record is simply a copy of the PRI_TYPE_DESC record from the ICF. It contains values that specify the primary types of all the channels. The PRI_TYPE_DESC structure is defined in TABLE_STRUCTS.DEF.

(13) RECORD /S_TRIG_DESC/ ST

The ST record is simply a copy of the S_TRIG_DESC record from the control file and has the same structure. It contains values that define triggers for SANDUS operations. The record structure is defined in TABLE_STRUCTS.DEF.

The channel headers only contain information that describes the configuration of the hardware that produced the data. They do not contain any information about the file or how the channel data appear in the file. Most of the contents of the header consists of the corresponding channel description record from the control file. The channel description records are defined in TABLE_STRUCTS.DEF.

The calling program has complete access to the channel header and is responsible for supplying the fields to the header before writing the channel data.

E.3 USER CALLABLE ROUTINES

E.3.1 Open_Big

Open_Big opens a BIG data file for processing. You must call it to open a file before you do any other operations on the file. It can either open an existing file or create a new file. When an existing file is opened, Open_Big returns the file header to the calling program. When a new file is to be created, Open_Big returns an empty file header. The calling program must then supply values for some of the file header fields before closing the file. Open_Big returns as the function value a status code that indicates success or failure. The calling program can access the information in the file header by declaring a local file header record and then calling Open_Big. To declare a local copy of the file header the program should include BIG_STRUCTS.DEF and use the following statement:

```
record /BIG_FILE_HDR_DESC/ file_hdr
```

Open_Big and the variable to receive the return status must be declared in the calling program as integer*4 variables.

The calling format for Open_Big is:

```
status = Open_Big ( filename,
                    default_directory,
                    file_status,
                    new_alloc,
                    access,
                    file_header,
                    lun)
```

where the arguments are:

(1) filename

name: filename
type: character string
access: read only

Specifies the name of the file to open or create. The filename argument may contain any or all parts of a full file specification. It may be blank if the default_directory argument contains a valid file name.

(2) default_directory

name: default_directory
type: character string
access: read only

Default directory specification to apply to the filename argument. The default_directory argument supplies any fields in the full file specification that are not supplied by the filename argument. Any fields specified in the filename argument override corresponding fields supplied in the default_directory argument. The default_directory argument may be blank.

(3) file_status

name: file_status
type: character string
access: read only

Status of file to open. The file_status argument must be 'OLD' if you want to open an existing file, and 'NEW' if you want to create a new file. Any other value results in an error being returned. If the value is 'NEW,' then the new_alloc argument must be nonzero.

(4) new_alloc

name: new_alloc
type: integer longword
access: read only

New file allocation size. The new_alloc argument specifies the number of disk blocks to allocate to a new file. It must be nonzero if the file_status argument is 'NEW'. If the file_status argument is 'OLD,' the new_alloc argument is ignored.

(5) access

name: access
type: character string
access: read only

File access mode. The calling program must supply the string 'WRITE' in this argument if the file is to be opened for write access. 'READ' specifies that the file will be opened for READ ONLY access. If the program specifies 'READ' and then tries to write to the file, an error will result.

(6) file_hdr

name: file_hdr
type: record
access: write only

Record to receive the file header. After opening the file, Open_Big reads the file header and returns it the file_hdr argument. If the file is new, the header will be empty except for the date field.

(7) lun

name: lun
type: integer longword
access: read only

Logical unit number to assign to the file. The logical unit number is used in subsequent file operations.

E.3.2 Big_Dir

Big_Dir returns the number of channels that have been written to a BIG data file, and an array that specifies which channels are in the file. The first value in the array is the number of the channel that appears first in the data file; the second array value is the second channel, and so on.

Big_Dir returns as the function value a status code that indicates success or failure. Big_Dir and the variable to receive the return status must be declared in the calling program as integer*4 variables.

The calling format for Big_Dir is:

```
status = Big_Dir (  lun,
                    directory,
                    dir_size,
                    num_chans)
```

The arguments are:

(1) lun

name: lun
type: integer longword
access: read only

Logical unit number assigned to the file. This is the logical unit number used to open the file in the call to Open_Big.

(2) directory

name: directory
type: integer word array
access: write only

Directory of channel numbers contained in the file. Big_Dir writes the channel numbers to the directory in the same order as they appear in the file. The number of channel numbers returned in the directory array is the lesser of dir_size and num_chans.

(3) dir_size

name: dir_size
type: integer longword
access: read only

Size of the directory in words. The calling program must provide a large enough array for the directory to hold all possible channel numbers. If the directory is not large enough to hold all the channel numbers contained in the file, Big_Dir will only return enough channel numbers to fill the array.

(4) num_chans

name: num_chans
type: integer longword
access: write only

Number of channels returned in the directory array. Big_Dir returns this value to the calling program.

E.3.3 Write_Big

Write_Big writes a channel header and channel data out to a BIG file. It also updates the start_block and byte_count fields in the file header. The channel header must be initialized by the calling program and supplied to Write_Big as an argument.

The data for a channel need not all be written out in one call to Write_Big, but may be split into multiple buffers and sent to the routine with multiple calls. In that case, the channel header is only written on the first call.

There is a restriction on the use of Write_Big if you wish to send more than one buffer of data for a channel. Write_Big and Read_Big both use Block I/O mode to read and write data. One characteristic of writing in Block I/O mode is that complete 512-byte disk blocks are written with every write operation. If the buffer to be written is not a multiple of 512 bytes, then the write operation will add enough garbage bytes to the file to fill out the disk block.

For example, if you call Write_Big with a buffer that contains 300 bytes of data, then after the write operation, the file will contain the 300 data bytes followed by 212 bytes of garbage. If you then call Write_Big again with another buffer for the same channel, the second buffer will be written after the 212 bytes of garbage, in effect, causing the garbage to be interspersed with the data. For that reason, the number of bytes in the data buffer sent to Write_Big should be a multiple of 512 bytes, unless it is the last buffer for the channel. If the buffer is the last one or the only one for the channel, then it can be of any size with no problems.

There is no limit to the size of the buffer you can send to Write_Big (aside from the limit of 2,147,483,646 total bytes per channel), so you can always write a complete channel in one call.

Write_Big returns as the function value a status code that indicates success or failure. Write_Big and the variable to receive the return status must be declared in the calling program as integer*4 variables.

The calling format for Write_Big is:

```
status = Write_Big (    lun,
                        channel,
                        chan_type,
                        new_chan,
                        buffer,
                        buffer_size,
                        chan_hdr,
                        bytes_written)
```

The arguments are:

(1) lun

name: lun
type: integer longword
access: read only

Logical unit number assigned to the file. This is the logical unit number used in the call to Open_Big.

(2) channel

name: channel
type: integer word
access: read only

Channel number associated with the data. For realtime channels that are grouped together, channel should be the first channel number in the group.

(3) chan_type

name: chan_type
type: byte
access: read only

Channel type. This is a number that indicates the primary type of the channel. It is the same as the pri_chan_type field in the PRI_TYPE_DESC structure of the control file.

(4) new_chan

name: new_chan
type: boolean longword
access: read only

Indicates whether or not to start writing a new channel. If new_chan is TRUE, Write_Big initializes the start_block and byte_count fields in the file header and writes the channel header to the file. If new_chan is FALSE, Write_Big assumes that the buffer is a continuation of the channel data and starts writing at the block where the previous write operation finished.

(5) buffer

name: buffer
type: array of arbitrary type
access: read only

Data buffer to write. Write_Big treats buffer as an array of contiguous bytes. The type declared by the calling program is unimportant.

(6) buffer_size

name: buffer_size
type: integer longword
access: read only

Size of buffer in bytes. Although Write_Big doesn't care how the buffer was declared in the calling program, buffer_size must give the size in bytes.

(7) chan_hdr

name: chan_hdr
type: record
access: read only

Channel header. The calling program must initialize the channel header fields and then supply the channel header to Write_Big via the chan_hdr argument.

(8) bytes_written

name: bytes_written
type: integer longword
access: write only

Number of bytes actually written to the file. Write_Big returns in this argument the number of bytes it actually wrote to the file, not including the channel header. The value should be the same as the buffer_size argument if the write operation completed successfully. If it encountered an error, bytes_written will indicate the number of bytes Write_Big was able to write before the error occurred.

E.3.4 Read_Big_Hdr

Read_Big_Hdr reads a channel header from a file into a header record supplied by the calling routine. The calling routine may then do whatever it wants with the information in the header. The use of Read_Big_Hdr is optional, in the sense that a program does not need to know the information in the header in order to successfully read channel data from a file. However, it may need to know some of the information in order to be able to process the data. The channel header structure is defined in the file BIG_STRUCTS.DEF.

The calling program can access the information in the channel header by declaring a local channel header record and then calling Read_Big_Hdr. To declare a local copy of the channel header, the program should include the BIG_STRUCTS.DEF file, then use the following statement:

```
record /BIG_CHAN_HDR_DESC/ chan_hdr
```

Read_Big_Hdr returns as the function value a status code that indicates success or failure. Read_Big_Hdr and the variable to receive the return status must be declared in the calling program as integer*4 variables.

The calling format for Read_Big_Hdr is:

```
status = Read_Big_Hdr (   lun,
                          channel,
                          chan_type,
                          header)
```

The arguments are:

(1) lun

name: lun
type: integer longword
access: read only

Logical unit number assigned to the file. This is the logical unit number used in the call to Open_Big.

(2) channel

name: channel
type: integer word
access: read only

Number of the channel from which header is read.

(3) chan_type

name: chan_type
type: byte
access: read only

Channel type. This is a number that indicates the primary type of the channel. It is the same as the pri_chan_type field in the PRI_TYPE_DESC structure of the control file.

(4) header

name: header
type: record
access: write only

Local record variable to receive the channel header. The structure of the record variable must match the structure of the channel header.

E.3.5 Read_Big

Read_Big reads channel data from a BIG file. The data for a channel need not all be read in one call to this routine, but may be read into multiple buffers using multiple calls.

The user should be aware of a situation that occurs when Read_Big is used to read the data into more than one buffer with multiple calls. Read_Big and Write_Big both use Block I/O mode to read and write data. One characteristic of reading in Block I/O mode is that complete 512-byte disk blocks are read with every read operation. For example, if the user wants to read 300 bytes of data from a file using Block I/O, the system will read a complete 512-byte block from the file, but only load the buffer with the first 300 bytes. Subsequent read operations for the same channel would start on subsequent blocks, so that the 212 bytes that didn't get loaded into the first buffer would never be transferred. If the requested number of bytes included all the data for the channel, then there would be no problem, but if the data were split into multiple buffers and the buffers were not multiples of 512 bytes, then some of the data would not be transferred.

Read_Big avoids this problem by automatically sizing the buffer if the number of bytes requested is not a multiple of 512 and if the number does not account for all the data in the channel. For example, if there were 2000 bytes of data for the channel and the calling program requested 1200 bytes, then Read_Big would trim the request to 1024 bytes. Read_Big also trims the request if it is more than the amount of data left to be read for the channel. For example, if there were 2000 bytes of data and the calling program requested 2500, bytes Read_Big would return 2000 bytes.

Read_Big returns two arguments besides the data buffer, the number of bytes actually read and the number of bytes left to read for a channel. If the calling routine supplies a buffer that is a multiple of 512 and no errors occur during the read, then the number of bytes actually read should be the same as the requested number. If Read_Big trims the requested number, then the two counts will not agree, which is a normal condition.

There is no limit to the size of the buffer you can send to Read_Big (aside from the limit of 2,147,483,646 total bytes per channel), so you can always read a complete channel in one call.

Read_Big returns as the function value a status code that indicates success or failure. Read_Big and the variable to receive the return status must be declared in the calling program as integer*4 variables.

The calling format for Read_Big is:

```
status = Read_Big ( lun,
                    channel,
                    chan_type,
                    new_chan,
                    buffer,
                    buffer_size,
                    bytes_read,
                    bytes_left,
                    start_block)
```

The arguments are:

(1) lun

name: lun
type: integer longword
access: read only

Logical unit number assigned to the file. This is the logical unit number used in the call to Open_Big.

(2) channel

name: channel
type: integer word
access: read only

Channel number to read.

(3) chan_type

name: chan_type
type: byte
access: read only

Channel type. This is a number that indicates the primary type of the channel. It is the same as the pri_chan_type field in the PRI_TYPE_DESC structure of the control file.

(4) new_chan

name: new_chan
type: boolean longword
access: read only

Indicates whether or not to start reading a new channel. If new_chan is TRUE, Read_Big considers the request to be the first read operation on the channel data, so it will start reading at the first block. If new_chan is FALSE, Read_Big assumes that the buffer is a continuation of the channel data and starts the read from the point at which the previous read left off.

(5) buffer

name: buffer
type: array of arbitrary type
access: write only

Buffer to read data into. Read_Big treats buffer as an array of contiguous bytes, so the type that the calling program declares it to be is unimportant.

(6) buffer_size

name: buffer_size
type: integer longword
access: read only

Size of buffer in bytes. Although Read_Big doesn't care how the buffer was declared in the calling program, buffer_size must give the size in bytes.

(7) bytes_read

name: bytes_read
type: integer longword
access: write only

Number of bytes actually read. This argument will always be less than, or equal to, buffer_size. A disagreement between the two values does not necessarily mean an error occurred.

(8) bytes_left

name: bytes_left
type: integer longword
access: write only

Number of bytes left to read for a channel. This argument will always be greater than, or equal to, zero.

(9) start_block

name: start_block
type: integer longword
access: write only

Number of block in the BIG file where this read started. This argument will always be greater than zero.

E.3.6 Close_Big

Close_Big closes a BIG file. Its use is optional, but recommended. Close_Big returns as the function value a status code that indicates success or failure. Close_Big and the variable to receive the return status must be declared in the calling program as integer*4 variables.

The calling format for Close_Big is:

status = Close_Big (lun)

The argument is:

(1) lun

name: lun
type: integer longword
access: read only

Logical unit number assigned to the file. This is the logical unit number used in the call to Open_Big.

E.4 Implementation

As mentioned above, all of the BIG file utility routines return status values to the calling program that indicate the success or failure of the routines. The status codes are defined in such a way that success codes are odd values and failure codes are even values. In VAX FORTRAN the user can test numeric values as logical expressions; odd values test TRUE and even values test FALSE. The user should always test the return status and take appropriate actions. If the routine failed, write out an error message. There is a routine named Bad_Status that will convert a status code to an ASCII error message and print it out on the terminal. Bad_Status is called as a subroutine with three arguments. The arguments, in order, are:

(1) calling_name

name: calling_name
type: character string
access: read only

Name of the calling routine. This name gets printed out in order to show which routine the program was in when it tried to call the BIG file utility routine.

(2) called_name

name: called_name
type: character string
access: read only

Name of the called routine. This name gets printed out in order to show which utility routine returned the error status.

(3) status

name: status
type: integer longword
access: read only

Status code returned by the called routine. The status code is converted to a system-supplied error message, and the error message is printed out.

A typical example is:

```
program Main
integer*4 Open_Big, status
      .
      .
      .
status = Open_Big ( ... )
if (status) then        ! Success
      .
      .
      .
else                    ! Failure
   call Bad_Status ('Main', 'Open_Big', status)
      .
      .
      .
end if
```

There is a program named BIGREAD in the directory LD:[REPAIR] that uses Open_Big, Big_Dir, Read_Big_Hdr, Read_Big, and Close_Big. It reads a channel header and, optionally, data from a raw data file; then, it formats and writes the information out in readable form to the terminal and in an output file. The user may want to run it on any of the BIG data files (there are many scattered around the system). By looking at the program itself, the user will become familiar with how it uses the various routines.

The BIG routines and Bad_Status are kept in an object library named UTILITY.OLB in the directory UTILITY$LIBRARY, along with other useful utility routines. There is an alternate library, UTILITY.DBG_OLB, that has the identical routines as UTILITY.OLB, but with the difference that it has been compiled /DEBUG. Before you can run a program that calls these routines, you must link them with your program. To do so, use the following command (or your own variation):

```
$ DEFINE UTILITY$LIBRARY  "directory holding the UTILITY libraries"
$ LINK PROG,UTILITY$LIBRARY:UTILITY/LIBRARY
```

OR

```
$ LINK/DEBUG PROG,UTILITY$LIBRARY:UTILITY.DBG_OLB/LIBRARY
```

APPENDIX F

MAINTENANCE OF ICF (TABLE), BIG, AND CHANNEL FILE UTILITY ROUTINES

APPENDIX F

MAINTENANCE OF ICF (TABLE), BIG, AND CHANNEL FILE UTILITY ROUTINES

F.1 INTRODUCTION

This Appendix describes the changes that must be made in the ICF Utility Routines if TABLE_STRUCTS.DEF, BIG_STRUCTS.DEF, BIG_DEF.PRM, and/or BIG_STRUCTS.PRM are changed.[1] Changes in any one or all of these files will affect most of the Nevada Test Site (NTS) Instrumentation System programs. A change in CHN_HEADER.DEF will affect only ANALYZE, PROCESS, PRELEWD, and DSP. Because of this dependency, the user must look carefully at all the source code and understand records and structures before doing anything. It is best if only one person has responsibility for maintenance of all the ICF routines and their utility programs.

The above five files are in INCLUDE$INCLUDE, which is defined as:

$ DEFINE INCLUDE$INCLUDE LD:[INCLUDE] (or wherever they are maintained)

There have been two changes that have occurred in these files. The first, and most significant, is the addition of a new digitizer. This requires additions to every one of the above files and to CHN_HEADER.DEF. The steps necessary for this are described in paragraph 2. The second change is the addition (or deletion) of elements from existing structures, which requires only a record length determination. The subset of steps necessary for this are listed in paragraph 3.

The first step, regardless of what is to be changed, is to verify each and every addition and deletion with **ALL** interested parties. When the ICF creator and all interested parties have approved every change and the user has backed up all files to a neutral directory, changes may be made.

---

[1] Much of this document and all of the routines described herein were written by Jon Anspach when he worked for Organization 9321. These routines have been modified when necessary and documentation updated by Peter Kaestner of Department 9321. The files that were called tables are now Instrument Control Files (ICFs).

F.1.1 Record Type

Without a great deal of thought, the user should NOT change the x_REC_TYPE variable associated with every structure. The value of this variable is unique, and identifies the structure to the table routines and most other programs. The value is determined from the structure's position in TABLE_STRUCTS.DEF; it must not be greater than the dimension of the variables REC_NAME and REC_LEN (currently 40) in the structure STRUCT_DESC. This dimension can be changed.

All the major NTS Instrumentation System programs test this value to determine the data's source. Each program has its own parameter file defining the significant values. The structures named UNUSED... in TABLE_STRUCTS.DEF may be used for new structures without causing problems, but the STRUCT_REC_TYPE variable value should not be changed.

F.2 ADDING A NEW DIGITIZER

BACKUP ALL FILES!!!

Regardless of what else is going on, ALWAYS backup these files to a neutral directory before editing them.

Adding a new digitizer will require defining the necessary structures in TABLE_STRUCTS.DEF and a new CAL structure in CHN_HEADER.DEF. This work can be done by the person(s) responsible for integrating the digitizer into the NTS Instrumentation System. The changes described here must be done after that work is done.

F.2.1 Additions to BIG_STRUCTS.DEF

Adding a new digitizer will cause many significant changes to a host of programs.
The following is part of BIG_STRUCTS.DEF showing the form of the header description for an RTD720 channel. Any new digitizer will need a similar header description created for it in BIG_STRUCTS.DEF.

. . . fragment from BIG_STRUCTS.DEF . . .

```
            ! This structure is used for a RTD720 channel.
structure /RTD_HDR_DESC/
      integer*4 %fill(5)    ! Unused (to be defined)
      integer*4 fill        ! =1 ==> TBL_STREAM_MIS=bad (F)
                            ! =0 ==> TBL_STREAM_MIS=good (T)
      record /RTD_DEV_DESC/ rtd     ! Defined in Table_Structs.def
      record /RTD_CHAN_DESC/ rtdc  ! Defined in Table_Structs.def

      union
        map
            record /RTD_EXP_SUB_DESC/ rtdes(0:rtd_exp_sub_max)
                            ! Parameter defined in Big_Structs.prm
        end map             ! Defined in Table_Structs.def
        map
            record /RTD_LC_SUB_DESC/ rtdls(0:rtd_lc_sub_max)
                            ! Parameter defined in Big_Structs.prm
        end map             ! Defined in Table_Structs.def
        map
            record /RTD_CC_SUB_DESC/ rtdcs(0:rtd_cc_sub_max)
                            ! Parameter defined in Big_Structs.prm
        end map             ! Defined in Table_Structs.def
      end union
end structure
```

. . . end of fragment . . .

F.2.2 Obtain Record Lengths

The next step is to obtain the lengths (in bytes) of all structures (hence records) in TABLE_STRUCTS.DEF, BIG_STRUCTS.DEF, and in CHN_HEADER.DEF. This can be done by counting the bytes, or by writing a small program that declares records for every structure. By compiling this program with the qualifier /LIST and examining the .LIS file produced, these lengths are available.

The following is an excerpt from an actual program. Notice how it gets the lengths of the individual records as well as CHN file headers and BIG file headers.

```
      program chn_headtest
      include 'include$include:big_structs.def/nolist'
      include 'include$include:chn_header.def/nolist'
      .
      .
      .
c RTD structures
c for the CHN file header with USE_INDEX=0, use in CHN_HEADER.DEF
c
      structure / rtd_exp /
            record / general_chan / gc
            record / gen_desc / g
            record / rtd_dev_desc / al
            record / rtd_chan_desc / n
            record / rtd_exp_sub_desc / ns
            record / cal_rtd / oo
      end structure
      record / rtd_exp / re
c for the BIG file structure /RTD_HDR_DESC/, for use in BIG routines
c
      record /RTD_HDR_DESC / rtd
      .
      .
      .
      end
```

If you have access to the program FORTRAN-lint (FLINT), you can have it analyze a small program that contains the include files describing the structures. FLINT will compute the record lengths if the /XREFERENCE qualifier is present. There may be other source code analyzers that do a similar job.

F.2.3 Changes to BIG_DEF.PRM

Edit BIG_DEF.PRM. Change, and add as necessary, parameter statements that define BIG file header record lengths to those determined in the previous step. Using the existing parameter

statements as a template, add similar statements for any new channel types. Modify existing parameters to reflect the conditions for the next event. The following file fragment shows where some of this needs to be done.

. . . fragment from BIG_DEF.PRM . . .

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
!
! The following parameters define the file and channel header sizes.
!
!                        ***** IMPORTANT *****
!
! THE FILE AND CHANNEL HEADER BYTE SIZES SPECIFY THE NUMBER OF BYTES TO
! READ OR WRITE FOR HEADER I/O OPERATIONS.  THEY ARE NOT NECESSARILY THE
! SAME AS THE HEADER LENGTHS.  THE HEADERS CAN BE OF ANY LENGTH, BUT THE
! BYTE SIZES SPECIFIED HERE MUST BE EVEN NUMBERS.  THEREFORE, IF THE
! ACTUAL HEADER SIZE IS AN ODD NUMBER OF BYTES YOU MUST ROUND UP TO THE
! NEXT EVEN NUMBER IN THE PARAMETER STATEMENTS BELOW.  IN THAT CASE YOU
! SHOULD ALSO MAKE SURE YOU ALLOW SPACE FOR THE EXTRA BYTE IN ANY
! PROGRAMS THAT READ HEADERS; OTHERWISE, YOU MIGHT OVERWRITE PART OF YOUR
! PROGRAM BY ONE BYTE.
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

      integer*4 B_FILE_HDR_BYTES
      parameter (B_FILE_HDR_BYTES = 4980)

      integer*4 B_FILE_HDR_BLOCKS
      parameter (B_FILE_HDR_BLOCKS = (B_FILE_HDR_BYTES +
      1     BYTES_PER_BLOCK - 1) / BYTES_PER_BLOCK)

      .
      .
      .

      integer*4 B_RTD_HDR_BYTES
      parameter (B_RTD_HDR_BYTES = 9180)

      integer*4 B_RTD_HDR_BLOCKS
      parameter (B_RTD_HDR_BLOCKS = (B_RTD_HDR_BYTES + BYTES_PER_BLOCK - 1) /
      1                                BYTES_PER_BLOCK)

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!       End of file and channel header sizes.
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
                         . . . end of fragment . . .
```

F.2.4 Changes to CHN_HEADER.DEF

Edit CHN_HEADER.DEF. Modify the file as necessary to reflect the structure lengths obtained in Paragraph F.2.2. Much of this editing is editing comment fields that guide the user through CHN_HEADER.DEF, the determination of CHN header lengths in 512 byte blocks. Any addition or deletion of a digitizer requires modification of the CHN_FILE_HDR_DESC structure, which is in CHN_HEADER.DEF. See both Section 5, and Appendix D of the Maintenance Manual.

F.2.5 Changes to the BIG File Routines

If a new digitizer was added to, or dropped from, BIG_STRUCTS.DEF and TABLE_STRUCTS.DEF (which implies a change to structures in CHN_HEAD), then you must edit the following four files:

(1) READ_BIG.FOR
(2) READ_BIG_HDR.FOR
(3) WRITE_BIG.FOR
(4) WRITE_BIG_HDR.FOR

Each source file contains comments indicating the changes to be made. It will consist of adding and/or deleting an ELSE IF(...) statement. I have included a fragment from the file READ_BIG.FOR to illustrate the nature of these changes.

. . . fragment from READ_BIG.FOR . . .

```
!
! *****  This IF statement must have a condition for every channel type.  If
! *****  a new channel type is added you must add a condition here to handle
! *****  it.  The parameter constants used are defined in
! *****  INCLUDE$INCLUDE:BIG_DEF.INC.
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

      if (chan_type .eq. S_ALOG_TYPE) then
            chan_hdr_blocks = B_S_ALOG_HDR_BLOCKS
```

```
        else if (chan_type .eq. S_DIG_TYPE) then
                chan_hdr_blocks = B_S_DIG_HDR_BLOCKS

        else if (chan_type .eq. S_AMUX_TYPE) then
                chan_hdr_blocks = B_S_AMUX_HDR_BLOCKS

        else if (chan_type .eq. T7912_TYPE) then
                chan_hdr_blocks = B_T7912_HDR_BLOCKS

! deleted 5/20/1992 by PCK Device no longer in use
!       else if (chan_type .eq. L6880_TYPE) then
!               chan_hdr_blocks = B_L6880_HDR_BLOCKS

        else if (chan_type .eq. T7103_TYPE) then
                chan_hdr_blocks = B_T7103_HDR_BLOCKS

        else if (chan_type .eq. RTD_TYPE) then
                chan_hdr_blocks = B_RTD_HDR_BLOCKS

        else
                Read_Big = 0
                return
        end if
```

. . . end fragment . . .

F.2.6 Recompile Object Libraries

Recompile all the TBL and BIG modules, and recreate the object libraries using the two command files available. They are BIG_COMPILE_ALL.COM and TBL_COMPILE_ALL.COM. Do not introduce the new libraries and structure definition files until all interested parties are fully informed and indicate they are ready for this change.

Throughout the documentation, various symbols such as INCLUDE$INCLUDE have been defined. The purpose is now clear. During checkout, redefine the necessary symbols locally to point to the latest set of files, while leaving the source code unchanged. Compile and debug, and when everyone is ready, restore the original definitions.

F.2.7 Recreate CREATE_TBL.EXE

Recompile and relink CREATE_TBL.FOR, which is used to create an empty ICF. It is located in TOOLS$LIBRARY, which is defined as:

$ DEFINE TOOLS$LIBRARY LD:[TOOLS] (or wherever file maintained)

For more information on CREATE_TBL see Appendix B.

F.3 ADDITION/DELETION OF ELEMENTS FROM STRUCTURES

If it is necessary to add, or delete elements from existing structures, then the steps described in paragraphs F.2.2, F.2.3, F.2.4, and F.2.6 should be followed.

F.4 CHANGES TO THE BIG_FILE_HDR_DESC STRUCTURE

If the structure BIG_FILE_HDR_DESC in BIG_STRUCTS.DEF is changed, then all the routines described in the Appendix E of the Maintenance Manual must be examined for the impact of that change. Specifically, CLOSE_BIG.FOR must be changed to make sure the changed field is properly handled. This step depends on a through understanding of what is happening in these routines and in the nature of the change. FETCH, DECOM, and PROCESS will have to be modified in addition to REALIZE and ANALYZE. It would be unusual to modify this structure. The instructions in paragraph F.2.3, and F.2.6 should be followed.

APPENDIX G

SUMMARY LOG UTILITY (SLOG)

APPENDIX G
SUMMARY LOG UTILITY (SLOG)

G.1 INTRODUCTION

The execution of each major piece of software in the Nevada Test Site (NTS) Instrumentation System creates a log file. This file is very useful to the programmer and needs to be retained. The NTS operations people need a summary containing only significant entries from all the log files.

The Summary Log Utility (SLOG) is a utility that runs on the NTS Instrumentation System after all the activity associated with an event (or dryrun) has ended. It examines all LOG files known to it for records containing a programmer-determined severity code below a known threshold. This value, along with other coded values in a specific set of columns within the record, and the text of the record are sorted and written to an editable/printable file, which is distributed to the pertinent project leaders.

For each event, SLOG:

(1) reads each log file known to it,
(2) deletes column 1 carriage control information,
(3) extracts only tagged lines with a severity level less than some threshold, and writes them to a sort file
(4) sorts this file, and
(5) prints the sorted file as a summary log file.

G.2 USING SLOG

There are three distinct operations necessary to use SLOG. They are:

(1) Determine where the desired LOG files will be located when the event activities are over and write DCL to capture their name and location in a directory file.

(2) Parse the file thus generated to generate a second file in which the location of each log file is defined, one log file per line.

(3) Use this second file as command line parameter p1 to run SLOG.

G.2.1 Finding the LOG FILES

Included are excerpts from the command file SLOG$LIBRARY:PREPARE_P2.COM, which has been used to find the log files for an event, capture them in the directory file LOG_FILES.DAT, run the program SLOG$LIBRARY:PARSE_FILE, and, finally, run SLOG. The important DCL commands have been bolded.

. . . excerpts from PREPARE_P2.COM . . .

```
$!
$!  PREPARE_P2.COM prepares a file for SLOG which
$!  consists of [node::]disk:[directory]filename.type;ext
$!  one file per line.  This requires reworking log_files.dat,
$!  which is created with the following directory request
$!  $ dir/brief/col=1/output=file.lis.  The user needs to set up the
$!  specific file type to be searched for, and the file names he may
$!  wish to use.  The summary.log file is left in the current directory,
$!  and is set so the owner cannot, without thought, delete it.
$!
$       dir/brief/col=1/output=log_files.dat/since=9:30 -
              gear10::dd:[source.rmv11]*.log, -
                                  . . .
              n14vax::dd:[hddr.san504]*.log, -
                                  . . .
              n14vax::dd:[mm.ace11]*.log, -
                                  . . .
              n12vax::dd:[mm.ace11]prelewd.log, -
                                  . . .
              n12vax::dd:[mm.san504]prelewd.log, -
                                  . . .
              hddra::dd:[hddr.ace11]fetch.log, -
                                  . . .
              mm12c::dd:[mm.ace11]decom.log, -
                                  . . .
```

. . . end of excerpt . .

It is important that the search be as carefully specified as possible, because no date/time evaluation is done, and only the log files from the current event are meaningful. It requires a slight change to pick up log files from an archived set.

The "/brief/col=1" form of the directory command generates a very nice format as shown below:

```
Directory DD:[SOURCE.RMV11]

ANALYZE.LOG;1
INITIALIZE.LOG;1
PRELEWD.LOG;3
PRELEWD.LOG;2
REALIZE.LOG;1

Total of 4 files.
[EOB]
```

G.2.2 Parsing the Directory Lists

The file LOG_FILES.DAT then contains a collection of "Directory...Total of " groups, one group for each directory searched. The program PARSE_FILE looks for a file of named LOG_FILES.DAT (the name is a DATA statement).

PARSE_FILE opens the file READONLY and finds the string "directory", picks up the specification string and appends it to the beginning of each file, until a string "total.." is found. Each full file specification is then written to a new file, in that case LOG_FILES.LIS (the name is in a data statement). This sequence is repeated until an EOF is found.

LOG_FILES.LIS now looks as follows (using the LOG_FILES.DAT from above):

```
DD:[SOURCE.RMV11]ANALYZE.LOG;;1
DD:[SOURCE.RMV11]INITIALIZE.LOG;1
DD:[SOURCE.RMV11]PRELEWD.LOG;3
DD:[SOURCE.RMV11]PRELEWD.LOG;2
DD:[SOURCE.RMV11]REALIZE.LOG;1
```

. . . resume PREPARE_P2.COM excerpt . . .

```
$!  Now we take LOG_FILES.DAT and use it in PARSE_FILE to create
$!  the file expected by SLOG.
$!
$    define slog$library  ld:[slog]  ! or wherever file maintained
$!
$       run/nodebug slog$library:parse_file
$!
$!  Creates LOG_FILES.LIS from LOG_FILES.DAT, where LOG_FILES.LIS
$!  is the output file for SLOG, but first define the command.
$!
```

. . . end of excerpt . . .

G.2.3 Running SLOG

To use SLOG, enter the following DCL commands:

```
$ DEFINE  SLOG$LIBRARY LD:[SLOG] (or wherever file maintained)

$ SET COMMAND SLOG$LIBRARY:SLOG_COMMAND_DEF
$ SLOG "filename"
```

"Filename" is a required command line parameter that is the name of a file containing one full pathname:filename description for every LOG file SLOG is to examine.

The specifications for the LOG file are described in paragraph G.3.

SLOG has two optional command line qualifiers, /LEVEL and /QUEUE. LEVEL is the threshold level below which the severity code is significant. Its default value is 5, its range 0-9. QUEUE directs the output to the specified queue, with the default queue being SYS$PRINT. Because of the possibility of a pervasive error across the entire system, SUMMARY.LOG is not printed. Large SUMMARY.LOG files are not useful, but should always be examined with a text editor. It is always a good idea to protect SUMMARY.LOG against deletion, in order to keep a history of problems should it become important. The file should be archived along with the data. All LOG files should be archived for a short period of time.

. . . resume excerpt of PREPARE_P2.COM . . .

```
$    set command slog$library:slog_command_def
$!
$       slog log_files.lis
$!
$!
$       write sys$output "A summary.log file has been written, and should"
$       write sys$output "be printed if not too long ( >600 blocks)."
$!
$!
$       set protection=o:rwe summary.log;*
$!
```

. . . end of excerpt . . .

G.3 SPECIFICATIONS FOR THE LOG FILE

To implement SLOG, it is necessary that each log file processed have a format consisting of a header (repeated on each page) and the body. The file is written in ASCII, with carriage control information in column 1. Each record consists of not more than 113 characters, where columns 2-100 are used for log file messages and columns 101-113 are optional, and, if included, are formatted as described later.

Each page of the log should have a header consisting of the following information in columns 2-100 (columns 101-113 should not be present for the header):

(a) page number
(b) a date-time stamp
(c) the node the log was created on
(d) the data source and collection point
(e) the program generating the log file.

The body will consist of messages similar to those now generated, but limited to columns 2-100 inclusive. A message requiring multiple lines is possible, the sort will be done preserving the original order.

Columns 101-113 are present for SLOG. This field is created by the program generating the log file and will require some judgment by the programmer. If this field is missing, the record is ignored. If present, the record is important enough to be included in the summary. What is important is determined by either the programmer or the operations people.

If columns 101-113 are present, column 101 is blank and columns 102-113 are a tag, which can be sorted in a number of ways. To insure a consistent sort, all tag alphabetic characters should be in upper case. In the CHN file name CHNsnnrmmm-kk.DAT, the snnrmmm part may be lifted out to fill positions 1-7 in the tag. This thirteen character tag is coded as follows:

Position 101 is currently blank, and should not be used.

Position 102-104 contains a three character alphameric source designation (the snn in CHNsnnrmmm-kk.DAT), i.e., R21, for RMV 21, A11 for Ace 11, or S04 for Sandus 504.

Position 105 contains the new one character collection point designation (the r in CHNsnnrmmm-kk.DAT), i.e., S for source, M for Mass Memory, T for HDDR tape.

Position 106-108 contains a three character numeric channel designation (the mmm in CHNsnnrmmm-kk.DAT).

Position 109-110 contains a two character numeric subchannel designation that may be blank for single experiment channels or when the subchannel designation is not meaningful (the kk in CHNsnnrmmm-kk.DAT).

Position 111 contains one alphabetic character that identifies the specific collection point, ie., (currently A, B, C, or D, as in MASS MEM B or HDDR C).

Position 112 contains the one character originating program designation, ie., A for AUTOCAL, C for INITIALIZE, AND R for REALIZE. The choice of designation letter is dictated by the sorting sequence, with the idea that programs that are run first should sort first. For a complete list see the DATA statement in module INTERP.

Position 113 contains a one character numeric severity level designation.

Any field may be left blank when it is not applicable, ie., REALIZE and FETCH/DECOM do not know about subchannels. Subchannels are not meaningful in CAL files.

SLOG discards messages with a severity level greater than some value m (currently 5 but can be changed on the command line). The SUMMARY Log listing for a channel would go from the most significant error to the least significant error. Channel messages from both collection points would sort together. Positions containing a blank would sort to the top of the file, which is where general messages should appear.

G.4 MAINTAINING SLOG

All the SLOG source code is maintained in one file SLOG$LIBRARY:SLOG.FOR. It consists of the following modules:

(1) **SLOG** - This is the main program and controls the logic. It OPENs and CLOSEs each LOG file found in the file of LOG files, command line parameter p1.

(2) **NEXT_LINE** - Extracts the next record (line) from the ASCII file on the logical unit supplied as an argument. It detects READ errors, and EOF, and passes the information to the calling program.

(3) **LOGFILE** - Rearranges the tag field (cols 102-113) into a form more suitable for sorting.

(4) **ENVIRONS** - OPENs the output files SLOG.LOG, and SUMMARY.LOG, and puts the preliminary information in the LOG file.

(5) **GET_COMMAND_LINE** - Evaluates the current command line.

(6) **START_SORT** - Maintains the sort parameters and keys that define how the extracted lines will be sorted, the primary and secondary sorts, etc. It names a reference to the VAX/VMS SORT Routines.

(7) **RECOVER** - Recovers the sorted records, one at a time, and calls modules that format the record for output to the SUMMARY.LOG.

(8) **FOUND** - Is a standard routine that writes to the LOG file. For a complete discussion of FOUND, see Section 5 of the Maintenance Manual.

(9) **INTERP** - takes the sorted record and creates a SUMMARY.LOG record, where the coded tag has been expanded.

(10) **RITE_SUMMARY** - Write the record created by INTERP, counting the lines, and making a tidy document.

(11) **CLI** - Uses the Command Language Utility to parse the command line.

(12) **INDIRECT** - Parses the parameter p1, which is a file containing filenames.

(13) **SUBSTITUTE** - Any character found in a specified string that matches one of the characters in a second specified string is replaced with a specified character.

(14) **LIMITED** - Any character found in a specified string that is not one of the characters in a second specified string is replaced with a specified character.

There are two command files that are useful. SLOGLIB.COM compiles a module with the /DEBUG/NOOPT options and puts the object in the object library SLOG.OLB. SLOG.COM uses SLOGLIB.COM to compile modules, then links two executable files, SLOG.EXE, and DEBUG_SLOG.EXE. The link requires two external libraries, PRELEWD.OLB and UTILITY.OLB or UTILITY.DBG_OLB.

Appendix H

UTILITY LIBRARY

H.0 UTILITY LIBRARY

H.1 INTRODUCTION

A good, well-documented, utility library is a valuable addition to any software project. This appendix documents all the FORTRAN modules in the utility library **in alphabetical order** except the screen management utilities. They are used only by the SUPERMON software and are documented there. Each module header is listed and contains the module's purpose, and the input and output arguments. The source software often contains additional detail about the purpose or technique used.[1]

H.2 USEFUL COMMAND FILES

Five command procedures in this library can be used to compile part or all of the library. Each module is compiled two ways, with or without the /NOOPT/DEBUG options. In both cases, the object module is placed in an object library, UTILITY.DBG_OLB and UTILITY.OLB, respectively.

H.2.1 UTILITY_COMPILE.COM

This command procedure requires only one argument, the filename of the file to be compiled. If it is not present, a prompt is issued. The object module inserted into UTILITY.OLB is compiled with the following command:

```
$ fortran/list 'P1' -
        /nocheck -
        /extend_source
```

---

[1] Most of the routines described herein were written by Jon Anspach when he worked for Organization 9321. A number of these routines are described in more detail in Appendixes B and E. The two utility libraries, UTILITY.OLB and UTILITY.DBG_OLB, are used by all NTS instrumentation software.

The object module inserted into UTILITY.DBG_OLB is compiled with the following command:

```
$ fortran 'P1' -
        /check -
        /debug -
        /d_lines -
        /extend_source -
        /list -
        /nooptimize
```

H.2.2 UTILITY_COMPILE_ALL.COM

This command procedure deletes then recreates the two libraries, and uses UTILITY_COMPILE.COM on each .FOR file in the directory.

H.2.3 UTILITY_BATCH.COM

This command procedure sets the default to the UTILITY directory, and then executes UTILITY_COMPILE_ALL.COM.

H.2.4 TBL_COMPILE_ALL.COM

This command procedure is similar to UTILITY_COMPILE_ALL.COM, but only compiles files whose name matches the wildcard file specification *TBL*.FOR. The compilation is done using UTILITY_COMPILE.COM.

H.2.5 BIG_COMPILE_ALL.COM

This command procedure is similar to UTILITY_COMPILE_ALL.COM, but only compiles files whose name matches the wildcard file specification *BIG*.FOR. The compilation is done using UTILITY_COMPILE.COM.

This is file LD:[UTILITY]ABORT_PROG.FOR;17

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Abort_Prog
!
! FUNCTION:     This routine writes out an error message and aborts the program.
!               The error message is based on "code", and "lun" is the unit
!                       to which it is written.
!
! COMPILER:     VAX/VMS FORTRAN
!
! ARGUMENTS:
!
!       integer*4 lun           ! (read only) Logical unit to write messages to
!       integer*4 code          ! (read only) Status code of error
!
! DATE:                 June 10, 1987
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!               In support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!
! --------------------- R E C O R D   O F   U P D A T E S ------------------
!
!  ID   DATE                        REASON
! ---  --------  ------------------------------------------------------------
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
This is file LD:[UTILITY]ADD_TBL_REC.FOR;16

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Add_Tbl_Rec
!
! FUNCTION:     This routine adds a record to a table.
!               For additional information see ld:[doc]table_routines.doc
!                       and File_routines_maint.doc
!
! COMPILER:     VAX/VMS FORTRAN
!
! ARGUMENTS:
!
!   Name          Access    Description
!   ----          ------    -----------
!   lun           Read      Logical unit number of table
!   rec_name      Read      Name of record to write
!   rec_index     Read      Array index if the record is an array element
!   sub_rec       Read      Sub-record number
!   byte_array    Read      Input argument for the record
!
! DATE:                 December 23, 1987
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!                   in support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]ASSIGN_CHAN.FOR;5

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Assign_Chan
!
! FILENAME:     Assign_Chan.for
!
! FUNCTION:     Assign_Chan assigns an I/O channel to a device and returns the
!               channel number.
!
! COMPILER:     VAX/VMS FORTRAN
!
! INPUT:        device          - (read only) Device to assign the channel to
!
! OUTPUT:       Channel number of device is returned as the function value
!
! CALLS:        Bad_Status      - Reports a bad return status from a system call
!               Sys$Assign      - Assigns an I/O channel to a device
!
! DATE:                 January 21, 1986
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!               In support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!
! CHECKER:
!
! ---------------------- R E C O R D   O F   U P D A T E S ------------------
!
!  ID   DATE                        REASON
! ---  --------  -------------------------------------------------------------
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]BAD_STATUS.FOR;7

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                  Bad_Status
!
! FILENAME:     UTL_Bad_Status.for
!
! FUNCTION:     Bad_Status reports a bad return status from a system call.  The
!               first two arguments specify traceback information about the
!               calling routine.
!
! COMPILER:     VAX/VMS FORTRAN
!
! INPUT:        routine         - Character string specifying the name of the
!                                 calling routine
!               tracer          - Character string specifying any information
!                                 the calling routine deems necessary to locate
!                                 the error
!               sys_err_code    - Return status from the system call
!
! OUTPUT:       Messages written to SYS$OUTPUT
!
! CALLS:        Lib$Sys_Getmsg  - Gets the text message associated with an error
!                                 code
!
! DATE:                  May 6, 1986
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!               In support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!
! CHECKER:
!
! --------------------- R E C O R D   O F   U P D A T E S ------------------
!
!  ID   DATE                       REASON
! ---  --------  ------------------------------------------------------------
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]BIG_DIR.FOR;19

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Big_Dir
!
! FUNCTION:     Big_Dir provides a directory of the channels in a big data
!               file.  The directory is in the form of an integer array that
!               contains channel numbers in the order they appear in the
!               file.  The total number of channels in the file is also
!               supplied through an argument.
!               For additional information see ld:[doc]big_routines.doc
!
! COMPILER:     VAX/VMS FORTRAN
!
! INPUT:        dir_size        - Size of directory
!               lun             - Logical unit number of the BIG file
!
! OUTPUT:       dir             - Directory to return
!               num_chans       - Number of channels in the file
!
! DATE:                 January 6, 1986
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!                   In support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]BSEARCH_C.FOR;5

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Bsearch_c
!
! FILENAME:     Bsearch_c.for
!
! FUNCTION:     Bsearch_c conducts a binary search on an array of character
!               strings.  If the target string is found its index is returned
!               as the function value.  If the string is not found a zero is
!               returned.  Bsearch_c assumes that the array is sorted in
!               ascending order.
!
! COMPILER:     VAX/VMS FORTRAN
!
! INPUT:        character*(*) array(size) array to be searched
!               integer*4 size Dimension of that array
!               character*(*) string  string to be searched for
!
! OUTPUT:       integer*4 bsearch_c   0 ==> string not found
!                                  .ne. 0 ==> index of string in array
!
! CALLS:        n/a
!
! DATE:                 November 3, 1986
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!               In support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!
! CHECKER:
!
! ----------------------- R E C O R D   O F   U P D A T E S --------------------
!
!  ID    DATE                          REASON
! ---  --------  -------------------------------------------------------------
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]BSEARCH_L.FOR;4

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Bsearch_l
!
! FILENAME:     UTL_Bsearch_l.for
!
! FUNCTION:     Bsearch_l conducts a binary search on an array of longword
!               integers.  If the target integer is found its index is returned
!               as the function value.  If the integer is not found a zero is
!               returned.  Bsearch_l assumes that the array is sorted in
!               ascending order.
!
! COMPILER:     VAX/VMS FORTRAN
!
! INPUT:        integer*4 array(size)    to be searched
!               integer*4 size           dimension of array
!               integer*4 number         goal of search
!
! OUTPUT:       integer*4 bsearch_l      0 ==> "number' not found in array
!                                   .ne. 0 ==> index of 'number' in aray
!
! CALLS:        n/a
!
! DATE:                 November 3, 1986
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!               In support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!
! CHECKER:
!
! ...................... R E C O R D   O F   U P D A T E S ....................
!
!  ID   DATE                        REASON
! ...  ........  ..............................................................
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]BTEST_BB.FOR;6

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                  Btest_bb
!
! FILENAME:    Btest_bb.for
!
! FUNCTION:    Btest_bb tests a specific bit in the supplied byte.  If the bit
!              is set Btest_bb returns .TRUE., otherwise it returns .FALSE.
!              The argument corresponding to the bit number is a byte.
!
! COMPILER:    VAX/VMS FORTRAN
!
! INPUT:       bit_num          - Bit number to test
!              target_byte      - Byte in which to test bit
!
! OUTPUT:      Btest_bb         The result of the test is returned
!                               as the function value
!                               .true. ==> bit was set
!                               .false. ==> bit not set
!
! CALLS:       None
!
! DATE:                  June 17, 1986
!
! AUTHOR:      Jonathan P. Anspach
!              EG&G Energy Measurements, Inc.
!              In support of:
!              Sandia National Laboratories
!              Division 7121
!              Albuquerque, NM
!
!
! CHECKER:
!
! ---------------------- R E C O R D   O F   U P D A T E S --------------------
!
!  ID    DATE                        REASON
! ---  --------  --------------------------------------------------------------
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]BTEST_BL.FOR;7

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                  Btest_bl
!
! FILENAME:      UTL_Btest_bl.for
!
! FUNCTION:      Btest_bl tests a specific bit in the supplied byte.  If the bit
!                is set Btest_bl returns .TRUE., otherwise it returns .FALSE.
!        *****   The argument corresponding to the bit number is a longword.
!
! COMPILER:      VAX/VMS FORTRAN
!
! INPUT:         bit_num          - Bit number to test
!                target_byte      - Byte in which to test bit
!
! OUTPUT:        btest_bl         The result of the test is returned
!                                 as the function value.
!                                 .true. ==> bit set
!                                 .false. ==> bit not sset
!
! CALLS:         None
!
! DATE:                  June 17, 1986
!
! AUTHOR:        Jonathan P. Anspach
!                EG&G Energy Measurements, Inc.
!                In support of:
!                Sandia National Laboratories
!                Division 7121
!                Albuquerque, NM
!
!
! CHECKER:
!
! --------------------- R E C O R D   O F   U P D A T E S -------------------
!
!  ID    DATE                       REASON
! ---  --------  -----------------------------------------------------------
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]BTEST_BW.FOR;6

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Btest_bw
!
! FILENAME:     Btest_bw.for
!
! FUNCTION:     Btest_bw tests a specific bit in the supplied byte.  If the bit
!               is set Btest_bw returns .TRUE., otherwise it returns .FALSE.
!       ****    The argument corresponding to the bit number is a word.
!
! COMPILER:     VAX/VMS FORTRAN
!
! INPUT:        bit_num         - Bit number to test
!               target_byte     - Byte in which to test bit
!
! OUTPUT:       btest_bw        The result of the test is returned
!                               as the function value.
!                               .true. ==> bit set
!                               .false. ==> bit not set
!
! CALLS:        None
!
! DATE:                 June 17, 1986
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!               In support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!
! CHECKER:
!
! --------------------- R E C O R D   O F   U P D A T E S -------------------
!
!  ID   DATE                            REASON
! ---  --------  ------------------------------------------------------------
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
  This is file LD:[UTILITY]CANCEL_IO_REQ.FOR;5

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Cancel_IO_Req
!
! FILENAME:     Cancel_IO_Req.for
!
! FUNCTION:     Cancel_IO_Req cancels all I/O requests on a given channel.
!
! COMPILER:     VAX/VMS FORTRAN
!
! INPUT:        io_channel       - I/O channel to use
!
! OUTPUT:       None
!
! CALLS:        Bad_Status       - Reports a bad return status from a system call
!               Sys$cancel      - system service routine
!
! DATE:                 June 24, 1986
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!               In support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!
! CHECKER:
!
! ---------------------- R E C O R D   O F   U P D A T E S ---------------------
!
!  ID    DATE                        REASON
! ---  --------  ------------------------------------------------------------
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]CEILING.FOR;5

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Ceiling
!
! FILENAME:     Ceiling.for
!
! FUNCTION:     Ceiling calculates the ceiling function; that is, it does round-
!               up division.
!
! COMPILER:     VAX/VMS FORTRAN
!
! INPUT:        denominator     - Number to divide into numerator
!               numerator       - Number to divide by denominator
!
! OUTPUT:       The result is returned as the function value
!
! CALLS:        None
!
! DATE:                 September 4, 1986
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!               In support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!
! CHECKER:
!
! --------------------- R E C O R D   O F   U P D A T E S -------------------
!
!  ID    DATE                      REASON
! ---  --------  ------------------------------------------------------------
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]CHANGE_8330.FOR;61

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                    Change_8330
!
! FUNCTION:     This routine changes the format and source (or input) parameters
!               inside the EMR 8330.
!
!               The EMR 8330 processes commands very slowly, so this routine tries
!               to avoid as much parameter changing as it can.  It accomplishes
!               this by getting the current status of the EMR 8330 from the TA667.
!               If the EMR 8330 is already set for the correct format or source
!               Change_8330 leaves that setting alone.
!
! COMPILER:     VAX/VMS FORTRAN
!
! ARGUMENTS:
!
!   Name           Access    Description
!   ----           ------    -----------
!   dev_nam        Read      Device name of the EMR 8330
!   sta_num        Read      Station number of the EMR 8330 (also known as destination)
!   new_fmt        Read      New format to use
!   new_src        Read      New source to use
!   ta667_io_chan Read       I/O channel of the TA667
!
!   change_8330    write     assume either a returned bad status, or error
!                                pertaining to what was found
!
!FUNCTIONS AND SUBROUTINES REFERENCED
!
!  Type  Name                 Type  Name           Type  Name
!
!        GET_8330_STATUS      I*4   LIB$FREE_EF    I*4   LIB$GET_EF
!  I*4   SYS$ASSIGN           I*4   SYS$CANTIM     I*4   SYS$DASSGN
!  I*4   SYS$QIOW             I*4   SYS$SETIMR     I*4   SYS$WAITFR
!
! DATE:                    March 27, 1990
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!                   in support of:
!               Sandia National Laboratories
!               Division 9321
!               Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
  This is file LD:[UTILITY]CHANGE_COMM_SYM.FOR;16

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!c     This subroutine creates or modifies a file whose name is given
!c     by character variable COMM_SYM.  If a file with name
!
!c                              comm_sym//'.DAT'
!
!c     does not exist in the current default directory, the file is
!c     opened with
!
!c                              status = 'NEW'.
!
!c     Otherwise, the file is opened with
!
!c                      status = 'OLD', access = 'APPEND'.
!
!c     In either case, the content of comm_string is written as the last
!c     record of file with the name comm_sym//'.DAT'.  The maximum length
!c     of this record is 30 characters.  If comm_string is longer, the
!c     output record is truncated to 30 characters.  If it is shorter,
!c     the output record is padded to 30 characters.  The usage simplifies
!c     the logic for RTD_SCHEDULER, which is waiting to read the file and
!c     then update its display with the last record of the file.
!
!      character*(*)    comm_string      ! String to be written to file with name
!                                              comm_sym//'.DAT'.
!      character*(*)    comm_sym         ! Name portion of file to be modified.
!
!      integer*4        status           ! 1 ==> Successful completion - usually 1
!                                        ! 0 ==> comm_sym was a blank string
!                                        ! 2 ==> FORTRAN error opening a new file with
!                                             name 'comm_sym'.DAT
!                                        ! 4 ==> FORTRAN error opening an old file with
!                                             name 'comm_sym'.DAT
!                                        ! 6 ==> FORTRAN error writing to the file
!                                        ! even ==> an error code defined in system
!                                             include file '($rmsdef)'
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]CHECK_NUMERIC.FOR;4

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Check_Numeric
!
! FUNCTION:     This routines checks a character string for non-numeric
!               characters.  If the string contains only numeric characters,
!               '123' for example, a zero is returned as the function value.
!               If the string contains any non-numeric characters the position
!               of the first such character is returned as the function value.
!               For example, a string of '123a5b' would return the value 4.
!
! COMPILER:     VAX/VMS FORTRAN
!
! ARGUMENTS:
!
!   Name          Access   Description
!   ----          ------   -----------
!   string        Read     Character string to check for non-numeric characters
!
!   check_numeric write    0 ==> all characters are numeric,
!                      .ne. 0 ==> pointer to first non-numeric element
!
! DATE:                 April 20, 1989
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!                   in support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
  This is file LD:[UTILITY]CHECK_NUMERIC_STR.FOR;5

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Check_Numeric_Str
!
! FUNCTION:     Checks 'string' by using an internal read into a real*8.
!               Failure indicates that the string is not pure numeric.
!
! COMPILER:     VAX/VMS FORTRAN
!
! ARGUMENTS:
!
!   Name          Access    Description
!   ----          ------    -----------
!  string         read      to be checked
!
!  check_numeric_str write  .true. ==> all numeric
!                           .false. ==> not pure numeric
!
! DATE:                 June 27, 1989
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!                   in support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]CLOSE_BIG.FOR;34

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                   Close_Big
!
! FUNCTION:     Close_Big writes the file header out to a big data file and
!               then closes the file.
!
! COMPILER:     VAX/VMS FORTRAN
!
! INPUT:        file_hdr         - File header /BIG_FILE_HDR_DESC/
!               lun              - Logical unit number associated with the file
!
! OUTPUT:       The status of the close operation is returned as the function
!               value.
!
! DATE:                   January 6, 1988; examined 13-mar-1991, PC Kaestner
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!                   In support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
! For additional information see "The BIG Routines User's Manual", and
!               "The TBL and BIG Routine Maintenance Manual"
!
! CHANGES:      Added S_TRIG_DESC to Big_file_hdr_desc, and have added its
!               copy to the big file here.  13-mar-1991, PCK
!
!               11/11/91 added the USE_INDEX variable to the moves from file_hdr
!               to B_FILE_HDR(FNUM)
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]CLOSE_TBL.FOR;14

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                Close_Tbl
!
! FUNCTION:     This routine closes a table (ICF).  For additional information
!               set "The TABLE Routines User's Manual", and "The TBL and BIG
!               Routine Maintenance Manual"
!
! COMPILER:     VAX/VMS FORTRAN
!
! ARGUMENTS:
!
!   Name          Access    Description
!   ----          ------    -----------
!   lun           Read      Logical unit number of table
!
! DATE:                December 22, 1987
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!                   in support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]CNV_BCD_INT.FOR;4

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:              Cnv_Bcd_Int
!
! FILENAME:     UTL_Cnv_Bcd_Int.for
!
! FUNCTION:     Cnv_Bcd_Int converts a value from binary coded decimal (BCD) to
!               an Integer.  The BCD value must be between 0 and 99,999,999;
!               otherwise an error status is returned.
!
! COMPILER:     VAX/VMS FORTRAN
!
! INPUT:        bcd_value       - BCD value to convert to integer
!
! OUTPUT:       int_value       - Variable to receive the converted BCD number
!               A status value is returned that whether or not the BCD value
!               is within the valid range
!
! CALLS:        Ishft           - Shifts a bit string left or right
!
! DATE:              April 25, 1986
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!               In support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!
! CHECKER:
!
! --------------------- R E C O R D   O F   U P D A T E S -------------------
!
!  ID   DATE                        REASON
! ---  --------  -------------------------------------------------------------
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]CNV_CHAR_INT.FOR;7

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Cnv_Char_Int
!
! FILENAME:     UTL_Cnv_Char_Int.for
!
! FUNCTION:     Cnv_Char_Int converts a character string to an integer according
!               to a specified radix and returns the integer as the function
!               value.
!
! COMPILER:     VAX/VMS FORTRAN
!
! INPUT:        radix           - Radix to use in conversion
!               string          - Character string to convert
!
! OUTPUT:       number          - Number that the string is converted to
!
! CALLS:
!FUNCTIONS AND SUBROUTINES REFERENCED
!
! Type  Name          Type  Name          Type  Name          Type  Name
!
! I*4   OTS$CVT_TB_L  I*4   OTS$CVT_TI_L  I*4   OTS$CVT_TO_L  I*4   OTS$CVT_TZ_L
!
!
! DATE:                 March 21, 1986
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!               In support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!
! CHECKER:
!
! --------------------- R E C O R D   O F   U P D A T E S ------------------
!
!  ID    DATE                          REASON
! ---  --------  -----------------------------------------------------------
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]CNV_INT_BCD.FOR;3

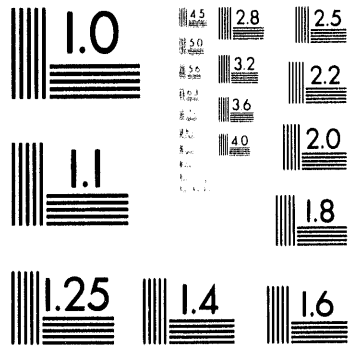
```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Cnv_Int_Bcd
!
! FILENAME:     UTL_Cnv_Int_Bcd.for
!
! FUNCTION:     Cnv_Int_Bcd converts an integer to a binary coded decimal (BCD)
!               value.  The integer must be in the range 0 to 99,999,999;
!               otherwise Cnv_Int_Bcd returns an error status.
!
! COMPILER:     VAX/VMS FORTRAN
!
! INPUT:        int_value       - Integer value convert to BCD
!
! OUTPUT:       bcd_value       - BCD value resulting from the conversion
!
! CALLS:        ishft           - Shifts a bit string left or right
!
! DATE:                 April 25, 1986
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!               In support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!
! CHECKER:
!
! --------------------- R E C O R D   O F   U P D A T E S -------------------
!
!  ID   DATE                    REASON
! ---  --------  ---------------------------------------------------------
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
This is file LD:[UTILITY]CNV_INT_CHAR.FOR;7

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                  Cnv_Int_Char
!
! FILENAME:     UTL_Cnv_Int_Char.for
!
! FUNCTION:     Cnv_Int_Char converts an integer to a character string.
!
! COMPILER:     VAX/VMS FORTRAN
!
! INPUT:        number   integer value to be converted
!               radix    base for the conversion.
!
! OUTPUT:       The character equivalent of the number (base radix) is
!               returned as the function value.
!
! CALLS:
!FUNCTIONS AND SUBROUTINES REFERENCED
!
! Name        Type Name          Type Name
!
! BAD_STATUS  I*4 OTS$CVT_L_TB   I*4 OTS$CVT_L_TI
!
! Type  Name        Type  Name
!
! I*4  OTS$CVT_L_TO I*4  OTS$CVT_L_TZ
!
! DATE:                  October 14, 1986
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!               In support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!
! CHECKER:
!
! ---------------------- R E C O R D   O F   U P D A T E S -------------------
!
!  ID    DATE                        REASON
! ---  --------  ------------------------------------------------------------
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
This is file LD:[UTILITY]CNV_NUM_LST_TBL.FOR;13

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Cnv_Num_Lst_Tbl
!
! FUNCTION:     Cnv_Num_Lst_Tbl converts a numeric list contained in a character
!               string to an array of integers.
!
!               A numeric list consists of a number or list of numbers separated
!               by commas and/or dashes.  A series of numbers separated by
!               commas simply indicates successive numbers, while two
!               numbers separated by a dash indicates an inclusive  range of
!               numbers.  The strings ",-" and "-," are illegal within the
!               numeric list, but a string of the form "1-2-3" is allowed.
!
!               The count of numbers passed back in the array is returned as
!               the function value.
!
! COMPILER:     VAX/VMS FORTRAN
!
! ARGUMENTs:    master_list  Character string to be decoded
!               radix   Number base for the decoding
!               return_array Integer values from string
!               array_size      # values in return_array
!
! FUNCTIONS AND SUBROUTINES REFERENCED
!
! Type  Name         Type  Name        Type  Name
!
! I*4  CNV_CHAR_INT  L*4  FND_CWC_STR  I*4  LIB$INDEX
!
! Type  Name            Type  Name          !
!
! I*4  STR_LENGTH       CHAR SUBSTRING
!
! DATE:                 February 11, 1987
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!                   In support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6

5 of 5

```
  This is file LD:[UTILITY]CNV_STATUS_TBL.FOR;37


!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                Cnv_Status_Tbl
!
! FUNCTION:     Converts SANDUS status bytes to table values.  Does not decode
!               all possible bits.  For further information see "Conversion
!               Routines User's Manual".  The converted values are compared
!               with the ICF, differences are 1) noted, and 2) the status
!               byte value replaces the incorrect value.
!
! COMPILER:     VAX/VMS FORTRAN
!
! FUNCTIONS AND SUBROUTINES REFERENCED
!
!  Type  Name            Type  Name
!
!   I*4  LIB$EXTZV             STR$UPCASE
!
! ARGUMENTS:
!
!   Name          Access    Description
!   ----          ------    -----------
!  status_bytes   read      the 5 SANDUS STATUS BYTES
!  rec_name       read      string indicating mode of SANDUS
!  generic_rec    record    The appropriate SANDUS header record
!  first_ad       write     The # of first A/D, 1 if only 1.
!
! DATE:                January 5, 1989, examined 11 Mar, 1991, PC Kaestner
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!                   in support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
! MODIFIED:     .module_id to .data_mod_type and .module_type to .mem_realtime
!               in preparation for Distant Zenith  11 mar, 1991 PCK
!               also changed .trigger_arm to .trig_arm
!               also changed .samp_rate to .samp_int
!               also changed .int_trig_level to .trig_level
!               also changed .manual_trig to .trig_manual
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
  This is file LD:[UTILITY]CNV_STR_INT.FOR;5


!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                Cnv_Str_Int
!
! FUNCTION:     Cnv_Str_Int converts a character string to an integer according
!               to a specified radix.  If the conversion was successful then
!               a success status code is returned as the function value.  If
!               an error occured during the conversion a failure status code
!               is returned.
!
! COMPILER:     VAX/VMS FORTRAN
!
! INPUT:        radix           - Radix to use in conversion
!               string          - Character string to convert
!
! OUTPUT:       number          - Number that the string is converted to
!
!FUNCTIONS AND SUBROUTINES REFERENCED
!
! Type  Name          Type  Name          Type  Name          Type  Name
!
! I*4   OTS$CVT_TB_L  I*4   OTS$CVT_TI_L  I*4   OTS$CVT_TO_L  I*4   OTS$CVT_TZ_L
!
! DATE:                 January 27, 1988
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!                   In support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]CNV_TBL_STATUS.FOR;52

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:              Cnv_Tbl_Status
!
! FUNCTION:     Convert table entries into their bit pattern in the STATUS
!               bytes which are the first five words of the SANDUS data.
!               For additional information, see "Conversion Routines User's
!               Manual"
!
! COMPILER:     VAX/VMS FORTRAN
!
! ARGUMENTS:    rec_name         string name to indicate SANDUS configuration
!               generic_rec     record SANDUS HDR DESC
!               status_bytes    the returned status bytes.
!
! CHANGES:
!               07-07-89 DJB     If DATABASE contains N/A for trigger mode
!                                this routine will default to EXT A instead
!                                of INTERNAL.
!
! DATE:              January 4, 1989, examined 11 Mar 1991, PC Kaestner
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!                   in support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
! MODIFIED:     changed .module_id to .data_mod_type throughout for DISTANT ZENITH
!               also changed .module_type to .mem_realtime
!               also changed .trigger_arm to .trig_arm
!               also changed .samp_rate to .samp_int
!               also changed .int_trig_level to .trig_level
!               also changed .manual_trig to .trig_manual
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
  This is file LD:[UTILITY]COPY_FILE.FOR;15

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                Copy_File
!
! FUNCTION:     This routine copies a file to a specified destination.  The
!               destination is supplied in the standard VMS file specification
!               format of node::device:[directory]name.type.  Any missing
!               fields in the file specification imply the current default.
!               A flag in the argument list controls whether or not to wait for
!               the file to copy before returning.
!
! COMPILER:     VAX/VMS FORTRAN
!
! ARGUMENTS:
!
!   Name          Access    Description
!   ----          ------    -----------
! file_name       Read      Name of file to copy
! destination     Read      Destination of file
! wait            Read      Flag indicating whether to wait until copy completes
! event_flag      Read      Event flag to set when copy has completed
! final_status    Write     Status of the copy operation
!
! FUNCTIONS AND SUBROUTINES REFERENCED
!
! Type  Name          Name               Type  Name
!
! I*4  LIB$FIND_FILE  LIB$FIND_FILE_END  I*4  LIB$SPAWN
! I*4  STR_LENGTH
!
! DATE:                 November 29, 1989
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!                   in support of:
!               Sandia National Laboratories
!               Division 9321
!               Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
  This is file LD:[UTILITY]CREATE_CAL.FOR;17

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
!       Written by W. G. Perkins
!
!               Version of 05/30/91
!
!       Input arguments:
!
!       integer*4       ace_number      ! AcE Identification number.
!       integer*4       ichnl           ! 7912AD channel number.
!
!       include         'include$include:table_structs.def'
!       record          /t7912_chan_desc/       chan_desc
!
!       Return values (create_cal =):   ! 1 ==> Normal completion
!                                       ! 0 ==> Table value of V_POSITION
!                                       !       is not in an allowed
!                                       !       range.
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

  This is file LD:[UTILITY]CREATE_LOGICAL_NAME.FOR;6

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Create_Logical_Name
!
! FUNCTION:     This routine creates a logical name in the specified logical
!               name table and assigns an equivalence string to it.
!
! COMPILER:     VAX/VMS FORTRAN
!
! ARGUMENTS:
!
!   Name          Access    Description
!   ----          ------    -----------
! logical_name    Read      Logical name to translate
! equivalence_str Read      Equivalence string to assign to the logical name
! table_name      Read      Table in which to insert the logical name
!
! FUNCTIONS AND SUBROUTINES REFERENCED
!
!  Type  Name            Type  Name            Type  Name
!
!        LIB$SIGNAL       I*4  STR_LENGTH       I*4  SYS$CRELNM
!
! DATE:                 November 13, 1989
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!                   in support of:
!               Sandia National Laboratories
!               Division 9321
!               Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]DATEOPEN.FOR;7

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!      Function .:   DATEOPEN
!
!      Purpose. .:   My own user written open routine to
!                    get the dates of interest.
!
!      Parameters:   FAB -- the fab for the file
!                    RAB -- the rab for the file
!                    LUN -- the lun for the file
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]DCL_END.FOR;8

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 DCL_End
!
! FUNCTION:     GET OUT of the program.  An EXIT has been requested by the
!               EXIT handler.  Break all i/o channels and leave.  The status
!               is returned in the function
!
! COMPILER:     VAX/VMS FORTRAN
!
! ARGUMENTS:    None but because its a function, dummy argument required
!
! FUNCTIONS AND SUBROUTINES REFERENCED
!
! Type  Name          Type  Name          Type  Name
!
! I*4  LIB$FREE_EF   I*4  LIB$GET_EF    I*4  STR$ANALYZE_SDESC
! I*4  SYS$CANEXH    I*4  SYS$DASSGN    I*4  SYS$QIOW
!
! DATE:                 April 4, 1990
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!                   in support of:
!               Sandia National Laboratories
!               Division 9321
!               Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]DCL_EXECUTE.FOR;16

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:               DCL_Execute
!
! FUNCTION:     This routine sends a DCL command via a mailbox to a subprocess.
!               The subprocess executes the command and sends any output back
!               via another mailbox.
!
! COMPILER:     VAX/VMS FORTRAN
!
! ARGUMENTS:
!
!   Name               Access    Description
!   ----               ------    -----------
!  commands           read       The commands to be sent, with "\" delimiter
!  max_output_lines   read      Max # output lines expected
!                                        (size of output_lines)
!  output_lines       write     Response from subprocess
!  num_output_lines   write     # lines in 'output_lines'
!
! FUNCTIONS AND SUBROUTINES REFERENCED
!
! Type  Name               Type  Name        Type  Name
!
! I*4  LIB$FREE_EF        I*4  LIB$GET_EF  I*4  LIB$INDEX
! I*4  STR$ANALYZE_SDESC  I*4  STR_LENGTH  I*4  SYS$QIOW
!
!
! DATE:               April 4, 1990
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!                   in support of:
!               Sandia National Laboratories
!               Division 9321
!               Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
  This is file LD:[UTILITY]DCL_EXIT_HANDLER.FOR;7

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 DCL_Exit_Handler
!
! FUNCTION:     Serves as a vehicle to call DCL_EXIT???????????????
!
! COMPILER:     VAX/VMS FORTRAN
!
! ARGUMENTS:
!
!   Name          Access    Description
!   ----          ------    -----------
! exit_status     write     not used
!
! FUNCTIONS AND SUBROUTINES REFERENCED
! Name            Type  Name
!
! BAD_STATUS       I*4  DCL_END
!
!
! DATE:                 April 5, 1990
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!                   in support of:
!               Sandia National Laboratories
!               Division 9321
!               Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
This is file LD:[UTILITY]DCL_INITIALIZE.FOR;10

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 DCL_Initialize
!
! FUNCTION:     This routine prepares a program to issue DCL commands.  It
!               creates input and output mailboxes for a subprocess and spawns
!               the subprocess.
!
! COMPILER:     VAX/VMS FORTRAN
!
! ARGUMENTS:    None, but being a function, a dummy argument is required.
!
! FUNCTIONS AND SUBROUTINES REFERENCED
!
! Type  Name              Type Name         Type  Name
!
!       DCL_EXIT_HANDLER  I*4  LIB$GETJPI   I*4  LIB$SPAWN
! I*4   SYS$CREMBX        I*4  SYS$DCLEXH
!
! DATE:                 April 4, 1990
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!                   in support of:
!               Sandia National Laboratories
!               Division 9321
!               Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]DCL_READ_OUTPUT.FOR;10

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                   DCL_Read_Output
!
! FUNCTION:      This routine reads output from a DCL command that was executed
!                previously using DCL_Execute.  Status is returned via the
!                function name.
!
! COMPILER:      VAX/VMS FORTRAN
!
! ARGUMENTS:
!
!   Name           Access    Description
!   ----           ------    -----------
! output_line      write     line read from subprocess
!
! FUNCTIONS AND SUBROUTINES REFERENCED
!
! Type  Name                Type  Name          Type  Name
!
! I*4  LIB$FREE_EF          I*4  LIB$GET_EF     I*4  LIB$INDEX
! I*4  STR$ANALYZE_SDESC    I*4  SYS$QIOW
!
! DATE:                   April 4, 1990
!
! AUTHOR:        Jonathan P. Anspach
!                EG&G Energy Measurements, Inc.
!                    in support of:
!                Sandia National Laboratories
!                Division 9321
!                Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]DEASSIGN_CHAN.FOR;5

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                  Deassign_Chan
!
! FILENAME:     Deassign_Chan.for
!
! FUNCTION:     Deassign_Chan deassigns an I/O channel.
!
! COMPILER:     VAX/VMS FORTRAN
!
! INPUT:        channel          - Channel number to deassign
!
! OUTPUT:       None
!
! CALLS:        Bad_Status       - Reports a bad return status from a system call
!               Sys$Dassgn       - Deassigns a channel
!
! DATE:                  January 21, 1986
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!               In support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!
! CHECKER:
!
! ---------------------- R E C O R D   O F   U P D A T E S -------------------
!
!  ID    DATE                          REASON
! ---  --------  -------------------------------------------------------------
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
  This is file LD:[UTILITY]DECOMPOSE_FILE_SPEC.FOR;6

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Decompose_File_Spec
!
! FUNCTION:     This routine takes a file specification and breaks it down into
!               a node, device, directory, name, extension and version number.
!
! COMPILER:     VAX/VMS FORTRAN
!
! ARGUMENTS:
!
!   Name          Access    Description
!   ----          ------    -----------
! file_spec       Read      File specification to decompose
! node            Write     Node name
! device          Write     Device name
! directory       Write     Directory name
! name            Write     File name
! extension       Write     File extension
! version         Write     Version number
!
! FUNCTIONS AND SUBROUTINES REFERENCED
!
! Type  Name
!
! I*4   LIB$INDEX
!
! DATE:                 January 16, 1990
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!                   in support of:
!               Sandia National Laboratories
!               Division 9321
!               Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
  This is file LD:[UTILITY]DECOMPOSE_STR.FOR;5

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Decompose_Str
!
! FUNCTION:     This subroutine decomposes a character string into individual
!               tokens.  Tokens are separated by whitespace (blanks or tabs).
!
! COMPILER:     VAX/VMS FORTRAN
!
! ARGUMENTS:
!   Name          Access    Description
!   ----          ------    -----------
!   string        Read      Character string to decompose
!   num_tokens    Read      Number of tokens to return
!   tokens        Write     Array of tokens returned to calling routine
!
! FUNCTIONS AND SUBROUTINES REFERENCED
!
!  Type  Name
!
!  I*4   STR_LENGTH
!
! DATE:                 November 4, 1987
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!                   in support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
  This is file LD:[UTILITY]DELETE_LOGICAL_NAME.FOR;4

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Delete_Logical_Name
!
! FUNCTION:     This routine deletes a logical name from the specified logical
!               name table.
!
! COMPILER:     VAX/VMS FORTRAN
!
! ARGUMENTS:
!
!   Name          Access    Description
!   ----          ------    -----------
! logical_name    Read      Logical name to delete
! table_name      Read      Table from which to delete the logical name
!
! FUNCTIONS AND SUBROUTINES REFERENCED
!
! Name            Type  Name            Type  Name
!
! LIB$SIGNAL       I*4  STR_LENGTH       I*4  SYS$DELLNM
!
! DATE:                 November 13, 1989
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!                   in support of:
!               Sandia National Laboratories
!               Division 9321
!               Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
  This is file LD:[UTILITY]DELETE_TBL_REC.FOR;12

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Delete_Tbl_Rec
!
! FUNCTION:     This routine deletes a record from a table (ICF).  For
!               additional information see "The TABLE Routines User's
!               Manual" and "The TBL and BIG Routines Maintenance Manual".
!
! COMPILER:     VAX/VMS FORTRAN
!
! ARGUMENTS:
!
!   Name          Access    Description
!   ----          ------    -----------
!   lun           Read      Logical unit number of table
!   rec_name      Read      Name of record to delete
!   rec_index     Read      Array index if the record is an array element
!   sub_rec       Read      Sub-record number
!
! FUNCTIONS AND SUBROUTINES REFERENCED
!
! Type  Name            Type  Name
!
! I*4  GET_TBL_INFO     I*4  STR$UPCASE
!
! DATE:                 December 23, 1987
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!                   in support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
  This is file LD:[UTILITY]DISCONNECT_NETWORK_LINK.FOR;9


!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                Disconnect_Network_Link
!
! FUNCTION:     If a network link exists, break it, after sending back the name
!               of the program doing the breaking.  If no link exists, exit.
!
! COMPILER:     VAX/VMS FORTRAN
!
! ARGUMENTS:  None
!
! FUNCTIONS AND SUBROUTINES REFERENCED
!
! Type  Name          Type  Name         Type  Name
!
! I*4  LIB$FREE_EF    I*4  LIB$GET_EF    LIB$PUT_OUTPUT
!      LIB$SIGNAL     I*4  STR_LENGTH    I*4  SYS$CANCEL
! I*4  SYS$CANEXH     I*4  SYS$QIOW
!
! DATE:                 November 1, 1989
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!                   in support of:
!               Sandia National Laboratories
!               Division 9321
!               Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
  This is file LD:[UTILITY]EF_MASK.FOR;6

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                EF_Mask
!
! FUNCTION:     EF_Mask takes an event flag number and an existing mask and
!               returns a new mask with the bit corresponding to the event
!               flag number set.  For example, if event flag number 3 is
!               passed, EF_Mask returns a mask with bit number 3 set in
!               addition to the other bits that were already set in the
!               existing mask.
!
! COMPILER:     VAX/VMS FORTRAN
!
! INPUT:        event_flag      - Event flag number to generate mask for
!               old_mask        - Existing mask to use
!
! OUTPUT:       The new mask is returned as the function value
!
! CALLS:        Ibset           - (Intrinsic) Sets bits in an integer
!
! DATE:                 February 19, 1986
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!               In support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
  This is file LD:[UTILITY]EF_SET.FOR;6

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                EF_Set
!
! FILENAME:     EF_Set.for
!
! FUNCTION:     EF_Set checks the status of an event flag and returns TRUE if
!               the event flag is set and FALSE if the event flag is clear.
!               See also ef_set_c.for
!
! COMPILER:     VAX/VMS FORTRAN
!
! INPUT:        event_flag      - Event flag to test
!
! OUTPUT:       Previous state of event flag returned as the function value
!
! CALLS:        Bad_Status      - Reports a bad return status
!               Sys$Readef      - Reads an event flag and returns the state
!
! DATE:                 January 20, 1986
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!               In support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!
! CHECKER:
!
! --------------------- R E C O R D   O F   U P D A T E S --------------------
!
!  ID    DATE                        REASON
! ---  --------  ------------------------------------------------------------
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]EF_SET_C.FOR;5

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 EF_Set_C
!
! FILENAME:     EF_Set_C.for
!
! FUNCTION:     EF_Set_C checks the status of an event flag and in the process
!               clears it.  If the event flag was set EF_Set_C returns TRUE;
!               otherwise it returns FALSE.  See also ef_set.for
!
! COMPILER:     VAX/VMS FORTRAN
!
! INPUT:        event_flag      - Event flag to test and clear
!
! OUTPUT:       Function value returned
!
! CALLS:        Bad_Status      - Reports a bad return status
!               Sys$Clref       - Clears an event flag and returns the previous
!                                 state
!
! DATE:                 January 20, 1986
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!               In support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!
! CHECKER:
!
! ---------------------- R E C O R D   O F   U P D A T E S -------------------
!
!  ID    DATE                       REASON
! ---  --------  ------------------------------------------------------------
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]FILE_COMPS.FOR;6

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 File_Comps
!
! FILENAME:      UTL_File_Comps.for
!
! FUNCTION:      File_Comps takes a string that contains a file specification and
!                splits it into its component parts.  The components are: node,
!                device, directory, name, type and version.  The components are
!                returned in the argument list as character strings, if not
!                present, a blank string is returned.  Please note that the
!                syntactical elements are present in some strings.
!
! COMPILER:      VAX/VMS FORTRAN
!
! INPUT:         file_spec       input file specification in the form
!                                xx::yy:[zzz]fff.eee;v
!
! OUTPUT:        node            node name (xx::)
!                device          device name (yy:)
!                dir             directory name ([zzz])
!                name            file name  (fff)
!                type            file type, or extension  (.eee)
!                version         version number (;v)
!
! FUNCTIONS AND SUBROUTINES REFFRENCED
!
! Name            Type  Name
!
! BAD_STATUS       I*4  SYS$FILESCAN
!
!
! DATE:                 October 15, 1986
!
! AUTHOR:        Jonathan P. Anspach
!                EG&G Energy Measurements, Inc.
!                In support of:
!                Sandia National Laboratories
!                Division 7121
!                Albuquerque, NM
!
!
! CHECKER:
!
! --------------------- R E C O R D   O F   U P D A T E S ------------------
!
!  ID   DATE                        REASON
! ---  --------  -------------------------------------------------------------
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
  This is file LD:[UTILITY]FILE_DATE.FOR;12

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!c
!c narrative          - Finds either the BACKUP, CREATION, EXPIRATION, or
!c                      REVISION date from the file named in FILENAME, and
!c                      returns that date as an ascii string in ANSWER.
!c
!c author and date    - P. C. Kaestner,9321     - 23 Dec, 1991
!c
!c entry conditions   - FILENAME is an ASCII string which specifies what
!c                              file is to be queried.
!c                      WHICH_DATE is an ASCII string with at least the
!c                              first three letters of the desired date.
!c
!c exit conditions    - ANSWER is an ASCII string with the requested date
!c                              and should be at least CHARACTER*23
!c                      ISTATUS has the same value as FILE_DATE
!c                      FILE_DATE is either SS$_NORMAL, or an error code.
!c
!c                              suggested use is as follows···
!c                      if(.not. file_date("file name","type of date",
!c                              "returned date",istatus) then
!c                              an error occurred, and istatus available
!c                              else
!c                              everything OK
!c                              endif
!c
!c calls              - DATEOPEN
!c
!c called by          - Most mainline programs
!c
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!c
```

This is file LD:[UTILITY]FND_CWC_STR.FOR;3

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Fnd_Cwc_Str
!
! FILENAME:     Fnd_Cwc_Str.for
!
! FUNCTION:     Fnd_Cwc_Str searches a character string for two characters
!               seperated by any amount of whitespace (spaces or tabs).  If the
!               characters are found then .TRUE. is returned as the function
!               value, the position of the first character is returned in the
!               fourth argument and the position of the second character is
!               returned in the fifth argument.  If the characters are not
!               found .FALSE. is returned as the function value.
!
! COMPILER:     VAX/VMS FORTRAN
!
! INPUT:        char1           - First character to search for
!               char2           - Second character to search for
!               string          - String to search
!
! OUTPUT:       char1_pos       - Position of first character
!               char2_pos       - Position of second character
!               .TRUE. or .FALSE. is returned as the function value
!
! FUNCTIONS AND SUBROUTINES REFERENCED
!
! Type  Name            Type  Name
!
! I*4   FND_SUCC_STR    I*4   STR_LENGTH
!
! DATE:                 March 6, 1986
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!               In support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!
! CHECKER:
!
! ---------------------- R E C O R D   O F   U P D A T E S --------------------
!
!  ID   DATE                            REASON
! ---  --------  -------------------------------------------------------------
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]FND_SUCC_STR.FOR;3

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Fnd_Succ_Str
!
! FILENAME:     Fnd_Succ_Str.for
!
! FUNCTION:     Fnd_Succ_Str finds successive substrings in a string.
!
! COMPILER:     VAX/VMS FORTRAN
!
! INPUT:        last_occ        - Position in the string to of the last occurrance
!                                 of the substring
!               string          - String to search for the substring
!               substr          - Substring to search for in the string
!
! OUTPUT:       The position of the substring in the string is returned as the
!               function value.
!
! FUNCTIONS AND SUBROUTINES REFERENCED
!
! Type  Name            Type  Name            Type  Name
!
! I*4   LIB$INDEX       I*4   STR_LENGTH      CHAR SUBSTRING
!
!
! DATE:                 March 7, 1986
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!               In support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!
! CHECKER:
!
! ---------------------- R E C O R D   O F   U P D A T E S --------------------
!
!  ID   DATE                         REASON
! ---  --------  ---------------------------------------------------------------
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]GET_8330_STATUS.FOR;4

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                  Get_8330_Status
!
! FUNCTION:      This routine asks the TA667 for status information on the EMR
!                8330.
!
! COMPILER:      VAX/VMS FORTRAN
!
! ARGUMENTS:
!
!   Name           Access   Description
!   ----           ------   -----------
!   ta667_io_chan  Read     TA667 I/O channel number
!   e8330_status   Write    EMR 8330 status information
!
! FUNCTIONS AND SUBROUTINES REFERENCED
!
! Name           Type  Name           Type  Name
!
! BAD_STATUS      I*4  READ_DRX_1           SET_DRX_FUNC
!
!
! DATE:                  June 22, 1988
!
! AUTHOR:        Jonathan P. Anspach
!                EG&G Energy Measurements, Inc.
!                    in support of:
!                Sandia National Laboratories
!                Division 7121
!                Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
  This is file LD:[UTILITY]GET_ACCNT_NAME.FOR;6

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Get_Accnt_Name
!
! FUNCTION:     This function returns the account name of the user as a
!               character string.
!
! COMPILER:     VAX/VMS FORTRAN
!
! ARGUMENTS:    None, but as a function requires a dummy argument
!
! FUNCTIONS AND SUBROUTINES REFERENCED
!
! Name            Type  Name
!
! BAD_STATUS       I*4  LIB$GETJPI
!
! DATE:                 January 20, 1988
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!                   in support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
  This is file LD:[UTILITY]GET_NODE_NAME.FOR;13

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Get_Node_Name
!
! FILENAME:     Get_Node_Name.for
!
! FUNCTION:     Get_Node_Name calls the operating system to determine the
!               node name of the computer.  The name is returned as the
!               function value.
!
! COMPILER:     VAX/VMS FORTRAN
!
! INPUT:        None
!
! OUTPUT:       The node name is returned as the function value, without
!               misc syntactical elements ( no _ or ::), a blank string
!               if errors.
!
! CALLS:        Bad_Status       - Reports a bad status return from a system call
!               Lib$Sys_Trnlog   - Translates a logical name
!
! DATE:                 April 1, 1987
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!               In support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!
! CHECKER:
!
! --------------------- R E C O R D   O F   U P D A T E S ------------------
!
!  ID    DATE                        REASON
! ---  --------  ------------------------------------------------------------
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
  This is file LD:[UTILITY]GET_PRIVILEGES.FOR;5

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Get_Privileges
!
! FUNCTION:     This routine returns a character string containing the
!               privileges currently authorized.  The string consists of
!               privilege names in uppercase, separated by commas.
!
! COMPILER:     VAX/VMS FORTRAN
!
! ARGUMENTS:    None, but as a function requires a dummy argument.
!
! FUNCTIONS AND SUBROUTINES REFERENCED
!
! Name            Type  Name
!
! BAD_STATUS       I*4  LIB$GETJPI
!
! DATE:                 May 26, 1989
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!                   in support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
  This is file LD:[UTILITY]GET_PROC_MODE.FOR;5

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Get_Proc_Mode
!
! FILENAME:     Get_Proc_Mode.for
!
! FUNCTION:     Get_Proc_Mode calls the operating system to determine the
!               mode of the current process.  The mode type is returned
!               as the function value.
!
! COMPILER:     VAX/VMS FORTRAN
!
! INPUT:        None, but as a function it requires a dummy argument.
!
! OUTPUT:       The process mode is returned as the function value
!
! CALLS:        Bad_Status      - Reports a bad status return from a system call
!               Lib$Getjpi      - Gets job/process information
!               Str_Length      - Returns the length of a string
!
! DATE:                 May 7, 1986
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!               In support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!
! CHECKER:
!
! --------------------- R E C O R D   O F   U P D A T E S --------------------
!
!  ID    DATE                        REASON
! ---  --------  ------------------------------------------------------------
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]GET_TBL_INFO.FOR;18

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:               Get_Tbl_Info
!
! FUNCTION:     This routine returns information about fields and records in
!               a table.  All of the information except the record key is
!               stored in the table itself, in a record described in the
!               Struct_Desc.inc include file.  For addtional information
!               see "The TABLE Routines User's Manual", and "The TBL and BIG
!               Routines Maintenance Manual".
!
! COMPILER:     VAX/VMS FORTRAN
!
! ARGUMENTS:
!
!   Name          Access   Description
!   ----          ------   -----------
!   rec_name      Read     Name of record structure containing the field
!   rec_index     Read     Index into array if the record is an array element
!   sub_rec       Read     Sub-record number
!   rec_key       Write    RMS record key
!   rec_length    Write    Length of record in bytes
!
! FUNCTIONS AND SUBROUTINES REFERENCED
!
! Type  Name
!
! I*4   LSEARCH_C
!
! DATE:               January 19, 1988
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!                   in support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
This is file LD:[UTILITY]GET_TERM_TYPE.FOR;5

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:               Get_Term_Type
!
! FILENAME:     Get_Term_Type.for
!
! FUNCTION:     Get_Term_Type calls the operating system to determine the
!               terminal type of the specified device.  The type is returned
!               as the function value.
!
! COMPILER:     VAX/VMS FORTRAN
!
! INPUT:        device_name     - Name of device to get terminal type for
!
! OUTPUT:       The type of terminal is returned as the function value
!
! CALLS:        Bad_Status      - Reports a bad status return from a system call
!               Lib$Getdvi      - Gets device/volume information
!               Str_Length      - Returns the length of a string
!
! DATE:               April 30, 1986
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!               In support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!
! CHECKER:
!
! --------------------- R E C O R D   O F   U P D A T E S ------------------
!
!  ID   DATE                          REASON
! ---  --------  ----------------------------------------------------------
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]HIBERNATE.FOR;4

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Hibernate
!
! FUNCTION:     This routine schedules a wakeup, then goes into hibernation
!               until the scheduled time.  The time can be an absolute or a
!               delta time.
!
! COMPILER:     VAX/VMS FORTRAN
!
! ARGUMENTS:
!
!   Name            Access   Description
!   ----            ------   -----------
! wakeup_time       Read     Absolute or delta time to wake up
!
! FUNCTIONS AND SUBROUTINES REFERENCED
!
! Name        Type  Name        Type  Name        Type  Name
!
! LIB$SIGNAL  I*4   SYS$BINTIM   I*4  SYS$HIBER   I*4   SYS$SCHDWK
!
! DATE:                 January 4, 1990
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!                   in support of:
!               Sandia National Laboratories
!               Division 9321
!               Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
  This is file LD:[UTILITY]LEFT_JUSTIFY.FOR;5

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Left_Justify
!
! FUNCTION:     Left_Justify removes leading whitespace (blanks and tabs)
!               from a string.
!
! COMPILER:     VAX/VMS FORTRAN
!
! INPUT:        string readonly, original string
!
! OUTPUT:       LEFT_JUSTIFY  write returned string
!
! FUNCTIONS AND SUBROUTINES REFERENCED
!
! Type  Name
!
! I*4  STR$FIND_FIRST_NOT_IN_SET
!
! DATE:                 February 9, 1988
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!                   In support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

  This is file LD:[UTILITY]LOCK_RESOURCE.FOR;8

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!C
!C   This FUNCTION accepts calling program requests for a lock on a resource
!C   identified by the 'RESNAME' argument. If the resource has already
!C   been locked by another user, the calling program may choose to wait
!C   until the resource becomes available by setting the WFLAG equal to
!C   '1'. If the resource is not available, FUNCTION LOCK_RESOURCE will
!C   place the request in a queue and place the caller on hold until it
!C   becomes available. If the caller does not choose to wait, WFLAG  is
!C   set to '0' and LOCK_RESOURCE will respond immediately. If the resource
!C   is available, the lock will be granted (status = 1). If not, the
!C   request will not be placed in the queue, and the lock will not be
!C   granted (status not equal '1'). The lock ID will be returned to
!C   the caller. This parameter must be provided to the UNLOCK_RESOURCE
!C   FUNCTION when the caller releases the resource to other users.
!C
! FUNCTIONS AND SUBROUTINES REFERENCED
!
!  Type  Name            Type  Name
!
!   I*4  SYS$ENQ          I*4  SYS$ENQW
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]LOG_STATUS.FOR;13

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Log_Status
!
! FUNCTION:     Log_Status reports a bad return status from a system call.  The
!               arguments specify traceback information about the calling
!               routine.  It writes output defining the error if possible.
!
! COMPILER:     VAX/VMS FORTRAN
!
! ARGUMENTS:
!
!   Name          Access   Description
!   ----          ------   -----------
! log_lun         read     Logical unit on which to write message
! routine         read     Name of the routine where error detected
! tracer          read     Name of the routine that caused error
! sys_err_code    read     status of error
!
! FUNCTIONS AND SUBROUTINES REFERENCED
!
! Type  Name      Type  Name
!
! I*4  LIB$INDEX  I*4  LIB$SYS_GETMSG
!
! DATE:                 November 2, 1988
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!                   in support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
  This is file LD:[UTILITY]LSEARCH_C.FOR;6

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                Lsearch_c
!
! FUNCTION:      Lsearch conducts a linear search on an array of character
!                strings.  If the target string is found its index is returned
!                as the function value.  If the string is not found a zero is
!                returned.
!
! COMPILER:      VAX/VMS FORTRAN
!
! INPUT:         array    array of character strings
!                size     dimension of array
!                string   target string
!
! OUTPUT:        lsearch_c  0 if target string not found, index into array
!                           if found
!
! DATE:                  October 17, 1986
!
! AUTHOR:        Jonathan P. Anspach
!                EG&G Energy Measurements, Inc.
!                In support of:
!                Sandia National Laboratories
!                Division 7121
!                Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

  This is file LD:[UTILITY]LSEARCH_L.FOR;6

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                Lsearch_l
!
! FUNCTION:      Lsearch conducts a linear search on an array of integer
!                longwords.  If the target integer is found its index is returned
!                as the function value.  If the integer is not found a zero is
!                returned.
!
! COMPILER:      VAX/VMS FORTRAN
!
! INPUT:         ARRAY  array of integer longwords
!                SIZE   # elements in ARRAY
!                NUMBER target value
!
! OUTPUT:        LSEARCH_L either 0 if number not found or index into array
!
! DATE:                  October 20, 1986
!
! AUTHOR:        Jonathan P. Anspach
!                EG&G Energy Measurements, Inc.
!                In support of:
!                Sandia National Laboratories
!                Division 7121
!                Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
  This is file LD:[UTILITY]LSEARCH_Q.FOR;5

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                Lsearch_q
!
! FUNCTION:     Lsearch conducts a linear search on an array of quadwords.
!               If the target quadword is found its index is returned as the
!               function value.  If the quadword is not found a zero is
!               returned.
!
! COMPILER:     VAX/VMS FORTRAN
!
! INPUT:        ARRAY   the array of quad words (r*8)
!               SIZE    the dimension of ARRAY
!               NUMBER   the target quad word
!
! OUTPUT:       LSEARCH_Q  0 if not found, or index into ARRAY
!
! DATE:                 October 20, 1986
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!               In support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

  This is file LD:[UTILITY]LSEARCH_W.FOR;6

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                Lsearch_w
!
! FUNCTION:     Lsearch conducts a linear search on an array of integer words.
!               If the target integer is found its index is returned as the
!               function value.  If the integer is not found a zero is
!               returned.
!
! COMPILER:     VAX/VMS FORTRAN
!
! INPUT:        ARRAY    array of I*2 integer words
!               SIZE     dimension of ARRAY
!               NUMBER   word to be found
!
! OUTPUT:       LSEARCH_W  0 if not found, index into ARRAY if found.
!
! DATE:                 October 20, 1986
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!               In support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]MODIFY_TBL_REC.FOR;37

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Modify_Tbl_Rec
!
! FUNCTION:     This routine modifies a record in a table.  For additional
!               information see "The TABLE Routines User's Manual" and
!               "The TBL and BIG Routine Maintenance Manual".
!
! COMPILER:     VAX/VMS FORTRAN
!
! ARGUMENTS:
!
!   Name          Access    Description
!   ....          ......    ...........
!   lun           Read      Logical unit number of table
!   rec_name      Read      Name of record to be modified
!   rec_index     Read      Array index if the record is an array element
!   sub_rec       Read      Sub-record number
!   byte_array    Read      Replacement record
!
! FUNCTIONS AND SUBROUTINES REFERENCED
!
!  Type  Name            Type  Name
!
!   I*4  GET_TBL_INFO     I*4  STRSUPCASE
!
! DATE:                 December 23, 1987
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!                   in support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]MOVE_FILE.FOR;6

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Move_File
!
! FUNCTION:     This routine copies a file to destination, then deletes the
!               original.  The destination is supplied in the standard VMS file
!               specification format of node::device:[directory]name.type.  Any
!               missing fields in the file specification imply the current
!               default.
!
! COMPILER:     VAX/VMS FORTRAN
!
! ARGUMENTS:
!
!   Name           Access   Description
!   ....           ......   ...........
! file_name        Read     Name of file to move
! destination      Read     Destination of file
! status           Write    Status of move operation
!
! FUNCTIONS AND SUBROUTINES REFERENCED
!
! Name                Type  Name             Type Name
! COPY_FILE           I*4   LIB$DELETE_FILE  I*4  LIB$FIND_FILE
! LIB$FIND_FILE_END
!
! DATE:                 November 30, 1989
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!                   in support of:
!               Sandia National Laboratories
!               Division 9321
!               Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]NETLINK_EXIT_HANDLER.FOR;3

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Netlink_Exit_Handler
!
! FUNCTION:     Tells the other end of the network link that the job is
!               aborting.
!
! COMPILER:     VAX/VMS FORTRAN
!
! ARGUMENTS:
!
!   Name           Access    Description
!   ----           ------    -----------
! exit_status    read       status at time exit handler invoked.
!
! CALLS                 Send_network_msg
!
! DATE:                 November 1, 1989
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!                   in support of:
!               Sandia National Laboratories
!               Division 9321
!               Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]NET_EXECUTE.FOR;7

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Net_Execute
!
! FUNCTION:     This routine passes a string of DCL commands to a command
!               procedure executing on a remote node.  The command procedure
!               is called REM_EXECUTE.COM and must be located in the remote
!               account's default directory and device.
!
! COMPILER:     VAX/VMS FORTRAN
!
! ARGUMENTS:
!
!   Name            Access    Description
!   ----            ------    -----------
! node              Read      Node names
! commands          Read      String of DCL commands, separated by backslashes
! username          Read      Remote account name to log in under
! password          Read      Remote password
! display_output    Read      Switch to indicate whether or not to read the output
!                             of REM_EXECUTE.COM and write it to the terminal
!
! FUNCTIONS AND SUBROUTINES REFERENCED
!
! Type  Name          Name        Type  Name          Type  Name
!
!       FOR$CLOSE     FOR$OPEN    I*4   LIB$FREE_LUN  I*4   LIB$GET_LUN
! I*4   STR_LENGTH    LIB$SIGNAL        STR$UPCASE
!
! DATE:                 January 18, 1990
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!                   in support of:
!               Sandia National Laboratories
!               Division 9321
!               Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]NEXT_WORD_STR.FOR;5

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Next_Word_Str
!
! FILENAME:     Next_Word_Str.for
!
! FUNCTION:     Next_Word_Str returns the next word in a string beginning at
!               the specified character position.  A word is delineated by a
!               space or tab.
!
! COMPILER:     VAX/VMS FORTRAN
!
! INPUT:        STRING          Input string
!               START_POS       starting position within STRING
!
! OUTPUT:       NEXT_WORD_STR   returns the word found
!
! FUNCTIONS AND SUBROUTINES REFERENCED
!
!  Type  Name
!
!  I*4  STR_LENGTH
!
! DATE:                 October 16, 1986
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!               In support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!
! CHECKER:
!
! ---------------------- R E C O R D   O F   U P D A T E S ------------------
!
!  ID    DATE                        REASON
! ---  --------  ------------------------------------------------------------
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]OPEN_BIG.FOR;35

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Open_Big
!
! FUNCTION:     Open_Big opens a big data file for processing.  If the file
!               already exists Open_Big opens it and reads the file header.
!               If the file doesn't exist this routine creates it. For
!               additional information see "The BIG Routines User's Manual",
!               and "The TBL and BIG Routines Maintenance Manual".
!
! COMPILER:     VAX/VMS FORTRAN
!
! INPUT:        directory       - Default directory that the file is in
!               filename        - Name of the file to open
!               file_status     - 'OLD' if the file exists, 'NEW' if it does not
!               new_alloc       - Number of blocks to allocate for a new file
!               access          - Specifies READ or WRITE access
!               lun             - Logical unit number associated with the file
!
! OUTPUT:       file_hdr        - File header
!
!               The status of the open (or create) operation is returned as the
!               function value.
!
! FUNCTIONS AND SUBROUTINES REFERENCED
!
! Type  Name            Name            Type  Name      Name
!
!       FOR$CLOSE       FOR$OPEN        I*4   FOR$RAB   LIB$DATE_TIME
!  I*4  STR_LENGTH      USR_OPEN_BIG
!  I*4  READ_BLOCK_IO   STR$UPCASE
!
! DATE:                 January 6, 1988; examined 13-mar-1991, PC Kaestner
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!                   In support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
This is file LD:[UTILITY]OPEN_TBL.FOR;40

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                Open_Tbl
!
! FUNCTION:     This routine opens a table.  It will not create a new table, so
!               the table must already exist.  It also reads the table structure
!               descriptions from the table and stores them in a common area
!               that is used by Get_Tbl_Info.  For additional information see:
!               "The TABLE Routines User's manual, and "The TBL and BIG Routines
!               Maintenance manual
!
! COMPILER:     VAX/VMS FORTRAN
!
! ARGUMENTS:
!
!   Name          Access    Description
!   ----          ------    -----------
!   filename      Read      Name of table
!   lun           Read      Logical unit number of table
!   access        read      READ or WRITE
!
! FUNCTIONS AND SUBROUTINES REFERENCED
!
! Name            Name
!
! FOR$OPEN        STR$UPCASE
!
! DATE:                December 22, 1987
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!                   in support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]PARSE_LIST.FOR;4

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Parse_List
!
! FUNCTION:     Parse_List parses the command line for the indicated
!               qualifier.  If the qualifier is present globally or locally
!               or defaulted the qualifiers values are obtained and returned
!               in the argument list.
!
! COMPILER:     VAX/VMS FORTRAN
!
! INPUTS:       qualifier       Qualifier to be parsed
!               max_values      Maximum number of qualifier values
!
! OUTPUTS:      values          Qualifier Values
!               num_values      # of qualifier values
!
! FUNCTIONS AND SUBROUTINES REFERENCED
!
!  Type  Name              Type  Name
!
!   I*4  CLI$GET_VALUE     I*4  CLI$PRESENT
!
! DATE:                 November 27, 1989
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!               In support of:
!               Sandia National Laboratories
!               Division 9321
!               Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]PARSE_NUMBER.FOR;5

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Parse_Number
!
! FILENAME:     Parse_Number.for
!
! FUNCTION:     Parse_Number parses the command line for the indicated
!               qualifier and returns the value of the qualifier as a number.
!
! COMPILER:     VAX/VMS FORTRAN
!
! INPUT:        QUALIFIER       Name of the qualifier to look for
!               DEFAULT         Default value if Qualifier missing
!               RADIX           Number base for the conversion
!
! OUTPUT:       VALUE           Returned value
!
! FUNCTIONS AND SUBROUTINES REFERENCED
!
!  Type  Name              Type  Name
!
!   I*4  CNV_CHAR_INT            PARSE_STRING
!
! DATE:                 December 22, 1986
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!               In support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!
! CHECKER:
!
! ---------------------- R E C O R D   O F   U P D A T E S ------------------
!
!  ID   DATE                          REASON
! ---  --------  ------------------------------------------------------------
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]PARSE_NUM_LST.FOR;9

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Parse_Num_Lst
!
! FILENAME:     PF_Parse_Num_Lst.for
!
! FUNCTION:     Parse_Num_Lst parses a numeric list entered on the command
!               line via a command qualifier.
!
!               A numeric list consists of a number or list of numbers separated
!               by commas and/or dashes.  A series of numbers separated by
!               commas simply indicates successive numbers, while two
!               numbers separated by a dash indicates an inclusive  range of
!               numbers.  The strings ",-" and "-," are illegal within the
!               numeric list, but a string of the form "1-2-3" is allowed.
!
!               The count of numbers passed back in the array is returned as
!               the function value.
!
! COMPILER:     VAX/VMS FORTRAN
!
! INPUT:        cmd_qual        The command line qualifier to look for
!               radix           The number base used for the conversion
!
! OUTPUT:       return_array    The array of returned numbers
!               array_size      # elements in RETURN_ARRAY
!
! FUNCTIONS AND SUBROUTINES REFERENCED
!
!  Type  Name              Name
!
!   I*4  CNV_NUM_LST_TBL   PARSE_STRING
!
! DATE:                 February 11, 1987
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!               In support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!
! CHECKER:
!
! ---------------------- R E C O R D   O F   U P D A T E S -------------------
!
!  ID   DATE                        REASON
! ---  --------  --------------------------------------------------------------
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]PARSE_PRESENCE.FOR;11

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Parse_Presence
!
! FUNCTION:     Parse_Presence parses the command line for the indicated
!               qualifier.  If the qualifier is present the value specified
!               in the present argument is returned.  If the qualifier is
!               negated the value specified in the negated argument is returned.
!               If the qualifier is absent the value specified in the absent
!               argument is returned.
!
! COMPILER:     VAX/VMS FORTRAN
!
! INPUT                 qualifier       THE COMMAND LINE QUALIFIER TO LOOK FOR
!               negated         If qualifier negated, return this value
!               absent          If absent return this value
!               present         If present, return this value
!
! OUTPUT        value           Value to be returned
!
! FUNCTIONS AND SUBROUTINES REFERENCED
!
!  Type Name            Name            Name
!
!   I*4 CLI$PRESENT     CLI$_ABSENT     CLI$_DEFAULTED
!       CLI$_LOCNEG     CLI$_LOCPRES    CLI$_NEGATED
!       CLI$_PRESENT
!
! DATE:                 March 18, 1988
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!                   In support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]PARSE_STRING.FOR;5

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Parse_String
!
! FILENAME:     UTL_Parse_String.for
!
! FUNCTION:     Parse_String parses the command line for the indicated
!               qualifier and returns the value of the qualifier as a
!               character string.
!
! COMPILER:     VAX/VMS FORTRAN
!
! INPUT:        QUALIFIER       Command line qualifier to look for
!               DEFAULT         If missing, value to be returned
!
! OUTPUT:       Value           Returned string
!
! FUNCTIONS AND SUBROUTINES REFERENCED
!
!  Type  Name           Type  Name
!
!   I*4  CLI$GET_VALUE   I*4  CLI$PRESENT
!
!
! DATE:                 December 22, 1986
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!               In support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!
! CHECKER:
!
! --------------------- R E C O R D   O F   U P D A T E S -------------------
!
!  ID   DATE                         REASON
! ---  --------  --------------------------------------------------------------
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]PARSE_SWITCH.FOR;6

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Parse_Switch
!
! FUNCTION:     Parse_Switch parses the command line for the indicated
!               qualifier.  If the qualifier is present globally or locally
!               or defaulted the value .true. is returned in the argument list.
!               If the qualifier is negated globally or locally the value false
!               is returned.  If the qualifier is absent from the command line
!               and is not defaulted the argument remains unchanged.
!
! COMPILER:     VAX/VMS FORTRAN
!
! INPUT                 Qualifier       Qualifier to parse for
!
! OUTPUT        Indicator       Returned value
!
! FUNCTIONS AND SUBROUTINES REFERENCED
!
!  Type Name            Name            Name
!
!   I*4 CLI$PRESENT     CLI$_ABSENT     CLI$_DEFAULTED
!       CLI$_LOCNEG     CLI$_LOCPRES    CLI$_NEGATED
!       CLI$_PRESENT
!
! DATE:                 December 22, 1986
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!               In support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]RANDOM_STR.FOR;4

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Random_Str
!
! FUNCTION:     This routine fills a character string with random capital
!               letters.
!
! COMPILER:     VAX/VMS FORTRAN
!
! ARGUMENTS:    None, but being a function requires a dummy argument.
!
! FUNCTIONS AND SUBROUTINES REFERENCED
!
!  Type  Name            Type  Name
!
!   R*4  FOR$SECNDS       R*4  MTH$RANDOM
!
!
! DATE:                 May 9, 1990
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!                   in support of:
!               Sandia National Laboratories
!               Division 9321
!               Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]READ_BIG.FOR;33

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Read_Big
!
! FUNCTION:    Read_Big reads data in from a big data file.  The data for a
!              channel need not all be read in one call to this routine, but
!              may be read using multiple calls (with one side   effect, see
!              note below).  For additional information see, "The BIG
!              Routines User's Manual", and "The TBL and BIG Maintenance
!              Manual".
!
!              NOTE:
!              Read_Big calls Read_Block_IO to read the data in Block I/O
!              mode.  Block I/O reads in complete 512-byte blocks on every
!              read operation.  If the calling routine supplies a buffer that
!              is not a multiple of 512 bytes in size then some data would be
!              lost on subsequent reads if Read_Big serviced the request as
!              is.  For example, if the calling routine requested 600 bytes
!              then Read_Block_IO when called would read two 512-byte blocks
!              (1024 bytes total) and fill the buffer with the   first 600
!              bytes.  However, on the next call Read_Block_IO would read the
!              next two blocks, and the 424 bytes left over from the first call
!              would never be returned.  For that reason Read_Big rounds
!              the size of the read request down to a 512-byte multiple.
!
!              Read_Big returns three arguments besides the data buffer; the
!              number of bytes actually read, the number of bytes left to
!              read for a channel and the starting block number for the read
!              in the Big file.  If the calling routine supplies a buffer
!              that is a multiple of 512 and no errors occur during the read
!              then the number of bytes actually read should be the same as
!              the requested number except.  If the read operation encounters
!              an error the actual number of bytes read in will probably be
!              different from the request.  If the number of bytes requested
!              is greater than the number of bytes left to read then
!              Read_Big will only read what's left.  Therefore, it is normal
!              for the number of bytes read to be different from the requested
!              number.
!
! COMPILER:    VAX/VMS FORTRAN
!
! INPUT:       buffer_size      - Number of bytes to read
!              channel          - Channel number associated with the data
!              lun              - Logical unit number of the xxxBIG file
!              new_chan         - Indicates if we are starting a new channel
!              chan_type        - Channel type byte
!
! OUTPUT:      buffer           - Data buffer read from the xxxBIG file
!              bytes_left       - Number of bytes left to read for a channel
!              bytes_read       - Actual number of bytes read
!              start_block      - Starting block number for the read
!
!              The status of the read operation is returned as the function
!              value.
!
! FUNCTIONS AND SUBROUTINES REFERENCED
!
!  Type  Name
!
!   I*4  READ_BLOCK_IO
```

```
!
! DATE:                January 7, 1988
!
! AUTHOR:       Jonathan P. Anapach
!               EG&G Energy Measurements, Inc.
!                   In support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
! CHANGES:      1. March 11,1991 - Added the starting block number of the
!               current read to the list of output parameters.
!               2. 11/11/91 by PCK added RTD byte count stuff for new HDR_DESC
!               3. 05/21/92 by PCK modified for Hunters Trophy
!               4. 05/06/93 by PCK modified to include Lecroys
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]READ_BIG_HDR.FOR;20

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Read_Big_Hdr
!
! FUNCTION:     Read_Big_Hdr reads the channel header of a big data file.
!
! COMPILER:     VAX/VMS FORTRAN
!
! INPUT:        channel         - Channel number associated with the header
!               lun             - Logical unit number of the xxxBIG file
!               chan_type       - Channel type byte
!
! OUTPUT:       header          - Channel header returned to calling routine
!
!               The status of the copy operation is returned as the function
!               value.
!
! FUNCTIONS AND SUBROUTINES REFERENCED
!
!  Type  Name
!
!   I*4  READ_BLOCK_IO
!
! DATE:                 January 7, 1988
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!                   In support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
! modified 5/21/92 for Hunters Trophy PCK
! modified 5/6/93 for LeCroy 7200 PCK
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]READ_BLOCK_IO.FOR;6

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Read_Block_IO
!
! FUNCTION:
!
! COMPILER:     VAX/VMS FORTRAN
!
! INPUT:        rab             - Record, Record Activation Block (RAB)
!               buffer_size     _ Dimension of BUFFER in bytes
!
! OUTPUT:       buffer          _ I*2 buffer for data
!               block_num       _ Starting block number
!               bytes_read      _ # bytes actually read into BUFFER
!
! FUNCTIONS AND SUBROUTINES REFERENCED
!
!  Type  Name
!
!   I*4  SYS$READ
!
! DATE:                 August 12, 1986
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!               In support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]READ_DRX_1.FOR;3

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Read_DRx_1
!
! FUNCTION:     This routine sends data to a device through a DRx, then reads
!               the response from the device back through the DRx into a single
!               buffer.  The status of the read is returned as the function value.
!
!               The DRx requires 4 overhead bytes at the front of any buffer
!               used with it, so the data actually starts at byte 5.
!
! COMPILER:     VAX/VMS FORTRAN
!
! ARGUMENTS:
!
!   Name           Access    Description
!   ----           ------    -----------
!   io_channel     Read      I/O channel of the DRx
!   send_buffer    Read      Buffer of data to send to the device
!   send_size      Read      Size of send buffer in bytes
!   send_timeout   Read      Number of seconds to wait for the send to complete
!   recv_buffer    Write     Buffer to receive data from the device
!   recv_size      Read      Size of receive buffer
!   recv_timeout   Read      Number of seconds to wait for the receive to complete
!   recv_count     Write     Number of bytes received from the device
!
! FUNCTIONS AND SUBROUTINES REFERENCED
!
!  Type  Name            Type  Name            Type  Name
!
!   I*4  LIB$FREE_EF      I*4  LIB$GET_EF       I*4  SYS$QIOW
!
! DATE:                 February 29, 1988
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!                   in support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
  This is file LD:[UTILITY]READ_FORMAT.FOR;111

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Read_Format
!
! FUNCTION:     This routine reads a stream format file in order to obtain the
!               following information about the stream:
!
!                       Frame length in bytes
!                       Maximum subcommutation depth of the data
!                       Channel numbers found in the stream
!
! COMPILER:     VAX/VMS FORTRAN
!
! ARGUMENTS:
!
!   Name          Access    Description
!   ----          ------    -----------
!   filename      Read      Name of file
!   max_frame_length
!                 Read      Maximum frame length
!   max_subcom_depth
!                 Read      Maximum subcommutation depth
!   format        Write     Channel format
!   frame_length  Write     Actual length of frame in bytes
!   subcom_depth  Write     Actual subcommutation depth
!
! FUNCTIONS AND SUBROUTINES REFERENCED
!
! Type  Name          Type  Name           Name        Name
!
! I*4   CEILING        I*4  CNV_CHAR_INT   FOR$CLOSE   FOR$OPEN
! I*4   LIB$FREE_LUN   I*4  LIB$GET_LUN
!
! DATE:                 December 22, 1988
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!                   In support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]READ_PASSWORD.FOR;10

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                  Read_Password
!
! FUNCTION:     This routine prompts for and reads a password.  The password is
!               not echoed on the screen.  It is converted to upper case and
!               returned as the function value.
!
! COMPILER:     VAX/VMS FORTRAN
!
! ARGUMENTS:
!
!   Name          Access    Description
!   ----          ------    -----------
!   prompt        Read      Prompt string to write to the terminal
!
! FUNCTIONS AND SUBROUTINES REFERENCED
!
!  Type  Name           Type   Name            type  Name
!
!   I*4  ASSIGN_CHAN           BAD_STATUS            DEASSIGN_CHAN
!   I*4  SYS$QIOW       I*4    LIB$FREE_EF     I*4   LIB$GET_EF
!   I*4  STR_LENGTH
!
! DATE:                  May 2, 1989
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!                   in support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
  This is file LD:[UTILITY]READ_TBL_REC.FOR;28

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Read_Tbl_Rec.for
!
! FUNCTION:     This routine reads a record from a table.  For additional
!               information see, "The TBL Routines User's Manual", and "The
!               TBL and BIG Routines Maintenance Manual".
!
! COMPILER:     VAX/VMS FORTRAN
!
! ARGUMENTS:
!
!   Name          Access    Description
!   ----          ------    -----------
!   lun           Read      Logical unit number of table
!   rec_name      Read      Name of record structure to read
!   rec_index     Read      Array index if the record is an array element
!   sub_rec       Read      Sub-record number
!   byte_array    Write     Return argument for the record
!
! FUNCTIONS AND SUBROUTINES REFERENCED
!
!  Type  Name             Type  Name
!
!   I*4  GET_TBL_INFO      I*4  STR$UPCASE
!
! DATE:                 December 22, 1987
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!                   in support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
  This is file LD:[UTILITY]REQUEST_NETWORK_LINK.FOR;19

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Request_Network_Link
!
! FUNCTION:     Open a channel for Task to Task communications.
!
! COMPILER:     VAX/VMS FORTRAN
!
! ARGUMENTS:
!
!   Name          Access    Description
!   ----          ------    -----------
!  program        read      name of the program to open channel with
!   onitor_string read      message to be sent to that program
!
! FUNCTIONS AND SUBROUTINES REFERENCED
!
!  Type  Name               Type  Name           Type  Name
!
!   I*4  LIB$ASN_WTH_MBX     I*4  LIB$FREE_EF     I*4  LIB$GET_EF
!        LIB$PUT_OUTPUT           LIB$SIGNAL           NETLINK_EXIT_HANDLER
!        SEPARATE_STR             STR$UPCASE      I*4  STR_LENGTH
!   I*4  SYS$DCLEXH           I*4  SYS$QIOW
!
! DATE:                 October 19, 1989
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!                   in support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
  This is file LD:[UTILITY]SEND_NETWORK_MSG.FOR;17

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Send_Network_Msg
!
! FUNCTION:
!
! COMPILER:     VAX/VMS FORTRAN
!
! ARGUMENTS:
!
!   Name          Access    Description
!   ----          ------    -----------
! status_param    read      string with descriptive info
! optional_param  read      string with specific info
!
! FUNCTIONS AND SUBROUTINES REFERENCED
!
! Type  Name               Type  Name          Type  Name
!
! I*4   LIB$ANALYZE_SDESC  I*4   LIB$FREE_EF   I*4   LIB$GET_EF
!       LIB$SIGNAL         I*4   STR_LENGTH     I*4   SYS$QIO
!
! DATE:                 October 23, 1989
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!                   in support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
  This is file LD:[UTILITY]SEPARATE_STR.FOR;6

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                Separate_Str
!
! FUNCTION:     This subroutine decomposes a character string into substrings.
!
! COMPILER:     VAX/VMS FORTRAN
!
! ARGUMENTS:    string          - READ -String to be examined
!               delimiter       - READ -sub-string separator
!               max_substrs     - READ -# number, dimension of substrs
!               substrs         - WRITE -array of substrs, note that this
!                                       should be able to take the longest
!                                       possible substring expected.
!               num_substrs     - WRITE -# of subrstrs found
!
! FUNCTIONS AND SUBROUTINES REFERENCED
!
!  Type  Name
!
!   I*4  STR_LENGTH
!
! DATE:                 October 31, 1989
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!                   in support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]SET_DRX_FUNC.FOR;3

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Set_DRx_Func
!
! FUNCTION:     This routine sets the function bits for a DRx.
!
! COMPILER:     VAX/VMS FORTRAN
!
! ARGUMENTS:
!
!   Name          Access   Description
!   ----          ------   -----------
!   io_channel    Read     DRx I/O channel number
!   function_mask Read     Mask to use in setting the function bits
!
! FUNCTIONS AND SUBROUTINES REFERENCED
!
! Name        Type  Name        Type  Name       Type  Name
!
! BAD_STATUS  I*4   LIB$FREE_EF I*4   LIB$GET_EF I*4   SYS$QIOW
!
! DATE:                 February 29, 1988
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!                   in support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]STRIP_FILE_NAME.FOR;5

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
!     Written by W. G. Perkins
!             Version of 01/25/91
!
!     This Character function strips any node or directory information
!     and any file extension off the argument FILE_NAME and passes the
!     stripped name back to the caller.
!
!     INPUT/OUTPUT VARIABLE
!
!     CHARACTER*(*)    FILE_NAME
!
! FUNCTIONS AND SUBROUTINES REFERENCED
!
!  Type  Name
!
!   I*4  LIB$INDEX
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]STR_LENGTH.FOR;5

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Str_Length
!
! FILENAME:     Str_Length.for
!
! FUNCTION:     Str_Length.for finds the unpadded length of a character
!               string and returns it as the function value.  The unpadded
!               length is not necessarily the same as the length returned by
!               the built-in Len function.
!
! COMPILER:     VAX/VMS FORTRAN
!
! INPUT:        string          - Character string to find the length of
!
! OUTPUT:       Length returned as the function value
!
! CALLS:        Char            - Converts an integer to a character
!
! DATE:                 February 18, 1986
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!               In support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!
! CHECKER:
!
! --------------------- R E C O R D   O F   U P D A T E S ------------------
!
!  ID    DATE                          REASON
! ---  --------  ---------------------------------------------------------
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]SUBSTRING.FOR;3

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                    Substring
!
! FILENAME:     Substring.for
!
! FUNCTION:     Substring returns a substring of a character string.  Its main
!               reason for existence is to do subscript checking so that any
!               subscript values can be used to specify the substring without
!               the program blowing up.
!
! COMPILER:     VAX/VMS FORTRAN
!
! INPUT:        end             - Ending position in string of substring
!               start           - Starting position in string of substring
!               string          - Character string to get substring from
!
! OUTPUT:       The substring is returned as the function value.
!
! CALLS:        Len             - Returns the length of a string
!
! DATE:                    March 4, 1986
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!               In support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!
! CHECKER:
!
! ----------------------- R E C O R D   O F   U P D A T E S --------------------
!
!  ID   DATE                          REASON
! ---  --------  ------------------------------------------------------------
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
  This is file LD:[UTILITY]TABLE_MODIFIED.FOR;26


!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
!C      Written by W. G. Perkins
!C              Version of 01/25/91
!C
!C      This integer function first tests to see if it is running on
!C      one of the nodes ACEnn, RMVJO, or SANmmm.  If it's not running
!C      on one of these nodes, it returns status value 0.  No further
!C      action occurs.
!C
!C      The code uses system service SYS$FILESCAN to parse FILE_SPEC for
!C      the file name, which is stored in FILE_NAME.  It then checks
!C      directory TABLE_DIR for a file named FILE_NAME.TBL (or named
!C      FILE_NAME.DAT if a SANDUS or SANDACE).  If there is no file
!C      named FILE_NAME.TBL (FILE_NAME.DAT if a SANDUS or SANDACE), the
!C      routine returns status value 4.  No further action occurs.
!C
!C      If FILE_NAME.TBL (FILE_NAME.DAT if a SANDUS or SANDACE) exists, the
!C      routine uses run-time library routine LIB$DELETE_SYMBOL to delete
!C      any local symbol with name FILE_NAME.  Any error return from
!C      LIB$DELETE_SYMBOL is passed back to the caller.
!C
!C      If deletion of any local symbol was successful, the code uses the
!C      run-time library routine LIB$SET_SYMBOL to create the global symbol
!C      FILE_NAME with value FILE_NAME.  An error return from LIB$SET_SYMBOL
!C      is passed back to the caller.  If the symbol creation is successful,
!C      the code returns status value 1 to the caller.
!C
!C      Error status values returned from various LIB$ calls that are
!C      passed back to the calling routine can be interpreted by the
!C      caller using the BAD_STATUS utility.
!C
!C      In general, an odd return value indicates success, and an even
!C      return value indicates some kind of failure.
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]TBL_CHAN_DIR.FOR;16

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Tbl_Chan_Dir
!
! FUNCTION:     Tbl_Chan_Dir provides a directory from a table of the channels
!               that match a specified channel type.  The directory is in the
!               form of an integer array that contains channel numbers in the
!               order they appear in the table.  The total number of channels
!               in the table is also returned through an argument.  For
!               additional information see, "The TABLE Routines User's Manual",
!               and "The TBL and BIG Routines Maintenance Guide".
!
! COMPILER:     VAX/VMS FORTRAN
!
! ARGUMENTS:
!  lun          read    Logical unit number for the desired ICF
!  chan_type    read    Type of hannel to look for
!  dir          write   Array of channels to be returned
!  dir_size     write   dimension of DIR
!  num_chans    write   actual # of channels returned
!
! FUNCTIONS AND SUBROUTINES REFERENCED
!
!  Type  Name
!
!   I*4  READ_TBL_REC
!
! DATE:                 January 15, 1988
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!                   In support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
  This is file LD:[UTILITY]TRANSLATE_LOGICAL_NAME.FOR;9

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Translate_Logical_Name
!
! FUNCTION:     This routine translates a logical name and returns the
!               equivalence string as the function variable.
!
! COMPILER:     VAX/VMS FORTRAN
!
! ARGUMENTS:
!
!   Name          Access    Description
!   ....          ......    ...........
! logical_name    Read      Logical name to translate
! table_name      Read      Table to search for the logical name
!
! FUNCTIONS AND SUBROUTINES REFERENCED
!
!  Name            Type  Name            Type  Name
!
!  LIB$SIGNAL       I*4  STR_LENGTH       I*4  SYS$TRNLNM
!
! DATE:                 November 13, 1989
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!                   in support of:
!               Sandia National Laboratories
!               Division 9321
!               Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]TYPE_LENGTH.FOR;9

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Type_Length
!
! FUNCTION:     This subroutine takes a data type specification and returns the
!               length (in bytes) of the data type.  The length is returned as
!               both a character string and as an integer.
!
! COMPILER:     VAX/VMS FORTRAN
!
! ARGUMENTS:
!   Name          Access   Description
!   ....          ......   ...........
!   type          Read     Data type to use
!   len_str       Write    Length of the data type as a character string
!   len           Write    Length of the data type as an integer
!
! FUNCTIONS AND SUBROUTINES REFERENCED
!
!  Type  Name            Type  Name         Name
!
!   I*4  CNV_CHAR_INT     I*4  LIB$INDEX    STR$UPCASE
!
! DATE:                 November 4, 1987
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!                   in support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
  This is file LD:[UTILITY]UNLOCK_RESOURCE.FOR;7


!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!C
!C
!C  This FUNCTION processes caller requests to release to other users
!C   a resource identified by the ID parameter. The ID parameter was
!C   provided to the caller when the lock was requested from the FUNC-
!C   TION LOCK_RESOURCE. If more than one lock was requested by the
!C   caller, it is the caller's responsibility to provide the correct
!C   ID for the resource being released.
!C
! FUNCTIONS AND SUBROUTINES REFERENCED
!
!  Type  Name
!
!   I*4  SYS$DEQ
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!


  This is file LD:[UTILITY]USR_OPEN_BIG.FOR;13


!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Usr_Open_Big
!
! FUNCTION:     Usr_Open_Big is a useropen routine to open a big data
!               file.  It sets the file to use block I/O.
!
! COMPILER:     VAX/VMS FORTRAN
!
! ARGUMENTS:    fab     File access Block
!               rab     Record access block
!               lun     Logical unit number
!
! FUNCTIONS AND SUBROUTINES REFERENCED
!
! Type  Name        Type  Name          Type  Name
!
! I*4  FOR$JIBSET    I*4  SYS$CONNECT    I*4  SYS$CREATE
! I*4  SYS$OPEN
!
! DATE:                 August 4, 1986
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!                   In support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]WAIT_EF_C.FOR;5

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Wait_EF_c
!
! FILENAME:     Wait_EF_c.for
!
! FUNCTION:     Wait_EF_c waits for an event flag to be set, then clears the
!               flag and returns control to the calling routine.
!
! COMPILER:     VAX/VMS FORTRAN
!
! INPUT:        event_flag      - Event flag number to use
!
! OUTPUT:       None
!
! CALLS:        Bad_Status      - Reports a bad return status from a system call
!               Sys$Clref       - Clears an event flag
!               Sys$Waitfr      - Waits for a single event flag
!
! DATE:                 May 30, 1986
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!               In support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!
! CHECKER:
!
! ---------------------- R E C O R D   O F   U P D A T E S -------------------
!
!  ID    DATE                           REASON
! ---  --------   ------------------------------------------------------------
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
  This is file LD:[UTILITY]WRITE_BIG.FOR;26

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                Write_Big
!
! FUNCTION:     This routine writes data out to a big data file.  If this is the
!               first call for a particular channel it writes the channel header
!               before writing any data.  Write_Big also updates the
!               start_block and byte_count fields in the file header.  The data
!               for a channel need not all be written out in one call, but may
!               be split into multiple buffers and sent to Write_Big with
!               multiple calls (with one restriction, see note below).  For
!               additional information see, "The BIG Routines user's manual",
!               and "The TBL and BIG Routines Maintenance Manual".
!
!               NOTE:
!               Write_Big calls Write_Block_IO to write the data in Block I/O
!               mode.  Block I/O writes out ,   'ete 512-byte blocks on every
!               write operation.  If the buffe      contains 200 bytes of data
!               then Write_Block_IO will write      ?00 bytes of data
!               followed by 312 bytes of garbage.  .r that reason the number of
!               bytes in the input buffer to this routine should always be a
!               multiple of 512, unless it is the last buffer for the channel.
!
! COMPILER:     VAX/VMS FORTRAN
!
! INPUT:        buffer          - Data buffer to write to the big file
!               buffe _size     - Size of buffer in bytes
!               channel         - Channel number associated with the data
!               chan_hdr        - Channel header
!               lun             - Logical unit number of the big file
!               new_chan        - Indicates if we are starting a new channel
!               chan_type       - Channel type
!
! OUTPUT:       bytes_written   - Number of bytes actually written to the file
!
!               The status of the write operation is returned as the function
!               value.
!
! FUNCTIONS AND SUBROUTINES REFERENCED
!
!  Type  Name
!
!   I*4  WRITE_BLOCK_IO
!
! DATE:                 January 7, 1988
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!                   In support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
! MODIFIED:     11/11/91 by PCK to allow for the new RTD header desc.
!               05/21/92 by PCK to prepare for HUnters Trophy, and delete
!                       LeCroy 6880 references
!                       05/06/93 by PCK to add LeCroy 7200s
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]WRITE_BIG_2.FOR;8

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:               Write_Big_2
!
! FUNCTION:     This routine writes data out to a big data file.  Write_Big_2
!               also updates the start_block and byte_count fields in the file
!               header.  The data for a channel need not all be written out in
!               one call, but may be split into multiple buffers and sent to
!               Write_Big_2 with multiple calls (with one restriction, see
!               note below).   For additional information see, "The BIG
!               Routines User's Manual", and "The TBL and BIG Routines
!               Maintenance Manual".
!
!               NOTE:
!               Write_Big_2 calls Write_Block_IO to write the data in Block I/O
!               mode.  Block I/O writes out complete 512-byte blocks on every
!               write operation.  If the buffer only contains 200 bytes of data
!               then Write_Block_IO will write out the 200 bytes of data
!               followed by 312 bytes of garbage.  For that reason the number of
!               bytes in the input buffer to this routine should always be a
!               multiple of 512, unless it is the last buffer for the channel.
!
! COMPILER:     VAX/VMS FORTRAN
!
! INPUT:        lun             read    logical unit number of the BIG file
!               channel         read    Channel # associated with the data
!               new_chan        read    Indicates if we are starting a new chan
!               buffer          read    Buffer to write to the BIG file
!               buffer_size     read    Size of BUFFER
!
! OUTPUT:       bytes_written   write   # bytes actually write to the BIG file
!
! FUNCTIONS AND SUBROUTINES REFERENCED
!
!  Type  Name
!
!   I*4  WRITE_BLOCK_IO
!
! DATE:               January 7, 1988
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!                   In support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]WRITE_BIG_HDR.FOR;15

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Write_Big_Hdr
!
! FUNCTION:     Writes the BIG file header, described in
!               INCLUDE$INCLUDE:BIG_STRUCTS.DEF.  For additional information
!               see, "The BIG routines User's Manual", and "The TBL and BIG
!               Routines Maintenance Manual".
!
! COMPILER:     VAX/VMS FORTRAN
!
! INPUT:        lun             read    BIG file logical unit number
!               channel         read    Channel number
!               chan_type       read    Type of channel
!               chan_hdr        read    a record passed as a byte array
!
! OUTPUT:       The function name returns the status of the operation
!
! FUNCTIONS AND SUBROUTINES REFERENCED
!
!  Type  Name
!
!   I*4  WRITE_BLOCK_IO
!
! DATE:                 January 7, 1988
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!                   In support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
! MODIFIED:     11/11/91 by PCK to add the RTD header desc stuff
!               05/21/92 by PCK to prepare for Hunters Trophy and
!                       delete LeCroy 6880 references
!               05/06/93 by PCK added LeCroy 7200
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This is file LD:[UTILITY]WRITE_BLOCK_IO.FOR;6

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                 Write_Block_IO
!
! FUNCTION:
!
! COMPILER:     VAX/VMS FORTRAN
!
! INPUT:        rab             read    Record access block
!               buffer_size     read    Size of BUFFER in bytes
!               buffer          read    The byte array to write
!
! OUTPUT:       block_num       write   Where to start writing
!               bytes_written   write   # bytes written
!
! FUNCTIONS AND SUBROUTINES REFERENCED
!
!  Type  Name
!
!   I*4  SYS$WRITE
!
! DATE:                 August 12, 1986
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!               In support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
  This is file LD:[UTILITY]WRITE_FORTRAN_ERR.FOR;6

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
! NAME:                Write_FORTRAN_Err
!
! FUNCTION:     This routine writes a text message that corresponds to a
!               supplied FORTRAN run-time error number.  These messages are
!               from Chapter 5 of the Fortran User's Manual.
!
!
! COMPILER:     VAX/VMS FORTRAN
!
! ARGUMENTS:
!
!   Name          Access    Description
!   ----          ------    -----------
!   error_num     Read      FORTRAN error number
!   lun           Read      Logical unit number to write to
!
! DATE:                December 22, 1987
!
! AUTHOR:       Jonathan P. Anspach
!               EG&G Energy Measurements, Inc.
!                  in support of:
!               Sandia National Laboratories
!               Division 7121
!               Albuquerque, NM
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

Appendix I

FILE READ ROUTINES

APPENDIX I
FILE READ ROUTINES

I.1 INTRODUCTION

Any programmer writing a program that uses or creates files will want to know what is in the file being created or used. Does it have the correct information? Is it in the correct format? To address these problems, and because the files created or used by the Nevada Test Site (NTS) Instrumentation System are nicely packaged with records and structures, three programs TABLEREAD, BIGREAD, and CHNREAD were written. These programs download the three basic kinds of files used at NTS, the Instrument Control Files (ICF) formerly called TABLES, the BIG file, and the CHN file respectively. These files are described in Appendices A, C, and D respectively.

The source for all three programs are in the directory REPAIR$INCLUDE. They depend heavily on the subprograms that dump the various records used in the NTS Instrumentation System. These subprograms are in this same directory and are in the library REPAIR$LIBRARY:DUMP.OLB, where the two logicals are defined:

$ DEFINE REPAIR$INCLUDE LD:[REPAIR] (or wherever sources are maintained)

and

$ DEFINE REPAIR$LIBRARY LD:[REPAIR] (or wherever the library is maintained).

The glue that makes the various xxxREAD routines work is the ability of the program DUMPPREP to create an "almost ready to compile" subprogram for a supplied structure. It removes most of the drudgery of creating the I/O list, and the corresponding FORMAT. Some information on that program is included in Section I.5.

I.2 TABLEREAD

TABLEREAD is used to verify the values in an ICF. It determines the kind of ICF and its internal structure. It is useful to help experimenters confirm that their latest changes were (or were not) in the most recent edition of the ICF. TABLEREAD is a place to look to find out how to use the ICF Utility Routines, Appendix B.

TABLEREAD was given its name when there were table files, which are now called Instrument Control Files (ICF). ICFs were originally called tables, but this caused too much confusion with INGRES database tables. TABLEREAD runs interactively; it allows the user to view the general records in an ICF as well as the header records for a few channels. It also allows the user to create an ASCII file dump of an entire ICF (BE AWARE THAT THE FILE CREATED CAN BE LARGE). ICFs are described in Appendix A.

I.2.1 Using TABLEREAD

TABLEREAD is invoked as follows:

```
$ RUN LD:[REPAIR]TABLEREAD.
```

If the user is going to use the program frequently, then it is necessary to define a symbol in LOGIN.COM such as:

```
$ TREAD == "RUN LD:[REPAIR]TABLEREAD"
```

to make it easy to use.

TABLEREAD generates two output files, TABLE.LIS, which is an ASCII file duplicating the terminal output, and TABLEREAD.LOG, which is an execution and error log file. The use of the word "table" should be understood to mean ICF. TABLEREAD does not modify the ICF, indeed the ICF is OPENed READONLY. There are no "privileged" users such as exist in CHNREAD.

TABLEREAD first requests the path and ICF name. The ICF name is then examined to see if it is a simulation ICF. It is opened READONLY using OPEN_TBL, and the "hidden" record STRUCT_DESC is dumped.

If the file contains simulation information, it is dumped to both the screen and TABLE.LIS. The program then closes the ICF and terminates.

If the file is not a SIMWORK ICF, TABLEREAD has significantly more work to do. It dumps to the screen and to TABLE.LIS all records with a record type less than or equal to 19. This will be all noninstrument oriented records.

Then the user is asked if he wants to dump all channels or individual channels. If all channels are chosen, the remainder of the file is dumped, but only to TABLE.LIS. If individual channels are desired, the channel number is requested, in octal if the ICF is a SANDUS or ACE ICF, in decimal otherwise. When this is finished, the program closes the files, and exits. When the ICF is for RTDs, the user must be aware that there is both a device and a channel. A request for a dump of a channel will get the device description record and all the channel and subchannel description records. A request for a dump of all channels will result in the device record appearing once for the device itself and once for each channel on that device.

1.2.2 TABLEREAD Maintenance Aids

TABLEREAD.COM will compile and link two executable versions of TABLEREAD, TABLEREAD.EXE, and TABLE_DEBUG.EXE. It requires both DUMP.OLB, and UTILITY.OLB. UTILITY.OLB is maintained in UTILITY$INCLUDE, which is defined as:

$ DEFINE UTILITY$INCLUDE LD:[UTILITY] (or wherever the library is maintained)

The source code for TABLEREAD is in TABLEREAD.FOR; it consists of the following subprograms:

(1) TABLEREAD - The main program where most of the logic occurs. Called routines are fairly simple and small.

(2) CLOSE_LUNS - Closes the logfile.

(3) SETUP - Opens a named file on the logical unit supplied.

(4) GET_TABLE_NAME - Gets the name of the ICF and determines if it is a simulation ICF.

(5) INQUIRE_CHANNEL_MODE - Determines whether to dump individual channels or all channels.

(6) GET_CHANNEL_INFO - Determines the type of channel and dumps all possible records for that channel. This is a nontrivial program.

(7) FOUND - Writes execution information and errors to TABLEREAD.LOG.

I.3 BIGREAD

BIGREAD runs interactively and allows the user to view the contents of a BIG file at his terminal, and in more detail from a file that captures the entire session. A BIG file is described elsewhere in the Appendix C.

BIGREAD is used to verify the elements of a channel header going into ANALYZE, or PROCESS. It is used infrequently to look at a channel's data. Because of its less frequent use, as compared with CHNREAD, it is not as fully developed as the other file reading routines. It is a good place to look for examples of how to use the BIG file utility routines described in Appendix E.

I.3.1 Using BIGREAD

BIGREAD is invoked as follows:

```
$ RUN LD:[REPAIR]BIGREAD.
```

If the user creates a symbol in LOGIN.COM such as:

$ BREAD == "RUN LD:[REPAIR]BIGREAD",

then it will always be easy to use when needed.

BIGREAD requests a full name of the BIG file to be examined, and then asks whether the user wishes to look at single channels within the file or at the file as a whole. There is currently only one option, and that is to look at single channels. The ability to examine the file as a whole has not been implemented. If this is a problem, consider the use of the VAX system utility DUMP.

After the required "Y" response, a selected part of the information in the BIG_FILE_HDR_DESC record is shown, including the channel number/primary type pairs for all channels in this BIG file. When a valid channel number is given, the contents of the record /xxx_HDR_DESC/ is dumped, where "xxx" is the digitizer name. It is important to use the screen lock key here, because this record is dumped without pause. Remember that a copy of this session is being created in BIGREAD.LOG, an ASCII file.

The user is next asked whether to dump channel data. "Y" or "N" are the allowed responses. If the response is "N", then a new channel number is requested. If you response is "Y", then the format of the data dump must be entered. Choices here are BYTE or WORD. Either choice will then present the user with a lower and upper record number bound based on the number of data bytes for this channel. The user may ask to see any record in this inclusive range, and to look at records in this range in any order, as often as desired. Exit by asking for an invalid record number (0 is a good choice).

The user may examine as many valid channels as desired and, at any point, may start working with a new BIG file.

I.3.2 BIGREAD Maintenance Aids

BIGLIB.COM is a command procedure that will compile a source program and insert/replace its object in the library file BIGREAD.OLB.

BIGREAD.COM is a command procedure that uses BIGLIB.COM and then creates the two executable files BIGREAD.EXE, and BIGDBG.EXE (a debug version of BIGREAD.EXE). It requires one parameter, the name of the source file (no extension please), or the word NONE, indicating that only the link is to occur.

BIG_COMPILE_ALL.COM is a command procedure that totally recreates the executable files. It deletes the existing BIGREAD.OLB and creates a new one. It then compile all files of the form *BIG*.FOR, inserting the object into the new library, and finally does the links. It is useful when a major change has occurred.

The source code is in REPAIR$LIBRARY, and all file names have BIG in them. Note that the file names may have BIG in them, but the subprograms contained therein may not.

(1) BIGREAD - This is the main program, serving primarily as a driver calling the various modules as required. It does contain most of the necessary logic.

(2) FOUND - Writes messages to the file BIGREAD.LOG about the session, with the date/time stamp and the creation date of the executable used. Other messages appear as required, interspersed with the record dumps.

(3) FILE_RECORD - Is currently a do nothing routine, intended to dump the BIG file as a file. I have never implemented it, but the idea is there. The source file is named BIG_FILE_RECORD.FOR

(4) PAUSE - Is used when dumping data to allow the user to scan the screen before it moves on. Hit any key to continue. The source file is named BIG_PAUSE.FOR.

(5) RDR - Is a function that reads 512 bytes (a VAX page) from a file. It is used when dumping data blocks. The source file is named BIG_RDR.FOR.

(6) DISPLAY_CHAN_DATA - Determines whether to display the channel data as I*2 words or bytes and call the appropriate subprogram. The source file is DISPLAY_BIG_CHAN_DATA.FOR.

(7) DISPLAY_CHAN_HEADER - Determines the file type and calls the subprograms necessary to dump the header. The source file is DISPLAY_BIG_CHAN_HEADER.FOR.

(8) DISPLAY_OCTAL_BYTE - Dumps 512 byte blocks in octal byte format. The source file is DISPLAY_BIG_OCTAL.FOR.

(9) DISPLAY_OCTAL_WORD - Dumps 512 byte blocks in octal word format. The source file is DISPLAY_BIG_OCTAL.FOR.

(10) INQUIRE_INTENT - Inquires whether the user wants to look at the BIG file as a whole (currently not allowed) or at single channels within the file.

(11) WHICH_BIGFILE - Gets the BIG file's full filename from the user. The source file is WHICH_BIG_FILE.FOR.

(12) WRITE_CHANNELS - Puts the channel number/primary channel type pairs on the screen as a set of valid choices. The source file is WRITE_BIG_CHANNELS.FOR.

(13) CHOSE_CHANNEL - Prompts the user for the channel to be dumped. The source file is CHOSE_BIG_CHANNEL.FOR.

(14) WRITE_BIG_HEADER - Writes to the screen a synopsis of the information in the BIG file header. The complete record is dumped into the file BIGREAD.LOG.

I.4 CHNREAD

CHNREAD runs interactively and allows the user to view the contents of a CHN file at his terminal, and in more detail from a file that captures the entire session. A CHN file is described in Appendix D.

CHNREAD creates two output files, CHNREAD.LOG, in which information on the progress of the session is maintained, and READOUT.LIS, which contains all the information requested by the user. Some few people are "privileged" users, that is, their usernames are known to CHNREAD. The program will allow "privileged" users to change CHN files, provided they own the file being changed.

I.4.1 Using CHNREAD

CHNREAD is invoked as follows:

```
$ RUN LD:[REPAIR]CHNREAD.
```

If the user creates a symbol in LOGIN.COM such as:

```
$ HREAD == "RUN LD:[REPAIR]CHNREAD",
```

then it will always be easy to use when needed.

CHNREAD requests the directory_name and then the filename. Frequently, the user will want to look at several CHN files from the same directory.

If, and only if, the user is a "privileged" user, the program will ask whether to open the file READONLY. The normal response is "Y". If however, the user intends to modify the file, it is necessary to answer "N". Modifications are made into the existing file, not to a new file, hence CHNREAD will only allow users who have WRITE access to the current directory and permission to write the CHN file to work with the file. Always make modifications to a copy of the original, not to the original file. CHNREAD will change the variable CHN_DATE to the current date-time if any modifications are made.

The users is then asked about each record in the order that they appear in the header. A "Y" response will cause the record to be dumped to the screen and to the file READOUT.LIS. Any other response will bypass that record.

At this point the user is then presented with a menu with a number of choices:

(1) Examine the data records in random order.
(2) Add a word to a data record. (Requires special privilege)
(3) Delete a word from a data record. (Requires special privilege)
(4) Modify a word in a data record. (Requires special privilege)
(5) Search for a pattern in a sequence of records.
(8) Modify the channel header. (Not seen, and requires special privilege)
(9) Terminate the use of this file (and go get another one).
(Quit) Exit this program.

Note: Options 6 and 7 currently are not active or available.

Having read the CHN file header, CHNREAD computes the number of data blocks in the record. Option 1 allows the user to look at any record within that range as often as necessary, and in any order. To exit from this option, the user enters a record number outside the range allowed (0 is a good value to use).

Options 2,3, and 4 require that the user be a "privileged" user, one known to CHNREAD. Options 2,3, and 4, all use the mechanics of option 1; that is, the user finds the record to be modified using the logic of option 1, and, when it is found, exits with an out-of-range record number. With the desired record in core, the user proceeds with the change. Note that because the records are exactly one block long, adding or deleting a word causes changes to be made in all subsequent records. Also note that since a 7912 record is in multiples of 512 words, CHNREAD will not allow additions or deletions in 7912 records.

Option 5 conducts a search through a range of records for a given pattern. It finds all occurrences of the pattern and gives the record number and byte position in that record of the occurrence and the number of words since the last occurrence or the beginning of the search.

If the user is a privileged user, Option 8, an inplace replaces, is available, but is not shown as an option. It requires a specialized knowledge of the CHN header structures. The routines that make the modifications know about the structure and have a dictionary of allowed changes. It is easy to add a new structure element to the dictionary. Significant changes in a structure may require mo. ications in this area of the CHNREAD source.

Option 9 allows the user to go back for another CHN file, where "BACK" allows the entry of a new pathname.

QUIT does just that.

The user can look for a specific item on the terminal screen, but the file READOUT.LIS is always created and will permit browsing, using an editor.

I.4.2 CHNREAD Maintenance Aids

CHNLIB.COM will compile and insert into CHNREAD.OLB the source file named as the parameter.

CHNREAD.COM requires one parameter, either the name of the module to be compiled or the word "NONE", requesting only a link. If a module is named as the parameter, that module is compiled and added to CHNREAD.OLB. In either case, a link is attempted. CHNREAD.COM uses CHNLIB.COM to compile the module named as the parameter. PRELEWD.OLB, DUMP.OLB, and UTILITY.OLB must be supplied as libraries. The two executable files created are CHNREAD.EXE and CHNDBG.EXE.

The source code for CHNREAD is contained in two modules, CHNREAD.FOR and MODIFY.FOR. CHNREAD.FOR contains the following subprograms:

(1) CHNREAD - This is the main program, serves primarily as a driver, calling the various needed subprograms.

(2) CHANGE_HEADER - This is the driver subprogram for CHN file header modification. It checks on the validity of the user, and then determines the type of CHN file. Makes all the calls to the appropriate modify routines.

(3) CRUSH_WORD - Removes one word from the nth position in an array, then moves the remaining words down one position leaving the last word empty.

(4) DECIDE - Writes the user options to the screen and waits for a valid answer.

(5) DELETE_WORD - Deletes one word from a record, then modifies all remaining records, the internal word count, and possibly the internal number of records, to reflect this change.

(6) DISPLAY_OCTAL_WORD - Writes an octal dump of a record to the screen, and to READOUT.LIS.

(7) EXPAND_WORD - Starts in the nth position of an array and moves all remaining words one position higher, thus opening a hole at that position.

(8) INSERT_WORD - Uses EXPAND_WORD to insert a word into a record, then modifies all remaining records, the internal word count, and possibly the internal record count to reflect this change.

(9) PATSRCH - Looks for a pattern in a range of data records.

(10) PUSH_WORD - Puts an INTEGER*2 word in the nth position of an array.

(11) READ_DATA - Reads random access data records from a file.

(12) DELETE - Delete the nth word/byte from an array, using DELETE_WORD or DELETE_BYTE.

(13) INSERT - Inserts a word/byte in the nth position of an array using INSERT_WORD or INSERT_BYTE.

(14) PUSH - Replaces a word/byte in the nth position of an array using PUSH_WORD or PUSH_BYTE.

(15) HEADER_DISPLAY - Bootstraps the CHN header into memory and determines the number of data blocks in the record.

(16) ONEARG - A generic call to the DUMP_* subprograms.

(17) TWOARG - A generic call to the DUMPI_* subprograms.

(18) DUMP_IT_ALL - Supervisor for the CHN file header dumps.

(19) WRR - Writes a record into a random access file.

(20) OPER - Determines the user's name and, if the file requested exists, opens it.

(21) WATCH_IT - If the file was modified, the variable CHN_DATE is modified to tell the user that the file has been changed.

(22) OBTAIN - Determines the name of the file the user wants to examine.

(23) FOUND - Logs information and errors to the LOG file.

(24) CHECK_USER - Checks the user's name against a list of "privileged" users allowed to modify CHN files.

The source file MODIFY.FOR contains the following subprograms (to better understand the name part of MOD_name, please see the file CHN_HEADER.DEF):

(1) MOD_GC - Modifies the GENERAL_CHAN structure.

(2) MOD_G - Modifies the GEN_DESC structure.

(3) FETCH_CHAR_VAR - Obtains a new character value to replace an existing header record character value.

(4) FETCH_INT_VAR - Obtains a new integer value to replace an existing header record integer value.

(5) FETCH_REAL4_VAR - Obtains a new REAL*4 value to replace an existing header record real value.

(6) FETCH_REAL8_VAR - Obtains a new REAL*8 value to replace an existing header record real value.

(7) FETCH_LOG_VAR - Obtains a new logical value to replace an existing header record logical value.

(8) MOD_SA - Modifies the S_ALOG_CHAN_DESC structure.

(9) MOD_SAC - Modifies the CAL_S_ALOG structure.

(10) MOD_T - Modifies the T7912_CHAN_DESC structure.

(11) MOD_TC - Modifies the CAL_7912 structure.

(12) MOD_C - Modifies the T7103_CHAN_DESC structure.

(13) MOD_CC - Modifies the CAL_T7103 structure.

(14) MOD_TS - Modifies the T7912_SUB_DESC structure.

(15) MOD_CS - Modifies the T7912_SUB_DESC structure.

(16) MOD_SDS - Modifies the S_DIG_SUB_DESC structure.

(17) MOD_R - Modifies the RTD_DEV_DESC structure.

(18) MOD_RC - Modifies the RTD_CHAN_DESC structure.

(19) MOD_RES - Modifies the RTD_EXP_SUB_DESC structure.

(20) MOD_RLS - Modifies the RTD_LC_SUB_DESC structure.

(21) MOD_RCS - Modifies the RTD_CC_SUB_DESC structure.

(22) MOD_R_CAL - Modifies the CAL_RTD structure.

(23) MOD_L - Modifies the LEC_DEV_DESC structure.

(24) MOD_LC - Modifies the LEC_CHAN_DESC structure.

(25) MOD_LES - Modifies the LEC_EXP_SUB_DESC structure.

(26) MOD_LLS - Modifies the LEC_LC_SUB_DESC structure.

(27) MOD_LCS - Modifies the LEC_CC_SUB_DESC structure.

(28) MOD_L_CAL - Modifies the CAL_LEC structure.

I.5 DUMPPREP

To create the DUMP... subprograms for all of the possible records, DUMPPREP reads a user specified structure description file (they usually have a .DEF extension), and generates a FORTRAN source code subprogram to write the contents of each "STRUCTURE...END STRUCTURE" pair found.

DUMPPREP is one source module DUMPPREP.FOR that, when compiled, should be LINKed with the UTILITY.OLB. The main subprogram, DECOMPOSE_LINE, has uses in other areas, see CREATE_TBL.FOR in LD:[TOOLS].

DUMPPREP will only accept files with a very specific format; it produces source code that the user needs to examine carefully. It marks areas of possible trouble with "******" starting in column 1. The format is specified in the beginning of the structure file TABLE_STRUCTS.DEF.

Consider the following two structures, which conform to the rules described in TABLE_STRUCTS.DEF

```
structure /test_sub1/
        integer*4 a
        real*4 b
        character*8 c
        logical*4 d
        integer*4 aa(3)
```

```
            real*4 bb(3)
            character*8 cc(3)
            logical*4 dd(3)
            integer*4 aaa(3,3)
            real*4 bbb(3,3)
            character*8 ccc(3,3)
            logical*4 ddd(3,3,3)
        end structure


        structure /test/
            integer*2 e
            byte   f
            byte g
            integer*4 %fill
            real*4 dummy
            record /test_sub1 / t1
        end structure
```

When run against DUMPPREP, they produce the following output:

```
      subroutine dumpi_TEST_SUB1 (index,iunit,str)
      include 'include$include:big_structs.def/nolist'
      include 'include$include:chn_header.def/nolist'
      record / TEST_SUB1 /str
      write(iunit,9000) index,str.last_edit
      Write(iunit,9001)
     1        STR.A,
     2        STR.B,
     3        STR.C,
     4        STR.D,
*****5        (STR.AA(i),i=1,3),
*****6        (STR.BB(i),i=1,3),
*****7        (STR.CC(i,
*****8        (STR.DD(i),i=1,3),
*****9        (STR.AAA(i),i=1,3,3)
      write(iunit,9002)
*****1        (STR.BBB(i),i=1,3,3),
*****2        (STR.CCC(i),,
*****3        (STR.DDD(i),i=1,3,3,3)
      return
9000  format(' This is a dump of the structure / TEST_SUB1 /' /
```

```
     1    ' with index ',i2,' which was last edited on ',a,'.) 9001  format(
     1       ' A                          ',I8/
     2       ' B                          ',1PE15.8/
     3       ' C                          ',A/
     4       ' D                          ',L8/
*****5       ' AA(i),i=1,3)               ',I8/
*****6       ' BB(i),i=1,3)               ',1PE15.8/
*****7       ' CC(i                       ',A/
*****8       ' DD(i),i=1,3)               ',L8/
*****9       ' AAA(i),i=1,3,3)            ',I8/)
9002  format(
*****1       ' BBB(i),i=1,3,3)            ',1PE15.8/
*****2       ' CCC(i),                    ',A/
*****3       ' DDD(i),i=1,3,3,3)          ',L8/)
      end
```

and

```
      subroutine dump_TEST (iunit,str)
      include 'include$include:big_structs.def/nolist'
      include 'include$include:chn_header.def/nolist'
      record / TEST /str
      write(iunit,9000) str.last_edit
      write(iunit,9001)
     1       STR.E,
     2       STR.F,
     3       STR.G,
     4       STR.DUMMY,
*****5       TEST_SUB
      return
9000  format(' This is a dump of the structure / TEST /' /
     1       ' which was last edited on ',a,'. ')
9001  format(
     1       ' E                          ',I8/
     2       ' F                          ',I8/
     3       ' G                          ',I8/
     4       ' DUMMY                      ',1PE15.8/
*****5       ' SUB                        'CALL DUMP_)
      end
```

In both subprograms the record's structure is assumed to be defined in either TABLE_STRUCTS.DEF, or CHN_HEADER.DEF. If this is not the case, then the list of "include" files must be modified. DUMPPREP assumes that there is a variable "LAST_EDIT", which may not be true, especially for structures defined in BIG_STRUCTS.DEF.

DISTRIBUTION:

1 Field Command
Defense Nuclear Agency
Attn: FCTT (George Lu)
Kirtland AFB
Albuquerque, NM 87117-5000

8 Defense Nuclear Agency
Attn: FCTTO (Kiran Shah)
6400 Mitchell St
North Las Vegas, NV 89031

1 Allied Signal
Attn: Frank Stepaniak
6400 Mitchell St
North Las Vegas, NV 89031

5 7141 Technical Library
1 7151 Technical Publications
10 7613-2 Document Processing
for DOE/OSTI
1 8523-2 Central Technical Files
1 [illegible] M. J. Navratil
1 [illegible] L. D. Bishop
1 [illegible] C. D. Boyer
1 [illegible] B. D. Ades
1 [illegible] B. D. Baker
8 [illegible] D. D. Gaudell
1 [illegible] T. L. Harvey
1 [illegible] D. D. French
1 [illegible] D. J. Isidoro
1 [illegible] P. O. Kaestner
1 [illegible] D. L. Kinchen
1 [illegible] J. W. Lee
1 [illegible] L. Livingston
1 [illegible] P. P. Mafer
1 [illegible] D. A. Mills
1 [illegible] J. L. Romine
1 [illegible] C. W. Cook
1 [illegible] C. L. Ogle
1 [illegible] D. G. Browne
1 [illegible] P. L. Nelson
1 [illegible] D. J. Bolding
1 [illegible] W. J. Kluecher
1 [illegible] P. W. Raymond
1 [illegible] B. W. Marsh ([illegible])
1 [illegible] B. J. Hohlbrandt ([illegible])

In the first subprogram, an 'indexed' subprogram was requested (note the dumpI in the subprogram name). This means that there is some index value associated with the record, and that index becomes the first argument. The second argument is the output logical unit number, and the third argument is the record itself. Note that the format is very strange. Except for singly dimensioned variables, the subscripts are wrong, and for all arrays the format is incorrect.

In the second subprogram, the 'index' argument is not present, and the I on the subprogram name is missing. The '%FILL' element in the record is missing from the variable list in the source. Any records declared within the structure must be fixed with a call to an dump subprogram.