WL-TR-96-2015

ADVANCED TURBINE AEROTHERMAL RESEARCH RIG (ATARR) DATA ACQUISITION SYSTEM OVERVIEW AND OPERATOR'S GUIDE

C. Haldeman M. Dunn
B. Meyer J. Moselle

Calspan Corp
Advanced Technology Center
PO Box 400
Buffalo NY 14225

SEPTEMBER 1995

FINAL

AERO PROPULSION & POWER DIRECTORATE
WRIGHT LABORATORY
AIR FORCE MATERIEL COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-7251

[illegible]

CHRISTIAN E. RANDELL, Lt, USAF
Turbine Design Engineer
Turbine Engine Division

Charles D. MacArthur

CHARLES D. MacARTHUR
Chief, Turbine Branch
Turbine Engine Divsion

[illegible]

RICHARD J. HILL
Chief of Technology
Turbine Engine Division
Aero Propulsion & Power Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE Sep 95	3. REPORT TYPE AND DATES COVERED Final

4. TITLE AND SUBTITLE Advanced Turbine Aerothermal Research Rig (ATARR) Data Acquisition System Overview and Operator's Guide	5. FUNDING NUMBERS C: F33615-88-C-2825 PE: 62203F PR: 3066 TA: 06 WU: 84
6. AUTHOR(S) C. Haldeman M. Dunn B. Meyer J. Moselle	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Calspan Corp Advanced Technology Ctr PO Box 400 Buffalo NY 14225	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Aero Propulsion & Power Directorate Wright Laboratory Air Force Materiel Command Wright Patterson AFB OH 45433-7251 POC: Lt Christian Randell, WL/POTT, 513-255-3150	10. SPONSORING/MONITORING AGENCY REPORT NUMBER WL-TR-96-2015
11. SUPPLEMENTARY NOTES	
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This document is one of a series prepared by Calspan Corporation for WPAFB WL/POTC that describes the construction and operation of the ATARR facility. This document addresses the issues surrounding the design and operation of the Data Acquisition System (DAS). The heart of the ATARR facility is the DAS. This system provides the engineers the ability to translate the voltage changes in different instruments into known, reliable engineering units. The DAS provides a context in which detailed technical questions can be answered through analysis of the basic instrumentation measurements. DAS controls all aspects of data acquisition from set-up through reduction; and, it is critical to understand the philosophy of how the system is constructed in order to best utilize its many capabilities. In a production facility, the computer system has a great deal of regimentation built into it. The test operator would not be allowed to run a test unless certain types of instruments were installed, and the number and procedures of the data reduction are limited to standard ones. Adding new reduction routines or test operations requires major changes to the computer system since there are many built-in checks to protect the test engineer from making flagrant mistakes.

14. SUBJECT TERMS Data Acquisition			15. NUMBER OF PAGES 198
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT SAR

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102

Table of Contents

List of Commonly Used Acronyms

DAS (Data Acquisition System)
DRP (Data Reduction Process)
DAS (Data Acquisition System)
TSP (Test Setup Process)
FCP (Fill and Calibration Process)
DAP (Data Acquisition Process)
A/D (Analog to Digital signal conversions)
RCDR (Run/Channel Descriptor Records
E.U. (Engineering Units)
HRF (Hardware Resources File)
SCF (Sensor Calibration File)
GUI (Graphical User Interface)

The ATARR Data Acquisition System (DAS)

This document is one of a series prepared by Calspan Corporation for the WPAFB/POTC that describes both the construction and the operation of the ATARR facility. This particular document addresses the issues surrounding the design and operation of the Data Acquisition System (DAS). The heart of the ATARR facility is the DAS since it is this system which provides the engineers the ability to translate the voltage changes in different instruments into known, reliable engineering units. In addition the DAS provides a context in which detailed technical questions can be answered through analysis of the basic instrumentation measurements. The DAS controls all aspects of data acquisition from set-up through reduction, and thus, it is critical to understand the philosophy of how the system is constructed in order to best utilize its' many capabilities. There have been many opposing forces pulling on the DAS design since the critical design review. One notable example is the need of a production facility (easy and fast) versus the need of a research facility (flexible). In a production facility, the computer system has a great deal of regimentation built into it. The test operator would not be allowed to run a test unless certain types of instruments were installed, and the number and procedures of the data reduction are limited to standard ones. Adding new reduction routines or test operations requires major changes to the computer system since there are many built in checks to protect the test engineer from making flagrant mistakes.

In a research facility, the computer system is far more general, requiring more attention from the test engineer. While mistakes which could be disastrous in nature can be checked by the computer, the test engineer can not be completely saved from his own inattention. This allows much more flexibility in the data reduction system and allows the DAS to be used for data acquisition on non-turbine testing (such as calibrations) without any change to the overall computer structure. Calspan has created a system which can both provide some protection against inattention on the part of the test engineer, a great deal of flexibility, and protects the engineer from creating a "garbage in-garbage out" situation by forcing the engineering to make key decisions, instead of allowing a system default which may not be appropriate in all test conditions. To accomplish this Calspan had to relay on some intrinsic checks (such as gauge naming conventions), external checks of engineer input (such as in the historical calibration data files), and a variety of input-output formats to provide complete flexibility and experimentation with data reduction schemes. It is important to re-emphasize that while the DAS system as configured resembles a "turn-key" product: one that is capable of being turned on and used immediately; one will not be able to fully use the DAS resources unless one understands the entire system in some detail.

This document has been split into three separate sections. Section I combines an overview of the entire DAS system with the operating philosophy of the DAS and some of the constraints involved in its original design. Section II consists of operating notes which describe both the programs and more complicated technical issues such as calibration and the different data reduction schemes. The final section contains some of the more basic information about the file naming conventions, file organization and some of the basic FORTRAN subroutines.

It is important to note that most of the information contained in this manual has been sent to WPAFB previously. This manual represents an attempt to organize that information in a more coherent manner that is easily updated. Since the DAS system was designed to be expanded, it only makes sense to design the documentation to be expanded easily as well.

The executive summary in the first section of this report describes in a cursory manner the overall capabilities of the system. The intended audience is an outside manager or engineer who may want to review the capabilities of the system to obtain some insight into whether a new type of experiment could be run and supported within the DAS. This section describes both hardware and software components of the DAS. At this time the hardware has been changed a great deal since Calspan left ATARR and is probably out-of-date already.

The remainder of the first section addresses two specific areas. The first is the decision issues and constraints that Calspan considered when the DAS system was designed and built. This information is important both for historical reasons, but also because of the results of those discussions. The second area involves those parts of the DAS system which probably need to be in all of the versions of the DAS system to make it part of one continuos family. While the DAS system has undergone two major revisions (and undoubtedly more in the future) there are some parts which will probably be used in all the revisions. These are contained in this section.

The second section discusses various technical procedures and describes the program used in the present version of the DAS system. This is the part of the manual which is very version oriented. It is set-up as a set of stand-alone notes. These notes can discuss a variety of topics and will deal with all issues involving a topic (hardware, software, storage, etc.). In this manner, the system is very easy to document and update. If a particular item needs to be better documented, or a specific piece of software needed, it can easily be added to this section with very little effect on the entire document.

The final section is actually the main part of what has become the version 1 report. This report has a great deal of information about the file naming convention, gauge labels and some of the basic FORTRAN subroutines. While this report does not discuss the overall aspects of how the system runs, it is included because of the wealth of individual detail that it contains.

As a result, only the executive summary in the first section will need to be changed if there are major changes to the hardware or software capabilities. The rest of the first section should be relatively stable. The second section will be the one which is changed as capabilities are added over time. However, the format of that section is such that maintaining the documentation should be relatively simple. The final section is basically submitted as a reference for items such as the naming convention and file notation. This should not have to be changed in the future.

Section 1. DAS Summary and Overview

1.1 Executive Summary

The ATARR Data Acquisition System (DAS) comprised of the instrumentation front-ends, the amplifier-digitizer components, the software suite, and calibration techniques; satisfy the dual needs of the facility: the flexibility required to be a world-class research tool, and the timeliness required for a production test facility. Some of the major cost-growth incurred during the construction of ATARR was incurred in developing the software and instrumentation to the level where ATARR can be a totally self-sufficient facility from calibration to data reduction at a level of accuracy far greater than attempted before and far beyond the original scope of the project. The rewards have been considerable. Successes have included the deciphering of firmware problems in the Keithley 195A, and problems in thermal drift in an analog switch in the amplifiers, neither of which were known by the manufacturers. The key was not in any one of the areas listed above, but rather in how all four areas: hardware, calibration equipment, software, and techniques are incorporated to yield high accuracy data. This overview is designed to provide a quick summary of the existing DAS configuration and the software capabilities.

1.1.1 DAS Hardware

The DAS hardware has undergone many additions since Calspan turned over the ATARR facility to the Air Force, thus this overview will be general. The DAS hardware is composed of three main sub-components: the amplifier/digitizers, the instrumentation front-ends, and the calibration equipment. The amplifiers are DSP 1402E's which have programmable gains from 1 to 5000, have various filter settings (Butterworth, Bessel, 4 pole, eight pole), filter frequencies, and offset capability which is equal to ±full amplifier range. All parameters can be remotely set, and are programmed from the host computer (a Sun workstation) over a GPIB (General Purpose Instrument Bus, the standardized version of the HP-IB). The DAS software was built to recognize two types of A/D equipment DSP and Datalabs although different types are easily added. Both types are 12 bit systems using ±5 V input, although the inner workings of the memory addressing are much different for each type. However, the software allows for different accuracy systems (such as a 16 bit system), and different maximum voltage ranges. At the time Calspan turned the facility over to the Air Force, 40 DSP digitizers existed (with a maximum sampling frequency of 100 Khz, there are 27 Datalab channels (three have sampling frequencies of 1 MHz, and 24 have sampling frequencies of 200 Khz), and there were 128 more DSP channels on order. The Datalab channels were originally designed as a stand-alone system. They offer their own amplification and memory storage systems. They can be configured to store multiple runs within the memory unit, although presently the entire memory (256 Ksamples) is used for each run. The DSP digitizers are part of a more modular unit which has its own timer and memory module. There was storage for 10 Msamples for the original forty channels, which works out to a maximum of 256 Ksample per channel (although if fewer channels are used more memory becomes available). Both the Datalabs and the DSP's can be triggered using software or from an external trigger. They can also be clocked from an internal timer or from an external signal, and they do not have to be the same. One set can be used to take data synchronously with rotor speed (using an encoder mounted on the rotor shaft to generate the timing signals) and the other set can be used to take data synchronously with a clock (sampling at fixed time intervals). With the addition of the new DSP channels, the total number of individual clock settings is a function primarily of the number of TRAQ controllers in the system.

The specialty instrumentation front-ends consist of the RTD power supplies, the pressure transducer power supplies and the heat flux gauge power supplies. In all cases the signals from the instrumentation front-ends can be offset at these units which makes the instrumentation fully compatible with other types of A/D systems which may not have a signal offset capability (such as PC based systems). All power is taken from ultra-stable, variable power supplies. Presently there are three separate power supplies, one for the RTD's one for the pressure transducers, and one for the heat flux gauges. It should be noted that there are some pressure transducers on the ATARR facility which are not used for testing but rather are used for control of the facility itself and these take their power from other sources and are used as input to the monitoring and control system (MCS).

Each of the front-ends has been built with gold-platted XLR connectors (to reduce corrosion on the contact leads) and with high precision, low thermal effect (50 ppm/°C) resistors. This makes the front-ends extremely temperature insensitive, which when coupled with the nominally temperature controlled instrument room reduces the error which can arise from temperature shifts (one of the biggest sources of poor calibration data).

Some of the calibration equipment was added to the facility after it was clear that ATARR needed to have a way of calibrating all types of instruments. Original instruments such as the Baratrons were part of the original scope of the project. However there was the addition of a Keithley 195A multimeter which uses a capacitance charge method of acquiring data as a way of acquiring data separately from the DAS system. The Keithley is a high accuracy multimeter which has a standard calibration procedure which is easily followed and can be routinely checked. The addition of this multimeter provided a way of calibrating the individual component of the A/D system: both amplifiers and digitizers. In addition the Keithley is used with an Omega laboratory RTD probe, calibrated in 1°C increments from -180°C to 480°C. Data fits can be created which allow this probe to be used to within 0.01°C (well below the resolution of the A/D from most RTD testing applications). An oil bath was also purchased for calibration of RTD's and thermocouples.

Pressure vessel of various sizes exist for which all pressure probes can be calibrated statically against one of the facility pressure standards. It is also a relatively easy task to incorporate new types of calibration equipment into the DAS system.

1.1.2 DAS Software

Clearly one of the main endeavors in building the ATARR facility was to deliver a software system that could be used by a variety of people who would ultimately have different concerns. Initially the DAS was configured only as a production facility, capable of cranking out many tests at a time. Over the course of the software development, several enhancements were added which allow the facility to be used as a research tool also, with on-line diagnostic capabilities. Specific properties which define the DAS system are those of file storage, user interfaces, and record keeping.

1.1.2.1 File Storage

The DAS system was designed to process and store large amounts of data. To do this the primary form of a data file is called the DAS format which has two records at the top of the file called the run and channel descriptor records, and the data stored in binary form. Two different types of data files are created in the DAS system. The most prevalent are the standard files in which only the dependent variable is stored. The independent variable (time or angle) is either stored in another file and accessed separately since this information is common to many files, or it is created "on-the-fly" if it is incrementing by a fixed amount every interval. A second type of file is called an "X-file" which has both the dependent and independent values stored in the file. For both cases there are two records at the beginning of the file which contain information about the run (the Run Descriptor Record, RDR) and the individual channel (Channel Descriptor Record, CDR). Items such as calibration constants, amplifier set-up digitizer set-up, type of data reduction scheme, etc. are contained in these areas. All information needed to reduce the data to the raw engineering units are stored in these records. Several utility programs exist to display and print the values of these records. The run record is the same for all files in any particular run, while the channel records are specific for each channel. A mechanism has been created to keep track of the RDRs and CDR's in a run which is called the Run/Channel Descriptor File (RCDF).

While the DAS format is compact and useful for large amounts of data, it cannot be easily viewed, nor is the data easily converted to other types of programs. To do that a second format exists called the VAR format. This format consists of a series of lines at the top of a file which contain information about the number of x axis's, y axis's and labels, and then the data (ASCII) is set up in columns separated by commas. Several utility programs exist which allow direct plotting of VAR files and the conversion of VAR to/from DAS formats. These forms a very useful for transferring reduced data to another type of data reduction system (such as PV-Wave on the Sun, or to a PC) for more detailed analysis. Programs written for research can easily output data in the VAR format which can be transformed in the DAS format for direct comparison to actual test data. This was done extensively when the main valve was being tested.

1.1.2.2 User Interfaces

There are primarily two types of user interfaces in the DAS system. Most of the main routines used for setting up a test, taking data, and reducing data are X-11 window based. This makes for a very easy to understand user interface with on-line help available. Any program can be made to run with an X-11 interface with the only difficulty being the actual programming of the interface. There are five main processes which use the X-11 based interface: the Test Set-up Process (TSP), the Fill and Calibration Process (FCP), the Data Acquisition Process (DAP), the Data Reduction Process (DRP), and the Math Model (MM). These processes allow the user to set-up a test (TSP), do diagnostic work on instrumentation and calibration of instruments (FCP), acquire data in DAS format (DAP), reduce and plot data (DRP), and model the facility both before the run and after the run using actual data to determine the test conditions (MM).

The other main interface is a FORTRAN option controlled program. These programs are fairly standardized in their operation and are generally written for smaller tasks. For instance there are several FORTRAN utilities which allow the viewing and printing of the records of a DAS file. There are utilities which allow users to input calibration constants from sources outside the DAS (such as values obtained from the manufactures or an external calibration). There are programs designed to take DAS format files and do least squares fits. Another program is used to generate the ensemble averages of data in either the vane or rotor frame of reference.

It should be noted that almost all of the data reduction code is written in FORTRAN and this is not the same as the user interfaces. Most of the X-11 interfaces (written in C) call FORTRAN subroutines. However this transaction is transparent to the user. This discussion here has just been limited to the different types of user inputs to the system. The advantages of using a FORTRAN based input structure is that it is relatively easy to create. Almost all programs start out as FORTRAN based user input programs. Only after the programs have been refined, does one integrate them into the X-11 interface.

1.1.2.3 Record Keeping

One of the greatest problems associated with the ATARR facility was deciding how information regarding data reduction should be stored. While there is no substitute for a manual data reduction book that describes how certain files were generated or how calibrations have changed, the DAS system can keep track of much of this information, allowing many different users to reduce data. This is done through the Run Log File and the Sensor Calibration Files. The Run Log File is an ASCII file which records when modifications have been done to data. Entries can either be made manually or automatically. Most codes which manipulated data also automatically write to the file. The file contains the date, user, and action done. Thus one can look at the Run log File to see who has done what to the data. The Sensor Calibration Files contain the calibration data for every sensor. There are three types of files used: lab calibrations, pre or post test calibrations, and used calibrations. They are separated because they contain different types of information. For calibrations done within the DAS, calibration constants are stored to these files and recalled from these files. No human ever needs to enter a set of calibration values in this case which reduces the chance of clerical errors. For cases where calibration constants come from outside the DAS, the user can input the constants through one of the FORTRAN programs. Both types of record keeping use a simple format that can easily be expanded to other codes, whether they are written in C or FORTRAN.

1.1.3 ATARR Software Capabilities

1.1.3.1 Calibrations

One of the great advantages of ATARR the software is the capability of performing on-line pressure calibrations and separating them from other forms of calibration. Nominally laboratory calibrations while done over a wider range of conditions, are done in advance of the test. Pre or post-test calibrations can be done immediately before or after the test, while the transducers are in place in the test facility against the same standards used for the laboratory calibrations. This type of system allows much better tracking of individual transducers and provides the ability to "spot" troublesome transducers before a test is run. Presently three different calibrations can be tracked for a given channel within the data file. In addition, ASCII files exists which allow the user to track an individual sensors calibration historically given the different types of calibration used. The main difference between these two is the former allows the user to track up to three different calibrations which could be used on any particular sensor for a given run. The latter system allows the user to see how this sensor has behaved over many different runs.

Data Reduction Paths

Presently the DAS system contains four different data reduction paths. Many of the same programs can be used in different modes, and it is only important for the engineer to understand the options available to optimize performance for a specific research experiment.

Single File Manipulations

This is the original system and consists primarily of **drp** and **direct**. **drp** is the X-11 interface which allows the user to load in specific files, plot the results, convert to higher units, or do some basic single file manipulations.

Single File/Complex Manipulations

A program called **quickmod** allows the user to add complex algorithms and process the data without the restrictions of **direct**. This is all done in FORTRAN and is well documented in the program **quickmod**. The user creates a simple subroutine, which **quickmod** will access. The user can then experiment with different types of algorithms relatively easily. The main program, **quickmod,** takes care of reading in the files and writing the files so that overhead is eliminated. Thus most engineers should be capable of adding these types of algorithms without any extra support. This is the preferred method for verifying new algorithms.

Multiple Files/Interactive

Most of the main reduction programs (**do_fft**, **ensemble**, etc.) allow the user to either load in a series of files, or create a series of files from the RCDF, and then process them all in a similar manner. As an example one can create a 40 Khz low pass filter and use it on a whole selection of files. The new files are named by adding an extension to the input file. For instance, a series of *.1 files processed with a filter may be called *.1.flt. These names are all controllable by the user. One program that falls into this category is **reduce**. This program allows the user to reduce all data selected from *.0 to *.1 files.

Multiple Files/Automatic

This last style allows a full automatic reduction of the data to some specified level. It starts with *.1 files and proceeds from there. The program that controls this is **automate**, and like **quickmod** it is relatively easy to modify. This program is best used with simple data reduction tasks which are the same for multiple tests in a matrix. When configured correctly, this can give almost instantaneous verification of the quality of the data. Presently, there are several algorithms supported by **automate. automate** is run by providing it with a command line file (as described in the program). The output can be either plots, or ASCII values, and the computer will generate an error file if problems arise. While it does take a little while to generate the first input file, this file can be used on all the test matrices. Major reduction of test data has been reduced to 1-2 hours using this method when appropriate.

1.1.3.2 Information Exchange

Another strong capability of the DAS is that it can freely exchange information with the outside world, while storing information in its own, space saving context. This can be done for calibration constants which can be either generated in the DAS system itself or imported from other programs outside the DAS context. The DAS programs can create data for plots that can be used outside the DAS system, and the DAS can import data from other sources and convert it to DAS utilities. Thus, programs which can develop presentation quality plots can be used as easily as **drp**. Thus the engineer is not constrained to using on DAS programs, nor does he have to repeat many commercial available programs in the DAS system.

1.2 Operating Philosophy of ATARR Data Acquisition System

The heart of the ATARR facility is the DAS. It is this system which provides the engineers the ability to translate the voltage changes in different instruments into known, reliable engineering units. In addition the DAS provides a context in which detailed technical questions can be answered through analysis of the basic instrumentation measurements. The DAS controls all aspects of data acquisition from set-up through reduction, and thus, it is critical to understand the philosophy of how the system is constructed in order to best utilize its' many capabilities. There have been many opposing forces pulling on the DAS design since the critical design review. One notable example is the need of a production facility (easy and fast) versus the need of a research facility (flexible). In a production facility, the computer system has a great deal of regimentation built into it. The test operator would not be allowed to run a test unless certain types of instruments were installed, and the number and procedures of the data reduction are limited to standard ones. Adding new reduction routines or test operations requires major changes to the computer system since there are many built in checks to protect the test engineer from making flagrant mistakes.

In a research facility, the computer system is far more general, requiring more attention from the test engineer. While mistakes which could be disastrous in nature can be checked by the computer, the test engineer can not be completely saved from his own inattention. This allows much more flexibility in the data reduction system and allows the DAS to be used for data acquisition on non-turbine testing (such as calibrations) without any change to the overall computer structure. Calspan has created a system which can both provide some protection against inattention on the part of the test engineer, a great deal of flexibility, and protects the engineer from creating a "garbage in-garbage out" situation by forcing the engineering to make key decisions, instead of allowing a system default which may not be appropriate in all test conditions. To accomplish this Calspan had to relay on some intrinsic checks (such as gauge naming conventions), external checks of engineer input (such as in the historical calibration data files), and a variety of input-output formats to provide complete flexibility and experimentation with data reduction schemes.

To fully understand and utilize the many features of the DAS system requires an understanding of how the system was developed and the different constraints it was developed under. This information should provide a better picture for how seemingly separate tasks are integrated together. While it may not be important for a person who is only going to use the system as it presently exists, we feel this knowledge is critical for anyone thinking about modifying the DAS. As discussed latter in this section, the system has been designed so that it can be easily modified. This is clearly a semi-dangerous situation, where a person could, unwittingly, make a confused mess out of the existing DAS if major modifications where attempted without fully understanding all the inter-connections. While protection does exist to keep this situation from unfolding accidentally, there is no real protection against a deliberate effort to change the entire DAS.

The focus of this section is to document some of the different forces that have shaped the existing system and provide an overview of how the system is set-up. Detailed technical discussion of the code and data acquisition techniques is shown in section II.

1.2.1 The Statement of Work and Constraints in DAS Design

While the general requirements for the Data Acquisition System (DAS) are recorded in the statement of work, realistically that document does little to guide the design of the DAS from an operations standpoint.

> "3.1.2 Task 2 - Design of Data Acquisition System. A data acquisition system shall be designed to provide acquisition, storage, and post-test analysis and display of heat transfer and aerodynamic test data. (CDRL seq #8, Atch #1)
>
> 3.1.2.1 General Requirements and specifications for the DAS. The DAS shall be a flexible, multi-channel analog-to-digital sampling and recording system. The purpose of the DAS shall be to sample and record the following general classes of data which quantify the turbine performance: (a) total and static pressures, total temperatures, gas flow velocities and flow angles, and gas flow turbulence information; (b) static pressure, surface shear stress, the surface heat flux readings from sensors on the flowpath surfaces of the turbine; (c) turbine speed, turbine output torque, slide valve position, coolant flow data, brake operating data, and all other rig variables required to interpret and analyze the data of classes (a) and (b). Some of the data of class (c) may be available from the ATARR Monitoring and Control Subsystem. In selecting a design for the DAS, the contractor shall ensure that the system performance and operating parameters are compatible with the typical types of instrumentation used in each of classes (a), (b), and (c). It is expected that most data will be supplied to the DAS as analog electrical signals whose characteristics depend on the particular transducer employed. The DAS shall include appropriate signal pre-processing, amplification, or conditioning necessary to achieve the performance requirements and specifications"[1]

Other subsections of task 2 in the statement of work focus on the number of channels and resolution of the A-D system, which also are of little use to the system designer.

After much debate at Calspan, the operation of the DAS was reduced to two, somewhat opposing, operating desires for ATARR and a whole list of concerns. The first major trade-offs was between the need to design a DAS capable of running the turbine tests efficiently vs. the need to have the DAS also be able to acquire data for purposes other than turbine tests. Other types of tests could include instrument calibration, or as been demonstrated already by the amount the Macintosh Fx has been used, facility monitoring tests. The second trade-off seemed to be between the need to process large amounts of data quickly in a standardized format and the ability to process data in new and different ways.

Coupled with these seemingly contradictory requirements were a whole collection of concerns. In addition to the standard ones of increasing complexity and cost, were more important technical ones such as how should information be transferred within the data reduction scheme. Or perhaps even more importantly, how should the DAS system protect the test engineer from making serious blunders. Examples of these types of functions may range from the relatively obvious

[1] Statement of Work, ATARR, August 7, 1989

such as arming the data channels, to the far more sublime such as using the correct calibration constants.

1.2.1.1 Turbine Testing vs. Flexible Data Acquisition System

The first substantial debate at Calspan was over the needs of the facility. Originally, the Critical Design Review focused explicitly on the needs that the DAS must accommodate when rather routine turbine tests are being run. Issues focused on configuring a test, acquiring data and reducing data. The desire was to create a system which could lead the test engineer through the various steps to insure a satisfactory conclusion to the testing process. However as the long range operational goals of ATARR developed, it became clear that the DAS would also have to run in environments which were not entirely the same as a turbine test. A specific example is the calibration of instruments. The need for extremely high accuracy measurements (±0.25% efficiency) complicated the issue tremendously since these accuracies require the construction of a calibration system for the facility basically an order of magnitude better than most other facilities. Thus one type of calibration test would not suffice and several types would have to be supported such as laboratory (or bench) calibrations and on-line (during testing) calibrations. Finally it became clear that the DAS would also be needed to verify performance of key operating components of the facility, such as the main valve, cooling system, and isolation valve. When running in this configuration, the DAS would operate in a mode where only a few channels would need to be acquired, and long lead times in setting-up the DAS system could not be tolerated.

The rationale for choosing the UNIX based operating system in conjunction with a X-11 window interface has probably already become apparent. The windowing environment provides the ability for the DAS to lead the engineer through a series of operations by providing a Graphical User Interface (GUI) driven interface. This alleviates the need for the operator to understand all the nuances of the control language and all the protocols for connecting different pieces of hardware. However, the need for this type of communication is still present even if the operator does not have to do it, the computer must do it. And that translates back to a need for the designer to predict all the needs of the testing engineer. That task is relatively straight forward if it is well defined, such as running only a standard turbine test. But once one starts to incorporate possible deviations from a "normal" test, or starts to account for the many different types of testing, the ability of the designer to create a system which only has three buttons, acquire data, reduce data, and plot data becomes an impossibility.

Another side of the same argument is much more fundamental. All data acquired is measured in volts. A change in the voltage over the test time corresponds primarily to a change in the measured properties (i.e. pressure, temperature, or torque). However they can also occur if the instrument has changed such as when a heat-flux gauge changes resistance due to erosion of the platinum element, or if the probes output drifts due to temperature changes. Where great precision is desired, calibrations both before and after a specific run need to be done to account for any changes in instrumentation sensitivity. How one incorporates the calibration data into the measured voltages to create engineering units is a matter of judgment. The best situation occurs where one obtains the same answer no matter how one combines the calibration data with the raw voltages, but that rarely happens. Since there is no pre-determined proper way of doing this, someone has to create a justifiable algorithm for doing it and understand how those actions affect the interpretation of the data. In an "expert system", the computer does this, and the algorithm for making the decisions is fixed into the coding structure. A better system explicitly states where the engineer must make judgment calls, so that he/she understands where in the overall data reduction process these situations occur.

This same problem reoccurs when reducing data from primary engineering units (temperature, pressure, etc.) to more interesting parameters such as pressure ratios, Nusselt and Stanton numbers. In many of these cases there are different ways to derive these quantities. The Stanton number is a good example since in addition to the physical definitions there are empirical correlations which relate the Stanton number to the Reynolds and Prandtl number (usually derived from external flow experiments). These empirical relationships derive from a Stanton number definition of:

$$St = \frac{Nu_x}{Re\ Pr} \tag{I-1}$$

and the Nusselt number is replaced by some empirical correlation such as

$$Nu_x = 0.332\ Re_x^{.5} Pr^{.333} \quad \text{(laminar, flat plate, Pr>0.6)} \tag{I-2}$$

reducing the Stanton number to a function only of the Reynolds number and Prandtl number. Equation 1-1 is a different form of the typical analytical definition which relate Stanton number to either the heat transfer coefficient:

$$St = \frac{h}{\rho\ U_\infty\ C_P} \tag{I-3}$$

where

h is the heat transfer coefficient

ρ is the density of the gas

U_∞ is the free stream gas velocity

C_p is the specific heat of the gas

or the heat flux,

$$St = \frac{\dot{q}}{\dot{m}'\ C_P \Delta T} \tag{I-4}$$

where

$$\dot{m}' = \frac{\dot{m}}{A} = \rho\ U_\infty$$

$\dot{q}$ is the heat flux

and ΔT is the driving temperature difference used in the calculation of $\dot{q}$. Which formula to use depends often on the information one is attempting to correlate, and the proper interpretation of the data depends on using the appropriate relationships. While the computer can make available all type of reduction algorithms to the engineer, it is the engineer's responsibility to invoke the appropriate algorithm, since it is dependent on the specific matrix point, and can not be determined *a priori*.

To accommodate the desire to make the DASeasy and self-guiding for a turbine measurement program, yet flexible enough to handle the variety of other testing situations, led to the compartmentalization of the entire DAS into different processes. These processes are describe in detail in section III of this report. The key is splitting the turbine testing procedure into specific steps which are compared to the steps needed for other operations such as bench-top calibration, or facility diagnostics. This results in groups of steps which are set-up as a stand-alone operations. These operations could be linked together in one manner to form the entire procedure for testing a turbine, or linked together with the addition (or subtraction) of other operations to form a completely different procedure.

Each of these units has been labeled as a "process." Presently the processes exist in three forms: X-11 interfaces, C-code, and FORTRAN code. Originally it was intended to have all the major processes working with X-11 interfaces. However, the programming in that environment is relatively lengthy, and there is relatively little flexibility lost by using either C or FORTRAN coding. Thus some of the processes have not yet been integrated in to the X-11 interfaces, and can be done in the future if no additional changes are needed. There are four X-11 processes: the Test Set-up Process (TSP), the Fill and Calibration Process (FCP), the Data Acquisition Process (DAP), and the Data Reduction Process (DRP). There are three processes written in FORTRAN: the Standards Reduction Process (SRP)[2], the Calibration Constants Process (CCP), and the Select Constants Process (Selcon). There is one written in C, Monitor.

Splitting up the overall testing operation into these processes has provided the ability to both guide the engineer through the entire operation, while also allowing the flexibility use the DAS in many different applications. More importantly, the processes are split such that judgment calls which need to be made, are made by the engineer and not the computer and the processes force the engineer to make these decisions before it will continue. While there are substantial benefits in using a pre-programmed interface, the problem is that adding new interfaces becomes a task requiring intimate knowledge of C, UNIX, and all the hardware. This is usually beyond the scope of most engineers experience, and is not something done easily. However, there is a need to

---

[2] This process is currently not being used. It however does exist and could easily be modified to serve a simialr role in the future if need be.

experiment and try new types of data reduction algorithms which can easily be implemented by an engineer. This problem led to the second major dichotomy which existed between the need to streamline data reduction processes and the ability to access information from a variety of different sources.

1.2.1.2 Streamlining Data Reduction vs. Immediate Data Access

First, it is important to realize that a standard turbine test generates a great deal of data. The original forty DSP Analog to Digital converters (A/D) can store 8 Meg. of memory. The other 27 channels (the Datalabs) have 9 Meg. of total memory. If one starts to account not only for the data generated during the actual turbine run, but also for data generated during calibrations (before and after the test), the total amount of data per run can become quite large. In fact most data files have the potential of becoming so large that they can not be handled as ASCII files, and rather have to be handled as binary files.

The problem with binary files is that key contextual information can not be stored by the numbers. This is done with header records (discussed in section II). These records contain information such as the units, the number of samples, the time between each sample, etc. which provides the context to interpret the data. However, reading these files and accessing the information becomes rather complicated.

The engineer needs the ability to access information outside of the DAS environment. This is critical when a new type of data reduction procedure is being tried, or new types of information are being generated. To add new processes or incorporate new FORTRAN subroutines into an existing process would be to cumbersome if the process were still experimental. Thus a series of subroutines, referred to as "utilities" were written to allow the user to access data generated by the DAS, and use it in their own FORTRAN code. These routines also allow the generation of files back into DAS format or into a different type of format accessible by some other program (Mathematica would be an example). Thus all the tools such as plotting, or traditional data analysis available in the DAS can be used on these files generated outside of the DAS. These utilities help the engineer avoid creating new interfaces when they really are not needed.

1.2.1.3 Other Constraints

The issues outlined above were the deciding issues in putting together the skeleton of the DAS system, but there were other constraints which put special demands on each of the individual processes. As mentioned previously, these focused on two key issues, keeping track of and passing information among the different processes, and providing checks within the system to prevent the engineer from making major blunders. The problems involved with keeping track of information can be shown quickly by examining the example of the Stanton number used previously.

Repeating equation 1-4 shows the Stanton number as a function of other parameters.

$$St = \frac{h}{\rho\, U_\infty\, C_P}$$

Each of these parameters could be time dependent or time averages, and in fact both types will generally be of interest. How one insures consistency throughout the entire data reduction process is of critical importance since errors in the bookkeeping will not be apparent. For instance this definition of a Stanton number can physically be thought of as a local heat transfer coefficient normalized by the physical properties of the fluid. This is important when geometric effects (such as cooling hole shape) are being evaluated for cooling effectiveness since any difference in heat transfer due to temperature differences, mass flow rate, and fluid properties are accounted for. Thus it is critical that the properties used to calculate $\dot{q}$ are the same ones used to calculate the Stanton number. Otherwise inconsistency, which is not obvious from the results, occurs. For one or two types of calculations it is a relatively easy task to do proper accounting. However, as the number and diversity of possible calculations increases, the potential to create a system where the wrong calculation is used routinely as a basis for other data reduction increases in probability.

To avoid this problem it is worthwhile examining how data is transferred in the DAS. The different processes used in the DAS lend themselves to two types of data input . These can broadly be classified as those supplied by the engineer at execution time, and those that are hard-wired into the code. In both cases it could be either the actual data required, or the location of that data. The key to reducing the potential problems outlined in the previous paragraph is to know when to use each type of input.

One tool which has been created to help alleviate these problems are the Run/Channel Descriptor Records (RCDR) described in much more detail in section II. While the formal description of the RCDR is that of a FORTRAN structure, it can be thought of as a common block, where information of interest to the different subroutines and processes is stored. The RCDR are records stored at the beginning of each file, and as stated earlier they provide the "context" in which the binary data is interpreted. Storing specific values, such as the ΔT used to calculate $\dot{q}$, in the RCDR assures that the right values are accessed by subsequent subroutines. The different values stored in the RCDR are listed in section III and these range from historical information such as what A/D channel was used to the initial temperature of the test section . One of the benefits of this system is that the number of values is not fixed, but rather can change to reflect those calculations which are the most important. In addition, values do not need to be passed between programs in this fashion, they can be directly inputted into the process or subroutine by the user if desired. In addition, values which are being written to the RCDR can be made extremely explicit, so that the engineer is forced to understand exactly what is happening. And these transactions can easily be written to other types of files such as log files.

One example which shows the full utilization of the system is the way calibration of the pressure sensors is accomplished. As touched upon earlier, pressure transducers are expected to have a series of calibrations performed on them: a bench calibration, a pretest calibration, and a post-test calibration. Each has potential benefits and weaknesses. The bench calibration can be done in a controlled temperature, with many data points, and many different times. Yielding perhaps the best type of calibration possible for that instant in time. However, the bench calibration can be expected to be performed weeks prior to the actual experiment, allowing for long-term instrument and electronic drift to occur. Both the pre and post-test calibrations will be much closer in time (within 1/2 hour) of the experiment, but due to the pressures of the test plan and the lack of control over some environmental factors (e.g. temperature, pressure ranges, etc.), will probably not have the number of data points or the amount of data that the bench calibrations have. Thus the engineer is faced with a dilemma, which calibration should he use?

This becomes especially critical when the different calibrations seem to contradict each other. Four different sets of calibration data are provided for in the CDR. C_Lab has stored in it up to seven polynomial coefficients for the bench calibration. C_Pre and C_Post are similarly the coefficients for the pre and post-test calibrations. If fewer then seven coefficients are used, then the remaining ones are blank. Calibration Constants enter the DAS system either by being generated by a DAS process, or by being inputted into the DAS using CCP. In all cases, the actual values are stored in ASCII files which are used as logs in addition to the RCDR. These files are used to trace the calibration of individual instruments from run to run.

However, the job for the engineer is not over in terms of calibration. He has generated three different sets of calibration constants by the time he has acquired data, but he has yet to decide which one to use. Plotting the results may show little or great variation. As mentioned earlier there is no "right" way, but he/she is forced to make a decision. The Selcon process allows the user to store the desired calibration constants into a forth set of calibration. labeled as C_used and also in another historical ASCII file labeled as **xxx.used**.

Data files have distinct levels. The most basic level is called the zeroth order and is the raw counts recorded by the A-D. First order files are the engineering units converted from these zeroth files and these require the calibration constants and a data reduction algorithm. Higher level files are based on the information found in these first level files. It is clear that keeping track of what calibration constants are used is critical. One could have set-up the system such that the calibration constants are typed into the system. However, it would be hard to determined where these curves were generated. In the present system, level one files are only created using those constants found in C_used.

This process accomplishes all the required goals. It keeps track of all the different types of calibrations, storing them with the files, and without destroying any information. It has written into the code a way to log the different calibrations into historical files which are easily read. and most importantly, it puts the responsibility on the engineer for determining which set of calibration constants to use. If the Engineer wishes to always use the pre-test calibrations, he/she can, but it has to be done explicitly.

This process of determining the calibration of the pressure transducers shows another constraint that we tried to accommodate, the ability for the computer processes to protect the engineer from making major blunders. One method, as shown above, is to make critical steps explicit for the engineer. Generation of first order files cannot be accomplished if the calibration constant have not been placed in C_used, even if the other three are filled. Thus the engineer has to at least run through the Selcon process. This type of system check is used throughout the entire DAS.

Two other methods are also worth noting. These are the use of the standardized instrument naming convention and the use of toggles in the RCDR. Both of these topics are described in latter sections, but their operating principles are easily explained here. By using a standard naming convention, the computer process can be set up to make sure that key steps are not overlooked. For example if the third letter of each name represent the type of instrument (T for temperature, P for pressure, etc.) then when a pressure calibration is being run it is a simple manner to have the computer scan through all the channels and make sure that the engineer has marked all the pressure channels for calibration. If he has not the computer is designed to give an error message that the engineer can override if desired. This does not constrain the engineer, but the computer becomes a friendly checker, not forcing changes, but showing potential inconsistencies. And like other parts of the DAS, these checks are not fixed, but can be modified.

Toggles on the other hand are used by one process to indicate a particular state of the file to subsequent processes. Thus it is a simple manner to make the computer scan through the different toggles and make sure they are all set to the appropriate value before preceding. A good example of this process occurs with the data reduction algorithms. Presently there are two types of reduction algorithms referred to as "absolute" and "relative". These correspond to how a voltage measurement is converted to engineering units. The best system (in most cases) is the absolute system, where the voltage is converted directly to an engineering unit (E.U.) using calibrations derived for that particular instrument/ hardware configuration. For a variety of reasons, the ability to calibrate through the actual wiring may not be available (particularly with the heat flux gauges). To eliminate problems which could be induced by the wiring, a relative system is used where the actual recorded voltage is subtracted from a base line voltage. A calibration curve is used which converts ΔV to $\Delta E.U.$ and if the actual E.U. is known at the base voltage the E.U. at all times can be reconstructed. A problem could occur if the bench calibration was done in a relative calibration frame, and the pre and post-test calibrations were done in an absolute frame. A toggle (namely

DELTAV in this case) is used to keep track of which algorithm is used. And once again, the computer can be designed to look for particular problems by checking the toggles. And programs which set toggles also force the engineer to make choices, thus making the engineer confront the idea that this particular choice does in fact have major implications downstream in the data reduction process.

1.2.1.4 Summary

The main goal of this section has been to show how the various constraints and desires for the DAS have pushed us towards the present configuration, and how the design of the system accomplishes these goals. We felt it was critical to explicitly show these interconnections because the design of the system is relatively "passive" in that the control of many problems are accounted for by the use of toggles, naming conventions, and the RCDR. And since the system is easily modified, one could unwittingly eliminate many of these "passive" features just by changing the types of information stored in the RCDR, or by mistakenly eliminating what may seem to be extraneous information in the RCDR. And of course this section has not been designed to be the heart of the technical discussion, which is found latter in this report. The next part of this section is devoted to describing in a cursory fashion the main computer processes that presently exist. The final part of this section discusses just a few of the basic blocks of the DAS such as the plotting routines and some of the ASCII historical files. While not separate processes, they provide the ability with which some of the key engineering judgments are made and recorded; and as such it is important for the reader to understand their connections to the main computer processes as they review the more technical components of the DAS in latter sections.

1.2.2 General Description of the Computer Processes

As an overview we can examine the six different processes that have been developed to this point. The following descriptions are brief. The more detailed descriptions are found in section II of this report.

1.2.2.1 Test Setup Process (TSP)

This is the first process enacted and it accomplishes several tasks. It is in this process that channels and labels are assigned. One important property of the label is that it is used as the filename for the data recorded by the DAP. The standard naming convention is shown in table 1 (which, subject to the warnings of the previous sections, does not have to be adhered to). Also in this process the hardware configuration is stored for the test. This is accomplished by down-loading information from the Hardware Resource File HRF and the Sensor Calibration File (SCF (both of these are discussed latter). The channel is the designation used to denote the amalgamation of crate, amplifier, digitizer, and sensor. Upon exiting the process, the software amalgamates much of the information and stores it in the appropriate places in the Run/Channel Descriptor Record (RCDR.

1.2.2.2 Fill and Calibration Process (FCP)

This process is usually done immediately before a test, to fill the tank and do the on-line pressure calibrations, and after a tests to assess possible variations in the pressure transducers. However it also can be used for bench top calibrations or as a random data acquisition package. This process has many options which can be used in any combination. They are 1) mark the fill points, 2) calibrate supply tank transducers, and 3) calibrate test section transducers. The first option is used when the supply tank is being filled. "Marking" the fill points corresponds to having the computer store the data which corresponds to the partial pressure points for the N_2 and CO_2 which are needed to determine specific properties of the test gas. Options 2 and 3 allow for calibration of the pressure transducers in either section. This is basically done by filling either or both sections to some pressure. The pressure levels are recorded on the standards (the Baratrons) and the pressure transducers. This process has access to a plotting package (discussed in the next section) which allows many channels (up to 25) to be displayed simultaneously en mass or individually on the screen. The information is presented relative to the arithmetic mean, within a user specified number of standard deviations, in the recorded time interval which is also under user control. The statistics for the interval are reported in volts and pressure units to aid in the assessment of the data's quality. This allows the engineer at the time of calibration, to determine whether the transducers were stable. If the data is acceptable, the test engineer can save the data and the computer will use it to calculate calibration curves. If the data is not acceptable, the calibration point can be retaken (this would occur if the transducers were not a equilibrium), or the engineer can stop the process and check the instrumentation. This situation forces the engineer to make an assessment of the instrument quality before running an actual turbine test with bad instruments.

At the end of acquiring calibration data, the computer process reduces data interactively with the engineer to create calibration curves. These are then stored in the RCDR and also in the historical calibration files.

1.2.2.3 Data Acquisition Process (DAP)

Initiating this process probably means that a turbine experiment is nearly ready to go. All the groupings defined in the RCDR are checked for consistency with the existing hardware configuration (this is where the computer checks against serious blunders using toggles and naming conventions). Any anomalies detected during the check must be corrected in the TSP before the system can be armed. Once armed the system awaits the trigger which starts the experiment or a request to abort the state of readiness.

If an experimental trigger is detected, the data are recorded. Unless a false trigger occurred the data are then downloaded to the individual channel related DAS data files. At this point an immediate tape backup process should be initiated to safeguard the data from loss by conscious or accidental erasure. The RCDR will be incomplete because the initial conditions and standards parameters have not been determined. Another tape backup should be performed after these processes are completed.

1.2.2.4 Standards Reduction Process (SRP)

This process would be the next new process conducted in a standard test after the DAP. However, immediately after a test and a tape back-up of the data is made, the engineer may elect to run the FCP a second time to calibrate the pressure transducers after the test. However, with that process aside, this one has at its goals the reduction of the standards from the test. This allows the engineer to quickly see how the standard temperature, pressure , and speed of the turbine varied over the test. It also allows the storage of base line conditions in the RCDR. At the present time this process is not being used.

1.2.2.5 Data Reduction Process (DRP)

By the time this process is enacted, the RCDR will be mostly filled (the only parameters left will be those generated in this process). One should make another tape back-up before proceeding with this process. This process will be used over a course of months to reduce all the data to usable forms. This process has access to plotting routines and data reduction algorithms. In addition, other data files can be analyzed in this process by using the conversion routines described in section 4.

It is in this process that data is taken from its primary form (i.e. counts) and converted into engineering units. The file system (described in section 2) is set up so that each label is followed by a ".x" where "x" represents the level of reduction. A "0" corresponds to pure counts. A "1" is a primary engineering unit (which is time, pressure, temperature, or volts). The "2" is a level which can be derived only from a level 1. Types of data in this level are only heat flux data. The last type of data, "3" can be derived from any combination of "1's" and "2's" and would be pressure rations, Nusselt numbers, mass flow, etc. Conversion to level 1 (the primary engineering units) is simple since the appropriate calibration curves have already been selected in the ICP. However, one can imagine that selecting the appropriate calibration curves and the subsequent data reduction might take quite a bit of time.

1.2.3 General Description of the Accessory Files and Plotting Routines

As described throughout this first section, there are many key support utilities which are used by the different processes. While the technical details are left to subsequent sections, we briefly describe their purpose and operation here.

Plotting Routines- There are two plotting packages which are used in the DAS extensively. One allows multiple traces (up to 10) on a single graph. The user has control over labels and axis. This package is used extensively by the DRP. The second allows many single plot graphs to be expressed in one window. These graphs are displayed as variations around a mean and have their mean values and standard deviations and x-axis controlled as a group. This package is primarily used to view several sensors to see if they are exhibiting similar behavior. This need occurs primarily in the FCP when one is examining the sensors during calibration to make sure they are all at equilibrium. If needed this routine can also blow up individual graphs and to give more detail.

Hardware Configuration Files- These files contain the main information on how the hardware is interconnected. This information gets downloaded into the existing run data base during the TSP operation. Thus, once the hardware is set-up for a specific testing sequence, it can be copied into subsequent tests. If modifications are made to the hardware for specific tests, these can be made easily without having to retype the entire file. Presently there is only one of these files, although as the system grows there may be additional ones.

Historical Files- These are ASCII files which contain information gathered over several tests of interest to the engineer. They are set-up in a manner similar to log-books, and in fact they could easily be replaced by log-books. Their main function is to automatically keep track of information and to pass information between programs without making the engineer type in many constants and introducing the possibility of typographical errors.

Presently there are three such files all of which involve calibrations. One file named **xxx.lab** keeps a record of the bench top calibrations of each sensor. Data in these files include information such as date of calibration, hardware configuration, type of calibration, etc. These files are generated for each sensor individually and thus show how the calibrations change over the course of time. The other two files are similar in nature but have data which corresponds to the on-line calibrations (both pre- and post-test), and the actual calibration data used.

DAS Conversion Subroutines- There are two subroutines (called d2vfmt and v2dfmt) which allow the engineer to take the binary data out of the DAS environment, convert it to ASCII, and process it in some special way, and then convert it back (if desired) into a DAS readable binary form. This allows the engineer to quickly write his own FORTRAN programs to reduce data in forms other than those presently accepted in the DRP. This is critical when trying to correlate new types of data in an effort to develop new theories.

1.2.4 The DAS Computer Operating Environment

The DAS is designed to make efficient use of the UNIX operating system, the X11R5 window interface, and both "C" and FORTRAN language programming. To make effective use of the DAS a user needs to be proficient with a UNIX supported text editor such as its visually-oriented[3] text editor vi. A full set of user manuals is provided with the system discussing UNIX, X11R5, C, FORTRAN, and vi, but there are several reference texts the user might consider

[3] A visually oriented editor is a step above a line editor (which only allows the editing of single lines), but is not quite a "what you see, is what you get" editor, like a Macintosh. The basic difference between the two is that formatting commands in a "visually-oriented" editor are shown as commands and not the final product. This difference is not really important for the DAS. What is most critical is that an editor other then a line editor be used. Otherwise modifications to the DAS would become almost impossible.

obtaining for additional support. Some good examples are:

The UNIX Programming Environment by Brian W. Kernighan and Rob Pike[4]

Learning the vi Editor by Linda Lamb[5]

The C Programming Language by Brian W. Kernighan and Dennis M. Ritchie[6]

UNIX System V A Practical Guide by Mark Sobell[7]

The X Toolkit Cookbook by Paul Kimball[8]

There were several overriding considerations in the DAS design. Two prominent ones are portability and flexibility. UNIX is an operating system which strongly satisfies these requirements and continues to grow in preference and stature among programmers and system managers. It provides a file system which permits the management of information whether it is in the form of data files, programs or user created documents. This factor was weighted heavily when the decision was made to eliminate a separate data management process for handling the expected large volume of data to be generated by ATARR. The X (version 11) Window System permits multiple screens containing overlapping windows. It was designed at MIT and has become the industry standard for providing network-transparent windowing capabilities. ATARR's console and its two monochrome 19" terminals support the X11R5 protocol and ample use is made of its utility in the formal DAS processes using the X Toolkit Intrinsics and Athena Widget Set provided with the C Language X Interface documents.

In general, the everyday user will not be concerned with anything other than how to set a task in motion and access or interpret the output within the context of the UNIX time-sharing operating system. However, changing the DAS processes will most likely require a user with a firm grasp of the C and FORTRAN programming languages.

All programming related to the DAS is written in either the C or FORTRAN languages. FORTRAN is used exclusively for the computational support routines in the DRP and C is used for everything else. While it is not in the scope of this document to instruct the reader in C and FORTRAN programming skills, it does provide the details concerning compiling, linking and executing all DAS processes as well as the related software. One of the reasons for selecting UNIX becomes apparent when trying to re-compile the programs after changes have been made.

The UNIX *make* command provides the utility necessary to identify all the information required to create or alter any DAS program. A fine treatment of this useful command is presented

[4] Kernighan, Brian W., and Pike, Rob; The UNIX Programming Environment ; Prentice-Hall, 1984.

[5] Lamb, Linda; Learning the vi Editor; O'Reilly & Associates, 1990.

[6] Kernighan, Brian W., and Ritchie, Dennis M.; The C Programming Language; Prentice-Hall, 1988.

[7] Sobell, Mark; Unix System V A Practical Guide; Benjamin/Cummings Publishing Co. 1995

in the "UNIX BASICS" series of the magazine SUN EXPERT in the February and March 1992 issues[9]. The UNIX *Make* command identifies all the dependencies that go into compiling and linking files. Thus when one change is made in an associated program, the Make utility keeps track of how that influences the final program. The end result to the user is that all the connections are described in the Makefile, and this file becomes an important reference in any directory as to how the contents of the directory are compiled.

1.2.5 Summary

This first section has tried to give an overview of the entire DAS system, show how the various components are interconnected, and briefly outlined the purpose of the major components. However, all of these components are supported by a wide variety of subsystems, all of which need to be understood in order to fully utilize the DAS. These systems control the passing of information between processes, the "book-keeping" of where information came from, and the storage of information.

[8] Kimball, Paul; The X Toolkit Cookbook; Prentice Hall PTR, Englewood Cliffs NJ, 1995

[9] "Unix Basics", Sun Expert, February-M arch 1992.

Section 2. Operation Notes

There are many ways to organize those "key" pieces of information which seem relatively simple, yet for the uninitiated cause great consternation when operating the DAS. We have tried to organize this information in the past and have succumbed to either presenting too much information (to the point where the key points are lost in verbiage), or providing such a cursory overview that it was hard to see how many programs worked together. To solve these problems we have decided to split this part of the DAS manual into subsections which have separate goals.

The first part is called "Program Notes" and their write-ups are identified in the header of the page. These notes are designed specifically to review how a program operates, what its reversion history is, record problems with the code, and direct the reader to associated programs. The goal of this section is to make future documentation easier by summarizing each of the main programs in a separate note, which can stand by itself as a document, and is easily modified. Not all the programs are documented in this fashion; those that are small and self-explanatory are usually referenced in the write-ups of the larger programs.

The second part is called "DAS Technical Notes". This section is a collection of notes discussing how groups of programs operate together, or key technical issues. This also keeps individual notes short and easy to document in the future, plus it separates out the individual operation issues of each piece of code (which can sometimes be very lengthy) from how the code interacts with the rest of the DAS system. We hope this documentation system will allow the reader easy access to their required information.

The third part of this section contains "flowchart outlines" of some of the main programs. These have been generated over time as question have occurred over how particular programs are interconnected. These have mostly centered around those programs involved in data acquisition. The first part of this section describes some of the nomenclature used in these outlines. These outlines are not completely detailed, but the should provide a good overview of how the different subprograms are connected.

As a general overview, table II.1 lists the present contents of each of these sections. Note that there are larger descriptions of some other programs such as the facility model and the traversing ring software which have manuals of their own, and thus are not included here.

Table II.1 Overview of Separate DAS Documents

Program Notes	DAS Technical Notes	Flowchart Outlines
tsp Program Notes	The DAS Program Structure and History	tsp
dap Program Notes	Using "Traverse"	monitor
drp Program Notes	The RCDF and its Interactions with DAS files	dap
fcp Program Notes	Channel Lists and the Keithley Standard	fixit
mm and Travset Program Notes	Calibration Constants and their Path Through the DAS	getchaninfo
automate Operating Notes	The DAS Recording Mode: Absolute and Relative	zerochannel
monitor Operating Notes	Data Reduction to Primary Units	
Other Program Notes		

Listed below is general information common to all programs; both technical notes and program notes. It is far easier to explain these items here than adding repetitive explanations to every write-up.

General Items:

Throughout the write-ups, programs are always referred to in bold type and in the lower case letters used to start them (even at the beginning of a sentence). While this may not seem stylistically correct in some areas, it is important to remember that UNIX is a case sensitive language and typing **drp** will not do the same thing as typing **DRP** or **Drp**. Major Acronyms (such as the RDR, CDR, RCDR, RCDF) are always referred to in capital letters. When file names are being used as an example (such as *.1), the * refers to any name (it is a wild card). Thus a command to list *.1 will provide a listing of all the files ending in .1

Program Notes:

The X-11 program notes share many things in common. At the beginning of the note there is always a section in bold describing the program name (and what it stands for), the location of the code as delivered to WPAFB (which may have changed), the version number and the documentation author. The version number represents the authors best approximation of what major modifications have been made to the code and does not include minor fixes to bugs encountered along the development path. Basically, every code is either version 2.0 or version 2.1 with the author being responsible for most conversions to 2.1. All of the main code has been written by the following people: Bob Meyer, John Moselle, or Dave Yearke, with guidance from Charlie Haldeman. Most of the code have the major authors at the top. Many of the smaller FORTRAN subroutines are direct Calspan routines and authorship may have been lost in antiquity.

After the basic information, there are usually six sections listed in the following order:

Modification History: This lists the main modifications made to the code, hopefully the date and the author.

Main Program Objective: This describes as concisely as possible what the program is supposed to do

Main Program Interface: This describes the buttons on the interface, and any interconnection between them or order dependence. It should be noted that most of the X-11 interfaces have good "on-line" help files which provide more detail about each program. When possible, there will also be attached to these write-ups copies of the X-11 screens for reference. There are basically four types of X-11 Widgets used in the displays and in the writes-ups most of the widgets will be grouped into one of these categories.

Display Only: This is a box which displays information, but which the user cannot modify

Modifiable Display: This box can display information, but the user can input his own and it will be stored as such.

Menu: These are pull-down menus (sometimes only a toggle) that allow the user set choices. These are usually indicated by colored areas on the screen.

Command Buttons: These are buttons you push to do something (such as printing or changing the display).

Main Program Operation Notes: This section describes some basic information about operating the program (if it is not self explanatory from the previous information). Sometimes this is done by using examples

Known Bugs: Bugs are defined as something which should not happen, but does. This is different from "Quirks" as defined below. Hopefully all bugs have been fixed by now.

Known Quirks: Are annoyances in the program logic, which cause predictable behavior. These, like bugs should also be fixed, but may require some rethinking of the main program logic and generally will be fixed when a major upgrade occurs.

2.1 tsp Program Notes

Program Name: tsp.c
[Test Set-up Process]
Location: /usr/atarr/src/tsp
Access: Typing "tsp" in any directory that has a path to /usr/atarr/bin (or equivalent location)
Version: 2.1, last modified 9-94
Documentation Author: C. Haldeman, 12-29-94

Modification History:

Version 2.1 Sept. 1994

Modified to show the program version number in the main X-window interface, and to include a "copy thru" button. This button allows one to copy the contents from channel A to all the channels between channels A and B (shown in the channel number box). It behaves in a very similar manner to the "copy to channel" button.

Main Program Objective:

The program **tsp** provides one type of interface between the user and the RCDF[1]. Other interfaces are the "hardware_resources" file, **viewdas**, and **listdas**[2]. **tsp** is used to set-up the data acquisition portion of the experiment and it provides a methodology for reviewing/modifying the contents of the Run Descriptor Records (RDRs) and all the Channel Descriptor Records (CDRs) for any given run. When items in the display are modified, they are stored until the RCDF is saved as a whole. Only those parts of the (Run/Channel Descriptor Records) RCDRs which are necessary for the test setup are displayed. Some items of the RCDRs are displayed for general information and cannot be changed, while others need to be changed to account for the individual

[1] The "RCDF" or Run/Channel Descriptor File is an integral part of the DAS system. The reader is referred to the technical note on the RCDF and its role in the DAS system for an overview.

[2] These are covered in more detail in other locations, but as a quick overview:

The hardware_resources file is a file that contains all the physical information about the A/D system: amplifier location and A/D connections. This information is used to control the hardware via GPIB and CAMAC commands.

viewdas is a program which allows the user to display and change all of the parameters in the RCDRs of individual data files, or in the RCDF.

listdas is a program which will print out the information contained in the RCDRs of individual data files, or the RCDF in different types of formats experiments. This program is the one that ultimately sets-up the RCDF for any given run. This RCDF is then used by the other main programs (**fcp, monitor,** and **dap**) to acquire data.

Main Program Interface:

Upon activation, the user is faced with a large screen (see fig. 1) which contains two main parts: the command line and the display. The display toggles between the Channel Descriptor Records (CDRs) and the Run Descriptor Record (RDR) shown in fig. 2 and is controlled by the "Edit Run Parameters" command button located in the command line. The main command line consists of five buttons:

Load RCDF - This button activates a file selector which automatically looks in the current working directory for files with a "rcdf" appended to the name (e.g. run27.rcdf). The user can then load an old RCDF for updating or editing.

Save RCDF - This button activates a file selector which allows the user to save the file to a specific location.

Edit Run Parameters - This button changes the main display screen to examine/edit the information stored in the Run Descriptor Record[3]. This button toggles between the "Edit Run Parameters" and the "Edit Channel Parameters" screen

Update Calibrations - This button will update the calibration constants stored in the laboratory constant area (these are displayed in the calibration constant area of **tsp**). It accomplishes this by interrogating a file called **xxx.lab** (where **xxx** is the serial number of the sensor) stored in /usr/atarr/files, and reading back the last line of the file. This is very useful if the laboratory calibrations have changed from when the RCDF was first created, or if one wants to input the calibration constants for a large number of channels automatically instead of typing in all the numbers (as done through **ccp**).

Quit - This button stops the **tsp** application, and will always prompt the user to save the work (even if the user has just saved the RCDF!).

---

[3] As one will recall, for any given run or RCDF there is one Run record and multiple Channel Records. Each data channel contains a copy of the Run Record and its individual Channel Record. In essence, **tsp** is allowing the user to examine and edit the Run record (when one is in "Edit Run Parameters mode", and then scroll through each of the channel records individually when one is in the "Edit Channel Records" mode.

The next part of the display is the information for either the Run Descriptor Record or the Individual Channel Descriptor Records (depending upon the status of the "Edit Run Parameter Button"). These two displays are examined below:

Channel Parameter Display (fig. 1)

This display contains the main parts of the channel records. Some of which can be changed by the user, and others of which are shown for information purposes only. At the top of the display is a command line which has several buttons to control the channel being worked with. The information shown below this line corresponds to that particular channel. The command line consists of the following buttons/displays (from left to right):

Channel Number: (Modifiable Display)

This display contains the channel number you wish to operate on. It operates in two modes. When a number is entered into this display and one of the other buttons on this command line is pushed (copy to, copy thru, go to), the program will go to that channel number and will display the information for that particular channel. If one of the other buttons is pushed (next channel, previous channel) the display will show the current channel number.

Channel Label: (Modifiable Display)

This display will show the current channel label (note all capital letters, seven letters or less). The default value is "GRONK". The user can input a new name for each channel. The program will display the last stored value. This label is used by the DAS to create the data files after acquiring data.

Go To Channel: (Command)

This will change the display to show the channel record of the channel listed in the Channel Number box

Copy To Channel: (Command)

This will copy all of the modifiable parameters from the present channel to the new channel indicated in the Channel Number Box.

Previous Channel: (Command)

This decrements the channel number and display by one channel.

Next Channel: (Command)

This increments the channel number and display by one channel.

Copy Thru Channel: (Command)

This is similar to the "Copy To" channel button, except that it will fill all channels between the current one and the new one listed in the channel number box with the modifiable parameters.

The main display consists of several different parameters (some of which are modifiable and some are not). These are described below:

Sensor Type: (Menu)

This menu allows the user to select a sensor type that will be used with this particular channel. This information is important, since the sensor type will be used to determine the type of data reduction used by the reduction algorithms (**drp**, **direct**, **reduce**, **automate**, etc.). Whatever type is selected will be displayed in the box.

Sensor Serial number: (Menu)

Whatever sensor type is selected, an active display of all available serial numbers will be displayed from the file **sensors.list** located in /usr/atarr/files. The serial number is important because calibrations are based on serial number and not label.

Scale Factor: (Modifiable Display)

This display allows the user to set a preprogrammed scale factor (in Engineering Units/Volt). This is only used for heat flux gauges and for doing a preliminary check using **fcp**. Most conversions to Engineering units are handled through the calibration coefficients.

Channel: (Menu)

This display will show whether a particular channel is active or not. There are always the number of channels as described in the hardware_resources (129 for example for the TTF). However, only those channels which are active will be recorded and operated on. As an example, if there are 128 possible A/D channels and 1 spincoder, then there will be 129 possible channels. If one activates only one channel and then saves the RCDF. Upon reloading, all 129 channels will appear, but only the one channel will be active. When data is acquired, only that one channel will be acquired.

Recording Method: (Menu)

This toggles between three possible values: Absolute, Relative, and Calibrated. Of these only the first two are presently fully supported. This is an important parameter because it influences how the data is reduced (**direct** and **reduce**).

Sample Rate Source: (Menu)

This toggles between two possible sources (presently), the clock which is a fixed sample rate time base generated by the A/D or the indexer which presently is an external clock connector to the Spincoder, which generates samples at fixed angles of rotation.

Sample Rate: (Modifiable Display)

This sets the acquisition rate of the A/D channels (in Hertz). This display will affect all of the channels which are tied to the same controller (determined in the hardware_resources file). Originally at ATARR, there was only one controller for the DSP's and thus this would affect all the DSP channels.

Number of Samples to Acquire: (Modifiable Display)

This is the number of samples for each channel. Behaves in a similar manner to the sample rate display.

Below these are smaller groups of displays which include:

Laboratory Calibrations:

These display the laboratory calibrations of the individual sensor selected if the "update calibrations" button is pushed. If data is entered via this manner, the calibration constants are stored in the RCDF but are not written to the calibration constants file.

Thin Film Heat Transfer Parameters:

These contain all the information needed to reduce the data for the one sided heat-flux gauge. For a description of the individual displays see the heat transfer data reduction notes.

Digitizer information: (display only)

These show the A/D connections as described in the hardware_resources file.

Sensor Position: (Modifiable Display)

These contain position information about the sensors. Presently this information is only used by **ensemble** and one should refer to the operation of that code for further explanation.

Amplifier Information: (modifiable display)

Some of the parameters are display only and receive their information directly from the hardware resources file. The user selectable displays are the anti-aliasing frequency and anti-aliasing filter type (both of which are menu driven).

Run Parameter Display (fig. 2)-

This display contains information stored in the RDR. This information may be accessed by other programs so it is important that it nominally be correct, or at the least that the user knows what programs will need what information. Any program which asks for an RCDF file may use this information. The most important piece of information is the run number. This value is used by the computer to create a new directory (**runxxx**, where **xxx** is the run number) where the data will be stored. If a run already exists, the computer will ask if you wish to overwrite the run (when **drp** is executed). Thus, this run needs to be changed every time a new run is contemplated. The next two lines are strings which contain information which will be placed automatically on all plots generated in **drp**. These can be changed in **drp** if necessary. These should be considered as default values. All other data is used by only two programs at the present (the facility model, and **ensemble**). Besides the correct run number, there is nothing else in this display which must be set to acquire data, and any changes which must be made can be made after the fact (but it is, of course, much easier to do it correctly the first time).

Main Program Operation Notes:

In accordance with its main objective, **tsp** is used to setup the entire data acquisition and reduction for any given run. Thus it is best if one has at least a basic understanding of what needs to be accomplished in the experiments before starting the program. Several things will help make setting up the run easier to handle.

1) Setup the Sensors.list file before you start and include all the sensors serial numbers that you might use in the test matrix (not just the particular run). This makes modifications between runs easier.

2) Group similar channels together in the channel hierarchy:(channels 1-10 are RTD's, 11-20 are Kulites, etc.). Thus one can fill in most of the first channels information (Sensor type, recording type, active, filter frequency, filter type, etc.) and then copy thru to the last channel of the group. Thus one will only need to change a few parameters.

3) Make sure to understand how the data reduction will occur (Parameters: Sensor type, Recording mode). If there is any question refer to **direct.F** as a guide.

Known Bugs:

None

Known Quirks:

1) Entering laboratory calibrations through **tsp** does not store the data in the appropriate sensor calibration file.

2) The program will allow the user to load an RCDF with an incompatible hardware_resources file. If this situation arises, the main acquisition programs will not operate correctly (because the hardware resources file is not agreeing with the CDRs). Thus before tests are set-up, one should make sure the hardware_resources file is relatively static.

2.2 dap Program Notes

Program Name: dap.c

[Data Acquisition Process]

Location: /usr/atarr/src/dap

Access: Typing "dap" in any directory that has a path to /usr/atarr/bin (or equivalent location)

Version: 2.1, last modified 9-94

Documentation Author: C. Haldeman, 1-1-95

Modification History:

9-94 Modified to show version number in main X-window interface, and to display the spincoder and A/D sample rates and sample numbers when the RCDF is called-up.

Main Program Objective:

This program arms the computer for high frequency data acquisition. The program will arm the A/D hardware and it can be triggered either via software, or through an external source. The program will create a run directory based upon the run number stored in the RCDF which it uses as its base, and will create data files based on the Channel labels in this directory.

Main Program Interface:

Upon activation, the user is faced with a relatively small screen containing only command buttons (fig. 1).

Load RCDF - This button activates a file selector which is automatically set to look in the current working directory for files with a "rcdf" appended to the name (e.g. run27.rcdf). The user can then load a RCDF for the test. Upon loading the RCDF, the program will check for the Run number and will alert the user if a directory with that run number exists. The program will always generate or use a run directory in the location it is called from. The data generated will always be of type 0 (i.e. xxx.0, where xxx refers to the channel label POT.0 for example). The program will always attempt to verify the CDRs by interrogating the hardware and making sure it agrees with the hardware_resources file. If it does not, an error will occur.

Load Profiles - This button loads any information needed to arm the traversing rings. To program the traversing rings one needs to use **travset**.

Initialize - This button will start the A/D zeroing process, and is always done before any type of data acquisition. This button can only be used after a RCDF is loaded

Arm DAS- This button arms the A/D and will display a dialog telling the user that the A/D is armed which the user must acknowledge. At this point the system is ready to accept any kind of trigger. This button can only be used after the system is initialized.
Trigger DAS - This button triggers the A/D through a GPIB command to the Camac modules (and thus it will not trigger other devices such as the traversing rings. This is primarily used for software verification. Once the system is triggered, either via this button or from some other source, a dialog will alert the user that the A/D has been triggered.
Record DAS - This button actually takes the data and downloads it to the computer. This is a critical step. Even after the system is triggered, all of the data is stored in the A/D memory and can be lost if the system is rearmed and triggered. Only when this button is pushed is the data downloaded to the computer and the raw data files generated. This button only becomes active after the system is triggered.
Abort - Completely aborts the entire dap process and allows one to start over. Similar to quitting and reopening **dap**.
Help - This accesses the on-line help system
Quit - This button quits the **dap** application.

Main Program Operation Notes:

dap is a fairly straight forward program since it takes the RCDF generated by **tsp** to acquire the data, but does not do any of the data reduction (done by **drp**). In theory, if one had a collection of RCDF's ready to go, then one could acquire data as fast as one could load the RCDF and initialize the hardware. There are really only two areas of concern:
1) When the system is initialized, it is establishing a baseline. Where this occurs is only important when data is acquired in relative mode. If data is being acquired in this mode, one needs to make sure that the physical situation (pressure, temperature, etc.) of those channels is not changing between when the system is initialized and when the data is actually acquired.
2) If a false trigger occurs, the system can be immediately rearmed by pushing the arm button (it does not need to be reinitialized).

Known Bugs:

None

Known Quirks:

None.

2.3 drp Program Notes

Program Name: drp.c
[Data Reduction Process]
Location: /usr/atarr/src/drp
Access: Typing "drp" in any directory that has a path to /usr/atarr/bin (or equivalent location)
Version: 2.0
Documentation Author: C. Haldeman, 1-1-95

Modification History:

None, Version 2.0 is the main one sent to ATARR.

Main Program Objective:

As Dave Yearke and Bob Meyer point out at the beginning of the code for this program, the program name is really a misnomer because, in general, after using this program one has a great deal more data than when one started. This program represents one of the four "wings"[1] of data reduction: that having to do with single file manipulations. This program allows the user to input one set of files and process it and view the results in one of three basic modes. The first is plot mode where the user inputs a file and can plot it. The second is reduction mode where raw data is converted to engineering units using a menu interface. The third mode is called multi-mode which allows the user to combine different channels using simple file manipulations to create a new file.

Main Program Interface:

Upon activation, the user is faced with a large screen containing a row of command buttons and a ten trace space. The command buttons perform the following tasks:
Save Resources - This button saves whatever files are active in the trace windows at the time. If one is about to do an operation which would overwrite an unsaved resource, the program will prompt the user to save the resources to whatever directory is active.
Plot mode - This button will make the program plot whatever traces are active.
Reduction Mode - This button will bring up a secondary display for single file operation (see below) to convert raw data to engineering units.
Multi-mode - This button brings up a secondary display for multiple file operations (see below)
Use English Units - This button will make the display read in English units (if the main calibration constants are in their appropriate metric units).

[1] See the technical note about data reduction in the DAS environment

Help - This accesses the on-line help system.

Quit - This button stops the **drp** application.

Secondary Displays:

Ten Trace Display:

This display contains ten individual traces. Each trace has its own color assigned to it. There are three command buttons and two non-modifiable displays.

Active (Command) - This allows the trace to be used in plotting or computations

2nd Y Axis (Command) - This selects the trace to be displayed on the second (Right side) axis.

Select (Command) - This activates the file selector and allows the user to choose a file to be associate with the trace.

Label (Display) - This shows the channel label for the channel selected for this trace

Type (Display) - This describes the type of data (raw, pressure, temperature, other).

This display is used as a basis for doing calculations and plotting. All traces which are active can be either plotted (Plotting Mode), reduced to higher levels (Reduction Mode), or combined to form a new type of file (Multi-Mode). These are discussed below.

Plotting Display:

This display consists of a plotting area, two user defined label areas, a legend and a command box on the right side of the page. The display is a standard X-Y plot. By selecting any "2nd Axis" button in the ten trace display, the graph becomes a double Y plot. The legend is automatically displayed (along with the run number). The program will not allow the display of conflicting data types on the same axis. The command box contains four command buttons at the top:

Draw Grids - A toggle which either turns on or off the grid display

Plot - A toggle that redraws the X-Y graph with the present information selected

Print - A toggle that sends the plot to a postscript printer[2] or the printer defined in the PRINTER environment variable

Help - A toggle that accesses the on-line help display for **drp**

Under these commands are four distinct sections. The first three correspond to controlling the X, Y, and Y2 axis displays. The user can either select auto-scaling or can use the following four lines to configure the scaling of the individual axis (looking at the X axis as an example).

[2] The printer is the default printer for the Sun computer which the program is being executed from. There are several ways to "trap" the postscript file and convert to Tektronix printing and vice-versa. The user is referred to the technical note "Printing with the DAS System".

X minimum - the minimum value used on the X axis. As a note, if one is using the fourth display control "X decade" this becomes the minimum significant digit (see example at bottom).

X Maximum - the maximum value on the X axis

X Delta - the difference between the grip points

X Decade - the power of ten that the scale is displayed to

Example 1: Display the x axis from 0 to 10 in increments of one.

Xminimum = 0

X Maximum = 10

X Delta = 1

X Decade = 0

Example 2: Display the x axis from 0 to 100 in increments of ten.

Option 1	**Option 2**
Xminimum = 0	X Minimum = 0
X Maximum = 100	X maximum = 1
X Delta = 10	Delta = 1
X Decade = 0	Decade = 1

Note that all three axis can be controlled separately. Often it is convenient to autoscale the X and one of the Y axis, and then manually manipulate the second axis so that the grids align.

The last section has modifiable displays for the symbol spacing and the averaging interval.

Symbol Spacing (modifiable display)- This controls how often a symbol is displayed on the trace. Every trace has its own unique symbol. The number in this box represents how often the symbol is plotted. A value of 1 signifies that every point has a symbol, a value of 1000 signifies that the 1000 data point plotted (see below) will have the symbol. The default value is zero which means no symbols plotted

Average Interval (modifiable display) - This controls how many points are averaged together to create a plotted point. This is generally set to zero, implying that this option is not used. This value will also affect the symbol spacing. For example, if one has a 1000 point trace and one wishes to plot every point and have the 10 symbols, "Symbol Spacing" would be set to 100 and "Averaging Interval" set to 0. If one wanted to average every ten points, and still have 10 symbols, only 100 points would be plotted, "Symbol Spacing" would be set to 10 and "Averaging Interval" set to 10.

Reduction Mode:

In this mode active channels are converted from lower orders (*.0 and *.1) to higher orders (*.2 and *.3) and will replace the lower data forms in the trace display (i.e. a raw data file which is active in trace 1 will become an engineering unit file in trace 1). This display consists of the following:

Reduction Algorithm (Menu)- This menu has all of the reduction algorithms listed in **direct.F**. When selected, the computer will try to reduce all the traces which are active to this new level. As an example **xxx.0** pressure files will be converted to **xxx.1** pressure files when the execute button is pushed (see below). The computer performs several checks and will not convert pressure type files to temperature, for instance. If the data is recorded in relative mode it will check to make sure all the required variables have been set in the RCDF. If any of these conditions fail, the computer will alert the user.

Calibration Constants (Menu) - This menu allows the user to select which calibration constant will be used in the conversion process for display only. There are four calibration constants which can be used: Pre-test, Post-test, Laboratory, and Used[3]. This is only important when converting from ***.0** to ***.1**. Only those files which have been converted to ***.1** files using Used calibration constants can be stored (Save Resources). This menu is primarily used to quickly see how different calibration constants will affect the conversion to engineering units. The default value is always the Used calibration constants.

Beginning and End Times for Statistics - This allows the user to perform rudimentary statistics on all active traces. This includes means, standard deviations, RMS values over the time period specified. The results are displayed in a pop-up dialog.

Execute - This button performs the operation selected, either statistics or conversion.

Multi-mode Display:

This display will allow the user to combine several different files into a new file, or to modify an existing file. The display is split into two main parts. On the left is the trace number with a modifiable weighting factor. On the right is several different pull-down operations:

Operators(Menu): This menu contains several possible, but self explanatory arithmetic operators.

Operands (Menu): This menu contains several possible items:

Constant - the constant displayed below

Tsupply, Psupply, Tinit, Pinit - these are values stored in the RCDF

Trace 1-10 - These are traces 1-10.

Constant Value (Modifiable display)- This is the constant value which will be used if the operand "constant" is selected.

Target Label - This is the new filename (should conform to DAS specifications).

Target Units - This is the new set of units for the newly created file

[3] The user is referred to the technical note "Calibration Constants in the DAS system" for more information about how calibration constants are inputted into the system, stored, and used.

Multimode works in the following manner. All traces which are active are automatically multiplied by their weighting factor and added together (call this result A). This result is then combined with the selected operand (Item B) using the selected operator in the form A & B where & is the operator. This value is placed in temporary storage as trace 11, where it can be plotted. However, it can not be used in a calculation until it is saved (using Save Resources) and then reloaded into one of the ten trace areas. Several examples may help.

Example 1: Average Traces 1, 3, and 4

Activate only traces 1, 3, and 4. Make the weighting factors all 1 for these three traces. Select "/" and "constant" in the menus, and place "3" in the constant display.

Example 2: Subtract Traces 1 from 3

Activate only traces 1 and 3. Make the weighting factor for trace 1 "-1" and "1" for traces 3. Select "+" and "constant" in the menus, and place "0" in the constant display.

Example 1: Average Traces 1, 3, and 4 an divide the result by trace 3

Activate only traces 1, 3, and 4. Make the weighting factors all 1 for these three traces. Select "/" and "Trace 3" in the menus.

Main Program Operation Notes:

drp's main strength is that it allows the user to quickly examine different combinations of single files, and print the results. It is not really useful at many repetitive reductions, or very complicated processes. These are better down using one of the other types of reduction programs.

Known Bugs:

If the autoscale buttons are deselected and one places minimum and maximum values which have a range which can not be rationally divided by the value in the delta spacing, the system may crash.

Known Quirks:

None.

2.4 fcp Program Notes

Program Name: fcp.c
[Fill/Calibration Process]
Location: /usr/atarr/src/fcp
Access: Typing "fcp" in any directory that has a path to /usr/atarr/bin (or equivialant location)
Options: -v Perfroms a voltage calibration
Version: 2.1
Documentation Author: C. Haldeman, 1-1-95

Modification History:

Version 2.1, December 1994 by Dave Yerke

This modification made some changes to the memory allocation procedure and set-up some defaults for the menu system. This has seemed to correct the relativley infrequent, but very annoying system crashes of earlier versions.

Main Program Objective:

fcp is probably the most complicated program in the entire DAS. It fufills several roles; allowing the user an interactive and on-line diagnostic of the instrumentation, a calibration system, and a procedure for recording the partial pressure and temperatures during the fill process. Basically all of these roles can be summarized as steps in taking data for calibrations. As such **fcp** can be thought as functioning in two modes (selected by the option when invoked). The default mode is for calibration of ATARR sensors and gauges. The other mode (-v) is for voltage calibration of the amplifier/digitizer system. These distinctions will be mentioned where they occur in this document. One of the key strengths of **fcp** is that not all of the steps have to be completed. One can examine data channels, without calibrating the sensors. One can calibrate, but not use the partial pressure option, or one can use the partial pressure marking system without calibrating.

Main Program Interface:

When **fcp** is invoked (see figure 1), the user is presented with a screen containing a series of command buttons, a space for entering a plot legend, a menu widget for determining calibration type and a large plot window for displaying data. The plot window will contain either a series of graphs showing the raw data acquired on the last data point or a single plot of a particular channel. The raw graphs are minimally anotated and are used to determine stability of the system. The main X axis is sample number and the Y axis is the distribution arround the mean value (similar to an AC data plot). If the user presses the left mouse button on one of these plots, it will be expanded to fill the entire plot window and the plot control parameters that are available on the "Plot Params" button will take effect. A second press of the mouse button will restore the original multi-plot window. If there are more channels being calibrated than can be displayed on one screen, the plots will be arranged in a multiple screen sequence that can be displayed by clicking on the "Previous Screen" or "Next Screen" buttons. The following is a summary of the command buttons:

Previous Screen - Displays the contents of the previous screen full of channels. If the user is already at the first screen, the system will wrap around to the last screen.

Plot Data - Simply redisplays the current screen of plots

Next Screen - Displays the contents of the next screen full of channels. If the user is already at the last screen, the system will wrap around to the first screen.

Hardcopy - Causes the fcp to store the contents of the current plot screen into a temporary file for later printing by the Flush Plots button.

Flush Plots - Sends the stored plots to the printer. It then clears the temporary file and prepares to receive more plot requests.

Plot Params - Brings up a pop up window with plot control parameters to be used for generating displays in the plot window. The controls will be explained in a separate section in this manual.

Data Params - Generates a pop up window with the data acquisition control parameters. This is where the user is allowed to configure the data system and acquire data for display and calibration. This window will be explained later.

Calculate Mode - tells the system that the user is finished acquiring data and wishes to generate calibration curves for the channels. Once this button has been pressed, there is no way back to acquisition mode.

Save Data - stores the generated calibration constants to the run channel descriptor file and makes entries in the sensor history files to indicate the new calibration constants.

Print Summary - Doesn't actually print anything. It generates a series of files named XXXXXXX summary where XXXXXXX is the label for each channel being calibrated. These files will contain all of the information acquired and generated during the calibration.

Help - brings up a help screen containing brief descriptions of the command buttons.

QUIT - exits the **fcp** program.

The very first part of **fcp** is acquiring at least one data point; either to examine the instrument stability, start a calibration procedure, or to save as a partial pressure point. In this phase of **fcp**, the program centers around the "Plot Params" window and the "Data Params" window.

The plot control parameters pop up window (see figure 1) allows the user to tailor the parameters that are used to generate the graphs on the display. Most of the parameters only apply to the single plot format and are controlled by the following modifiable displays:

Auto Scale X Axis - this toggle tells the plotting system whether to use the user supplied X-Axis parameters or to generate its own scale based on the data.

X Minimum - Minimum value to use on the horizontal axis when auto scale is turned off.

X Maximum - Maximum value to use on the horizontal axis when auto scale is turned off.

X Spacing - The distance between major tic marks on the X axis.

X Decade - when this value is non-zero, the axis minimum and maximum values will be divided by the power of 10 specified. This allows very large or small number to be represented reasonably. For instance, X minimum of 10, maximum of 20, spacing of 2 and decade of 2 would allow a range of 1000 to 2000 to be displayed as 10 to 20 with a decade indicator of 10**2.

These parameters are similar for the vertical (Y) axis. There is a third set of parameters for the second Y axis which are only used in calculate mode because the actual data points and curve fit are displayed on one axis and the deviation from the curve is displayed on the right hand axis. The next set of parameters control the display of the multiple plot display.

Draw Grids (command button) - when this toggle is on, grids will be drawn for the major tick marks on the graph. Otherwise, only hash marks will be drawn.

Units (Menu) - Allows the user to switch display units from volts to A/D counts, Metric or English engineering units.

Screen Rows (Menu) - Allows the user to determine the number of rows to use for the multiplot display. Three seems to be a good number to use here.

Screen Columns (Menu) - Allows the user to determine the number of columns to use for the multiplot display. Four seems to be a good number to use here.

Sigma (Menu) - Determines the range of the data to be displayed in the multiplot window in standard deviations. A sigma of "full" means to set the range of the plot to the minimum and maximum of the actual data.

OK (Command button) - Accepts these values and remove the plot control parameters pop up window.

Cancel (Command button)- disregard all changes made on the plot control parameters pop up and dismiss the window.

Help (Command button) - brings up a help screen containing brief descriptions of the command buttons.

Once the plotting display is set the user can set-up the data acquisition part of the program. In summary, data points are collected by arming the data system for trigger, sending a software trigger to the system and the downloading the data from the hardware. This process is controlled in the "Data Params" window (see figure 1). Specifically, everytime the system is triggered, the number of samples specified are acquired at the sample rate specified and are displayed on the screen. However, when generating calibration curves or the partial pressure information, these samples are averaged together to generate a single point. This improves the accuracy by averaging out the effects of spurious noise and EMI that might be introduced into the system. The default parameters that the system uses allow for the collection of 1000 samples over a one second period. A more complete discription of the "Data Params" window is supplied below.

Number of Samples Per Point (Modifiable Display) - The number of samples that the data system will record every time the system is triggered. These will be averaged to generate single calibration point values. Default value is 1000.

Sample Rate to Acquire Data (modifiable display) - The sample rate (samples per second) that will be used to record samples identified above. The number of samples divided by the sample rate will determine the sampling interval for the data point in seconds.

Keithley Range Number (Modifiable Display) - The range number that will be given to the Keithley 195A multimeter for its recording of the calibration standard. When the calibration standard is a resistive temperature device (RTD), the range will correspond resistance range, otherwise it will be voltage.

Load RCDF (Command Button) - read the run channel descriptor file that will be used for this calibration. It is assumed that the user has already set gains and filter frequencies in the RCDF. **fcp** will ignore the sample rate and number of samples in the RCDF in favor of the values that will be generated by the user.

Load Channel List (Command Button) - Identifies the file which contains the list of channels and the Keithley mode that will be used for calibration. The file should contain a line that identifies the Keithley mode and scale factor for conversion of voltage to engineering units followed by several lines which are merely the channel labels that will be calibrated. Note that these are case dependant, so if the user enters the labels in lower case, they will not match upper case channel labels. The acceptable entries for the Keithley are "KEITHLEYT" for temperature and "KEITHLEYP" for pressure.

Init H/W (Command Button) - autozeros the channels that will be recorded for calibration.

ACQUIRE (Command Button) - arm, trigger and record one sample interval. Number of samples and sample rate used will be as identified above. Upon completion of sampling, the system will display the first screen full of graphs. Each graph will contain the samples recorded from one channel. If there are more channels than will fill a screen, multiple screens of information will be displayed using the "Next Screen and "Previous Screen buttons.

STORE (Command Button) - average the samples for each channel and generate a calibration point. Hitting this button will also cause the raw data acquired to be stored in a .4 file. Each successive stored point will be saved in the raw data files and the averaged calibration point will be generated.

Mark PP (Command Button) - if the user is performing a fill of the supply tank, the pressure and temperature at this set point will be saved in the RCDF for the purpose of determining gas mix.

Mark FP (Command Button) - if this is a fill procedure, the current temperature and pressure will be recorded as the final fill point for gas mix calculations. The output of this calculation will be used for determining gas properties during the data reduction process.

Help (Command Button) - brings up a help screen containing brief descriptions of the command buttons on the data acquisition control parameters pop up window.

Dismiss (Command Button) - temporarily removes the data acquisition control parameters pop up window. Pressing the "Data Params button will cause it to reappear if needed.

Once the acquisition is done (and this may take several data points), the user can calibrate the channels if desired. This is accesed from the "Calculate Mode" button. This is basically a simle curve fitting routine, which allows some discression over which data points are used. The plot window will contain the current channels calibration points, the curve fit through those points and the deviation of each point from the curve. The commands on the calibrate mode pop up window are as follows:

Do Temperature - This button is currently non functional. It was to be used to filter out global calibration when performing a batch calibrate. It is currently not used.

Order of Fit (Modifiable display) - A numeric field to identify the order of the polynomial to be used to generate the curve fit. A first order fit is linear.

Coef 0 - Coef 6 (Display) -The coeficients generated for this channel. Note that Coef 0 is the constant value and Coef 1 - Coef 6 are the power coefficients.

Use Local Delete (Command button) -This toggle tells whether or not to ignore the points identified in the Delete Local entry when generating the calibration coefficients for this channel.

Delete Local (Modifiable display) - A list of point numbers to ignore when generating calibration coefficients for this channel. The list is a blank separated list of numbers. The first point collected is number zero.

Use Global Delete - Delete Global Same as above but applied to all channels.

Fit Constraint Point (Modifiable display) - If there is a point number in this field, the curve fitting routine will weight this point extremely high when generating the curve. This has the effect of forcing the fit to pass through this point. This is sometimes used to force the fit to pass through the zero position.

Fit This Channel (Command button)- Generate calibration curves for the current channel using the parameters specified.

Fit All Channels (Command button) - Perform curve fit over all recorded channels that are being calibrated.

Help (Command button) - Brings up a help screen containing brief descriptions of controls on the Calibration Parameters window.

Dismiss (Command button) - Removes the Calibration Parameters window from the screen. It may be brought back with the Calculate Mode button on the main **fcp** window.

Main Program Operation Notes:

As mentioned earlier, **fcp** is a very powerful program. Its real power comes from the many ways it can be used. One of the most important which will be discussed is is evaluationof instrument quality.

By using the "snapshot" capability of **fcp** one can track amplifier drift, noise, uniformity of conditions; all without having to disconnect a channel. This is very important for noise problems where ground loops may be caused or removed when a channel is disconnected. In this mode of operation, it is similar to having 128 oscilliscopes all connected to the DAS system. One also can track the voltage entering the A/D system and where the signal is on the digitizer scale. This helps sort out A/D problems before an experiment is run.

The short-coming of **fcp** is that it takes a great deal of human involvement. This is great when debugging, but for extensive calibration work it can become tedious. Thus one may want to use **monitor** for this type of work.

Known Bugs:

None

Known Quirks:

fcp is designed to calibrate sensors and display sensors in engineering units, and when it is invoked, even if it is just going to look at channels, the program thinks it is getting ready to calibrate and display channels in engineering units. It does this by looking at the constants stored in C_Lab. If there are no constants in C_Lab, it will generate an error like:

"data reduction broke" or "English reduction broke"

It will acquire data and display the data, but sometimes the labels are not correct (there will just be a long list of numbers). This is due to the fact that **direct.f** is returning an error. If one were to try to plot either English or metric units, one would get all zeros. If one just wants to look at the raw voltages, one should use **fcp -v**. This will eliminate any possible errors, and the labels should always be correct.

2.5 mm and travset Program Notes

Both of these programs are discussed in more detail in two separate reports. **mm** (which is the X-11 interface for the ATARR facility model) is discussed in detail in a report from Calspan "Operation of the Facility Model", 5-7-93 by Haldeman and Dunn. That report also includes several memos describing the changes which have been made to the facility model and as such, all inquires should be directed there. **travset** is the X-11 program used to set-up the traversing rings. It has undergone some changes since the first version was placed at ATARR and it is documented under the AHWT program (which built the traversing rings). The reader is directed to the AHWT Traversing Ring operation manual for more information. For completeness the present status of these programs is listed below

Program Name: mm.c
[mathmodel]
Location: /usr/atarr/src/mathmodel
Access: Typing "mm" in any directory that has a path to /usr/atarr/bin (or equivalent location)
Version: 2.0

Status:

This system has had the individual pieces checked-out at Calspan, but we do not use it here. The program has been at WPAFB for some time, and our impression is that it (or its derivatives) must be performing satisfactorily.

Program Name: travset.c
[traversing ring set-up]
Location: /usr/atarr/src/dap
Access: Typing "travset" in any directory that has a path to /usr/atarr/bin (or equivalent location)
Version: 2.2

Status:

This system has undergone some improvements since the traversing rings have been sent back to Calspan. The program allows more direct control over the traversing ring motors and the interface has been modified to make it clearer as to the meaning of the variables. Full documentation will be available under the AHWT project.

2.6 automate Operating Notes

Program Name: automate.f
Location: /usr/atarr/src/fortran
Access: Typing "automate p_file g_file a_file [e_file]" in any directory that has a path to /usr/atarr/bin (or equivalent location)
where:

p_file	**output**	**printed results**
g_file	**output**	**graphic results**
a_file	**input**	**algorithm entries (see below)**
e_file	**output**	**errors (If missing, "auto.err" is used.)**

Version: 1.0, September, 1994
Documentation Author: C. Haldeman, 2-1-95

Modification History:

None

Main Program Objective:

automate is a batch oriented data reduction program (see accompanying DAS Technical Note). Its main purpose is to allow fast reduction of multiple runs. It needs an input file and will direct graphical output to one user specified file, printed output to another, and will keep track of where errors occur in the process.

Main Program Interface and Operation:

automate requires no input other than what is in the invoking command line. The real key to automate is configuring the input file. The program can do many different kinds of processing, all of which are documented in the program, but are reproduced below for clarity.

Conventions/Restrictions on **automate**:

1. All files reside in the current working directory.
2. Algorithms operate on DAS reduced data files (*.1 or higher).
2. Errors are appended to "auto.err" if no file is specified.
3. Information is appended (not overwritten) to output files.

Each algorithm to be processed requires two lines of input. The first contains descriptive information and the second varies dependent on the algorithm number chosen. An example is given below. The first pair of input lines provides an average over the five pressure transducers in upstream rake 1. Pair two provides an average over both upstream rakes. Pair three provides an area average that requires the inner and outer radii of the test section plus the radius at each measurement. The last pair has the FFT calculation time interval (msecs) and frequency range (Khz) for plotting the indicated channels. In each case only primary files (.1) are processed.

Example File

Line#	Contents
1	Average Pressure for Upstream Rake #1
2	1 .1 10 20 PU1A PU1B PU1C PU1D PU1E
3	Average Pressure over Both Upstream Rakes
4	1 .1 10 20 PU1A PU1B PU1C PU1D PU1E PU2A PU2B PU2C PU2D PU2E
5	Area Average in the Test Section
6	2 .1 11.5 14.5 11.7 PU1A 12.2 PU1B 12.7 PU1C 13.5 PU1D 14.3 PU1E
7	FFT Calculation for Upstream Rake Pressure Channels
8	0 .1 10 20 5 10 PU1A PU1B PU1C PU1D PU1E PU2A PU2B PU2C PU2D PU2E

The supported algorithms are listed below. One can see that this system is easy to expand.

Algorithm Notation:

chan_i => i-th channel label; ext => extension for files/channels;
f1,f2 => frequency range (kHz); t1,t2 => time range (msecs);
gamma => specific heat ratio; t_test => test gas temp (K);
N D1 ... Dn => n+1 file labels for programs differ and ratios;
iunits => units system indicator (1=SI, 2=English);
i_diff => 1 for N-Di, 2 for Di-N; iratio => 1 for N/Di, 2 for Di/N;
numgas => gas indicator (1=Air, 2=N2); outfil => DAS output file;
ps => Pstatic channel label; pt => Ptotal channel label,
tin tex pin pex => adiabatic efficiency file labels.

Current Algorithms Supported:

# Program Function (Input Line) {Command Line}

0 runfft Performs FFT on indicated channels
(0 ext t1 t2 f1 f2 chan_1 ... chan_n)
{runfft ext t1 t2 f1 f2 g_file e_file chan_1 ... chan_n}

1 runavg Computes average, std. dev., peak to peak
(1 ext t1 t2 chan_1 ... chan_n)
{runavg ext t1 t2 p_file e_file chan_1 ... chan_n}

2 filavg Computes avg from entire time trace of given channels

(2 ext outfil chan_1 ... chan_n)

{filavg ext outfil p_file e_file chan_1 ... chan_n}

3 ratios Ratio(s) of average value in specified interval

(3 ext t1 t2 iratio N D1 ... Dn)

{ratios ext t1 t2 p_file e_file iratio N D1 ... Dn}

N.B. iratio = 1 for N/Di; = 2 for Di/N (i=1...n)

4 differ Difference(s) of average value in specified interval

(4 ext t1 t2 iunits i_diff N D1 ... Dn)

{differ ext t1 t2 p_file e_file iunits i_diff N D1 ... Dn}

N.B. i_diff = 1 for N-Di; = 2 for Di-N (i=2...n)

5 machno Mach # as a function of Pstatic/Ptotal

(5 ext t1 t2 gamma ps pt)

{machno ext t1 t2 p_file e_file gamma ps pt}

6 gam_cp Gamma and Cp for specified test gas and temperature

(6 numgas t_test)

{gam_cp p_file e_file numgas t_test}

7 adiabatic Scalar adiabatic efficiency from inlet/exit Ps and Ts

(7 ext t1 t2 gamma tin tex pin pex)

{adiabatic ext t1 t2 p_file e_file gamma tin tex pin pex}

8 adiabaticf File adiabatic efficiency from inlet/exit Ps & Ts

(8 ext outfil gamma tin tex pin pex)

{adiabaticf ext outfil p_file e_file gamma tin tex pin pex}

9 eff1 File adiabatic efficiency from inlet/exit Ps & Ts and Tzero

(9 ext outfil gamma tin tex pin pex tzero)

{eff1 ext outfil p_file e_file gamma tin tex pin pex tzero}

As a more complete example, a relatively short input file is attached. The goal of this automate run was to check-out the variation between reference junction temperatures for the various thermocouple rakes. The input file basically creates a collection of different averages. The output is also shown. The real benefit to this type of system is that one can combine all types of routine calculations, and process them exactly the same way for different runs.

Known Bugs:

None

Known Quirks:

None

2.7 monitor Operating Notes

Program Name: monitor.c
Location: /usr/atarr/src/dap
Access: Typing "monitor" in any directory that has a path to /usr/atarr/bin (or equivalent location)
Options:

- **-v Perform voltage calibration**
- **-e Perform an English unit Calibration (not used)**
- **-c Perform a calibration**

Version: 2.0, December, 1994
Documentation Author: C. Haldeman, 1-1-95

Modification History:

None

Main Program Objective:

monitor is a program used to acquire data over long time periods like a data-logger. The data is stored in an ASCII tab delimited file. The user sets the A/D rate and the number of data points, and the sweep time. The program will then at every sweep interval acquire the requisite amount of data at the prescribed A/D rate for all channels. The program will average that data and generate one value, which is sent to the output file. The program opens and closes the file every time, so that if the program is stopped, or the system fails, almost all of the data will be recoverable. The number of channels displayed is governed by a separate file, and is not governed by the active channels in the RCDF. In addition, the program supports a GPIB interface which is used with a Keithley multimeter which can be used to record a value separately from the A/D system and thus can be used to calibrate the overall A/D system

Main Program Interface:

monitor is not a X-window program and thus prompts the user for information it needs to operate. The information required by **monitor** is similar to any data acquisition program.

The first two lines ask the user for an RCDF and a channel list. The RCDF loads information needed to set the amplifiers, but the actual A/D conversion rates and which channels are active are ignored by this program. Channels which are not listed in the channel list file are not used.

The next five lines: interval in seconds, number of intervals, DAS sample rate, and number of DAS samples, all control the A/D system. The first two correspond to the time between sweeps, usually larger than 2 seconds. The number of intervals can be any number. Because the program write to a file, there is never more than one A/D pass in memory, so there is no limit to how many data points can be acquired. The user can also quit the program when operating it before the preset number of data point have passed. The second two properties correspond the A/D system and tell the computer how many samples to take and at what rate for each pass. The final parameter sets the range on the Keithley (which defaults to autorange).

Once this is accomplished the user is asked to start the A/D zeroing process and is given a chance to either rezero the hardware or to start acquisition. Once the program has started to acquire data, it will do so until either it reaches the total number of sweeps set at the beginning, or until the user aborts the process.

Main Program Operation Notes:

Monitor has three basic operating modes, which basically relate the data taken to the standard in different ways. The calibrate mode, (-c option) basically records the channel voltages and the Keithley standard (converting it to either pressure or temperature based upon the name). The voltage mode (-v), the Keithley designator (either P or T) attached to the end is ignored, and one gets both volts for the standard and volts for all the channels. This is very useful for calibrating A/D systems. The last mode (-e) uses the calibrations in the RCDF and creates engineering units for comparison.

Whenever **monitor** is activated, it will write the output directly to the screen. To save the output as a file one has to start the program and pipe the output to a specific location. This can be done using the command:

monitor -x > ***filename***

which runs monitor in **x** mode and pipes the results to filename. To view the file that is being created by monitor, in another window type:

tail -f ***filename***

Known Bugs:

None

Known Quirks:

When many channels are being calibrated, the display will wrap the lines so that every channel can be seen, but it may be very hard to figure out which channel corresponds to which reading.

2.8 Other Program Notes

Documentation Author: C. Haldeman, 2-1-95

Since January of 1993 until October of 1994, several updates have been issued about the general status of many of the smaller DAS programs. Each of these programs seem to be simple enough that no real specific write-up needs to be given other than stating what their specific purpose is. Many of these programs are discussed in the DAS technical notes; however for completeness, it seems useful to summarize those past memos. To aid in this, the programs will be loosely grouped into utility, calibration constant, data reduction, and data analysis.

Utility Programs

thinit-

This program is an update of one written for me by Bob Meyer several years ago. It allows the user to sort though several DAS files, pull out every nth point and put it into a tab delimited multicolumn format which is useful for moving to spreadsheets on the Mac or PC. This new program uses a standard John Moselle interface so the user can select the channels, time spacing and sample number.

plotraw.f

This program allows one to generate a large batch process of plots. It is intended to be used to generate a hard-copy record of the run data. There are various options, one can generate 1,2 or 4 plots per page, use either raw data (*.0) or engineering data (*.1), and generate either English or metric units. One can select the number of files to plot at once through a menu system. One note, when this program is run, it stores all the plot data as one file, which can be saved as a plot file to be executed later, or is plotted directly to the printer. If one were to take a large amount of data, this file could get quite large.

We have found that if one generates several plot files, each containing 10 - 20 channels, one can then send these plots serially to the plotter without causing too much of a backlog. One can view the plot file on the screen by typing the command **draw xxx**, where **xxx** is the file name. One can send the plot to the printer after it has been created by typing **iplot xxx**.

modcdr.f

This program allows one to change one CDR value in a whole bunch of files within a run. This program was created when I used an RCDF with a wrong value. This process operates by looking for a file name and the new value. At the top of the input file is the location CDR parameter name that needs to be changed. The remainder of the file has lines containing a channel label and the new parameter value separated by a tab (Note: **modrdr** performs a similar task).

rcdf_diff.f

This program will compare two different RCDF's and show what the differences are.

cpyrdr.f

This program will actually copy an RDR to a new RCDF. It is used with our data lab system to update the Datalab directory.

listdas-

This program has been modified to accept different output format types. these format types are invoked by typing **listdas -x file1 [file 2 file 3 ...]** where **x** is the option type. Option types:

1 (or just listdas) - provides a full printout of the RDR and CDR for each channel

2 - Provides a listing of the calibration constants and a few other parameters

3- Provides a listing of the thin-film heat-flux sensor properties.

Presently, there exists the capability of installing two more formats (-4, and -5) which right now default to the same as typing **listdas**.

<u>Calibration Constant Programs</u>

fitmon-

This program will now let the user select subintervals for calibration. As an example if all the transducers in the model are 100 psi capable, but you have some which you believe will see 70 psi (such as inlet transducers) and others that will only see 20 psi (such as exit transducers) you might have a tendency to have different gains on the transducers. You could still calibrate all of them to 70 psi without damaging the sensors, but the ones which have the higher gain may saturate at 40 psi. With this new option, you can select which gauges get calibrated over what range. The actual calibration range is now displayed in the print-out.

In addition, the user can now plot either the standard deviation plots, or the actual data and fit. This is selected in the program by a toggle which asks for the proper X-axis as either data (which is the deviation plot) or the voltages (which is the fit type of plot). The plots can also be sent directly to a plot file, to be viewed (using **draw**) or plotted (using **iplot**) at a latter time, or they can be interactively plotted.

ccp-

This program has been modified so that one can enter the heat flux gauge run resistance from a file. This operate a little differently from the file entry for the calibration constants. The file should have the gauge labels, separated by white space, from the run resistance. This process alleviates a great deal of typing. At this point all of the calibration constants (pre, post, lab, and the heat transfer properties can be entered via hand input or files. These calibration constants are also written to ASCII files in /usr/atarr/files. In addition to all the listed commands in the program there are two others which are not listed and which are used primarily for relative recorded data:

"cntrl - b" This allows the user to set the baseline times in the RCDF
"cntrl -s" This allows the user to set the time over which standards are used. It also sets the standards flag and allows the user to continue processing. Thus if the standards time is not an important issue, a setting of 0, will allow data reduction to continue.

selcon-

This program is straight forward and allows the user to select which type of calibration constant should be entered into c_used.

Data Reduction Programs

reduce.f

This program is used to provide a capability of reducing large amounts of data in a batch format. **It is only intended to be used after one has intelligently determined the calibration constants; otherwise it's GIGO!** Given that warning it can be used to bypass the reduction part of **drp**. Once the calibration constants have been loaded into C_LAB, the program examines the input files (which can be user selected), checks the sensor type and acquisition mode for consistency and automatically converts from *.0 to *.1 files. Afterwards, the user can use **drp** to examine these files in more detail.

Data Analysis Programs

do_fft-

This program allows one to perform an FFT on any number of data channels, and plot the results. Data can be plotted directly to the screen or to a file. This program has been updated to display the y-axis as either a percentage of the maximum harmonic value, or as a straight amplitude. In the past versions there was a mistake in the calculation of the Y-axis and as a result it was not displaying the proper units. As of now, if one plots the actual (unscaled) FFT, the Y-axis represent the units of the input file (i.e. a pressure file will have units of kPa). If one plots the normalized FFT, the Y-axis represents the frequency component divided by the largest harmonic. There are a couple of important things to remember:

1) Even if one selects a frequency sub-interval, the maximum harmonic is taken out of the entire frequency domain.

2) Usually this is the 0^{th} harmonic, but it is not necessary so. For instance, a 4 unit amplitude sine wave offset from 0 by 1/2 would be normalized by 4 and not the 1/2.

3) The maximum value and frequency location is listed at the top of the plot, even if the frequency is not in the specified range.

das_filter.f

This program is invoked simply by typing the name (as all DAS programs). Its purpose is to apply different types of filter to the data. The user can select which data files are to be filtered, what type of filter to use (low frequency cut-off, high frequency cut-off, band-pass, and band-stop), and what the appended extension should be. Files are named by taking the original name and appending the extension to them. The program will also plot the characteristic filter properties.

pinned.f

This is a funky little program that allows the user to specify lower and upper bounds on the data. The program reads in the RCDF from the directory in which it is called and will examine all *.0 data to see if any of the data is beyond the range specified. The system is a greater than or equal definition, so having a lower limit of 0 and an upper limit of 4095 (in a 12 bit system) would list only those files which are actually pinned. One can of course change these limits to any range desired. This allows one to examine only those channels which are having some problems. This program is especially useful to check for transducers which have gone bad.

timebase.f

This program will allow one to convert from asynchronous sampling to synchronous sampling, and vice-versa. It will also allow one to change clock rates in an asynchronous frame. All that is required is an indexer file.

ensemble.f

This program is the ensemble averaging program and has several options.

1) It can do either of two stages

2) It can do either a vane average (as if you are sitting on the blade, watching the vanes go by) or a blade average (sitting on the vane watching the blade go by)

3) It can perform either an ensemble average (the same point on the circumference averaged with itself), a blade map (all the points in a revolution, 1000 for ATARR, moved so they line up on one passage), or a blade average (an averaging of a blade map)

4) The user can select whether one wants to average over a specific timer period or a number of revolutions

5) The program will correct for circumferential displacement of transducers. As an example, if you have two pressure transducers mounted on the rotor, but they are on different blades, clearly at any given time these transducers may be in different parts of the vane passage. The program will correct for this relative shift. It does this by examining gxpos in the CDR for the blade number. The blade number has nothing to do with the serial number of the blade. It must conform to the standard set out on the attached sheet. Basically, the program assumes that the once/rev mark occurs when a rotor leading edge is aligned with a vane trailing edge. This location is called relative dead zero (RDZ). The blades are numbered starting with blade 1 and moving in a counter

rotating direction. Another way of looking at this is that if one were to sit on one vane and watch the rotor go by, one would see blade one first, then blade two, etc. The same holds true for the vane numbering. Except in this case the vanes start at 1 and are labeled in the rotating direction (since if you were to sit on the rotor you would see vane one, then 2 go by). Thus to use this program one needs to know

A) Which blade each gauge is on relative to the once/rev mark

B) The angular offset of the once/rev mark

A program which is very helpful for this routine is **stages.f** which will prompt the user for the number of stages, number of vanes and blades and will draw the vanes and blades with the appropriate number.

2.9 The DAS Program Structure and History

By: Charles Haldeman
Date: 1-26-95

Overview:

This technical note provides a much needed overview of the entire DAS system and how the different programs relate to each other. It will focus primarily on the relationship between programs, rather than on how the individual programs operate or what the specific concepts used in the DAS are (such as a calibration file, a reduction toggle, or a run log). These can be found elsewhere in the DAS documentation. There are also some historical issues that are important so they are reviewed at the beginning of this note. As additional references, the reader is referred to "Other Program Notes" another technical note which summarizes the upgrades made to many of the smaller programs (which have been documented and sent to WPAFB over the years) and to the larger program write-ups in "DAS Program Notes".

Historical Issues:

The ATARR DAS system has undergone one main revision. Version 1[1] never really saw much use. It was changed from a "turn-key" operation where the engineer basically pushed a button to acquire the data and then pushed another one and data in not just primary units, but also non-dimensional units, was generated almost instantaneously. While this type of system sounds wonderful, it would have been a complete failure as a research tool, since it would have limited the type of reduction which could occur, and it could create the possibility of data misinterpretation. Version 1 was not wasted, its main pieces were integrated in slightly different ways to produce version 2. In the process there was a change in the RCDF structure which basically forms the dividing line between version 1 and 2. While version 2 may seem much more complicated than the original "turn-key" system, it has been a remarkably flexible system, allowing easy expansion.

For many reasons the ATARR program ran very tight on money and in addition to cutting out almost all documentation, the main job of checking out the DAS system was moved to Calspan where it could be done on other projects and developed further. When Calspan left ATARR in December of 1992, the main code was operational, but many of the original options envisioned for the DAS had not yet been implemented. During the course of subsequent measurement programs at Calspan, version 2 was refined, some small program corrections were made, but several new

[1] It was not known as version 1 until we had created version 2 and as such there are seldom any references to that particular system.

capabilities were added. By the fall of 1994, by mutual agreement with WPAFB, Calspan delivered its last code update, which was the final version 2 software. WPAFB had changed its directory structure making it hard for a direct transfer of code and was busy modifying its own code (which is what was meant to happen). Calspan has since started the transfer of its code to version 3 which is more aligned with the Calspan operational characteristics.

Version 2 in its present form represents an extremely flexible system. It has both the capability of running very complex research programs involving specialized data reduction schemes, or it can run testing programs where things do not vary much and data reduction can be done on line. Modifications to the main data reduction software can be made relatively easily. The entire system can be thought of in many different ways. The ones that we can think of are:

a) The division in the source code between **C** code, **X-11** code, and **FORTRAN** code.

b) Operational phases of the testing program: test set-up, data acquisition, calibration verification, utility, and data reduction

c) Types of data reduction

Each of these separations helps describe the overall system, and so they will be pursued individually.

The original ATARR DAS structure is shown in figure 1. In every main directory there was a Readme file placed to describe the overall configuration of that directory.

Figure 1: Original ATARR DAS Structure

```
                ┌── /bin  [these contain copies of all the executable programs]
                │
                ├── /files [ this contains the hardware resources file, the sensors.list file
                │          and all the calibration files]
                │
                ├── /include [this contains all the include files used in all the programs in src]
/usr ──── /atarr│
                ├── /lib [this is the main library functions for the ATARR programs]
                │
                ├── /man ──/man1   [this contains the on-line help files for the X-11 programs]
                │
                └── /src ┬── /aquire  [data acquisition code used with datalabs]
                         │
                         ├── /dap [data acquisition programs]
                         │
                         ├── /drp [data reduction programs]
                         │
                         ├── /dsp [all the base code used for controlling the DSP equipment
                         │        copies are found in the appropriate directories]
                         │
                         ├── /fileselect [ this includes the main code for the file selctor widget]
                         │
                         ├── /fortan [this has all the fortan data reduction code]
                         │
                         ├── /graph [contains the code for generating techtronix plots]
                         │
                         ├── /graphics [contains some X-window stuff]
                         │
                         ├── /mathmodel [contains the programs used to create the ATARR
                         │              facility model]
                         │
                         ├── /runchan [has the information about the run/channel descriptor
                         │            records]
                         │
                         ├── /tsp [contains the information about the test setup process]
                         │
                         └── /util [contains extra utilities used by the ATARR DAS system]
```

Types of Source Code:

The source code is separated into three, not so distinct groups: the C code, the X-11 interfaces, and the FORTRAN code. To make matters worse, sometimes there are FORTRAN structures within the C code, and the X-11 code can call either FORTRAN or C subroutines. The reason for this is relatively simple. Much of the main hardware control routines were written in C. To conserve costs, many of the data reduction codes were taken from the original Calspan versions with only slight modification. Each type has its own particular strengths.

C- This is the language primarily used by systems administrators, UNIX, and advanced computer hardware. Because of the type of system chosen, it was a natural selection for the main operating code

FORTRAN - This is the language universally understood by engineers, and is generally the language used for the more complicated numerical computations. Many of the programs are written as subroutines. There are some major programs also written in FORTRAN (**ccp, selcon, thinit**, etc.) which use a fairly standard interface (courtesy of John Moselle). While not as pretty as an X-11 interface, it can run on any machine and is more easily modified. The interface does help direct the user to some degree.

X-11 - This interface provides a simple point and click interface for the data reduction system. Ideally, a good designer can make the system self-explanatory and very easy for a person with no familiarity with computers to run the entire system. The downfall is twofold:

a) This requires a very good X-11 programmer (hard to find)

b) Programming the interface takes a great deal of time

Originally, Calspan built four main X-11 interfaces: **tsp, dap, drp,** and **fcp.** It took so long to update these that no more were built until we decided to have a fully operational FORTRAN versions of the code called by an X-11 interface (this was done with **mm**, which calls the facility model). We presently also have one other X-11 interface called **travset** which controls the traversing rings.

Basically, all the main code used to set-up the tests and acquire data are X-11 based (except for **monitor** which was built primarily as an afterthought, but has become very important in the calibration systems). Most of the data reduction code is FORTRAN based. Since the majority of the data acquisition code should not need to be changed, it can be left alone. The FORTRAN codes are usually the ones that undergo the greatest changes, and luckily this is a language which is well known. For those individuals wishing to learn more about X-windows, there is always the possibility of taking the FORTRAN code and creating an X-11 interface for it.

Operational Phases of the Testing Program:

A second way of looking at the DAS system is to separate the many different programs based upon their general areas of use. There are basically five specific areas, although one could make an argument that some of the smaller programs belong in many areas. These are:

1) Test set up
2) Data Acquisition
3) Calibration constant development and selection
4) Utility programs
5) Data reduction

Test Set-Up

This phase of the process centers around allocating the channels, setting up the acquisition rates, filters, timing sources, etc. The main centerpiece of this part of the process is **tsp** which is the X-11 interface that allows the user to set-up the channels, and also allows the viewer some access to the RCDF after the run. In addition to **tsp**, there are several other programs listed below which can be useful in setting up a test, or in manipulating or viewing the RCDF.

facility or **mm** - **facility** is the FORTRAN implementation of the MIT math model (with some additions from Calspan). **mm** is the X-11 interface for this program. The operation of this program is well documented[2]. Basically this program can be run in two modes, one is to predict the performance of the facility, and the other will calculate the actual gas properties based upon the partial pressures and will store the required properties in the RCDF.

viewdas - This program allows the user to examine the complete RDR or CDR of any file in the system and make changes. Thus it is quite dangerous, since it can create a situation where the natural connection between the RCDF and the RDR and CDRs is broken.

listdas - Is the preferred way to examine the RDR and CDRs of files since one can print the values out in a variety of formats and the user runs no risk of inadvertently modifying the RCDF or RDR and CDRs.

modcdr - This program is a simple utility which allows the user to systematically change a parameter in all the CDRs in a particular run directory. It is seldom needed unless one makes a major mistake, such as not inputting the series resistor for the heat-flux gauges for a particular run (which is what I did, and is why the program exists).

modrdr - This is the equivalent of **modcdr** but it operates on the Run Descriptor Record.

travset - This is an X-11 interface which is used to control the traversing rings and the LED groove information. This program creates as an output, a file which can be loaded by **dap** to initialize the traversing rings, and a data reduction program **traverse** which can convert LED pulses into position data.

Data Acquisition

This part of the process consists of three main programs.

[2] Haldeman and Dunn "Operation of the Facility Model", 5-7-93. Binder contains this memo and copies of the technical memos which were sent from Calspan to WPAFB relating to changes made in the model.

dap - This is the main program used during measurement programs. It is an X-11 interfaced program although it is fairly straight forward in the arming of the DAS and recording of the data.

fcp - This is a secondary X-11 program which has many uses. Its primary description would be that it takes a snapshot of the A/D system at any point in time and can display the results. This can be used to record the partial pressures of the supply tank, look for noisy channels, and calibrate the A/D system. In addition to taking data, it also has the capability of processing the data into a set of calibrations and storing those results directly into the RCDF and the calibration constant files.

monitor - This is a FORTRAN based program which is used to acquire data over long periods of time. Its primary goal is one of either monitoring channels or calibrating channels. The program works by arming the A/D and taking short bursts of data every sweep and then averaging all that data together to generate one time point. The output is ASCII, tab delimited files with the channel names across the top and the values of every channel listed at every sweep point. The program can run at sweep rates from about 2 seconds on up. This file format is easily transferable to PC's for manipulation.

Calibration Constant Development and Selection

This part of the process is described in more detail in a separate DAS technical note (Calibration Constants and their Path Through the DAS). However one can look at this process as having two separate parts: one of entering the calibration constants into the DAS system or generating the calibration constants, and the other of selecting which ones to use.

Calibration constants can be generated either within the DAS system, or external to the DAS system. Only two programs work internally in the DAS system, **fcp** (which was described above) and **standards**.

standards - This program was originally designed to help automate the selection of the calibration constants and to create some "standard" reference values needed by some other data reduction codes (such as the initial supply tank pressure). It is not used at Calspan, but should be more useful to ATARR.

Calibration constants often are generated outside the DAS system. **monitor** is a good example of a program which acquires data which needs to be reduced in some manner. Other times the calibration constants come from a completely separate experiment or from the instrument manufacture. There are some programs in existence which process data taken by either **monitor** or the DAS system and create calibration constants, but do not automatically place them in the RCDF or calibration files.

fitmon - This program takes a standard monitor output file and provides a method for doing a polynomial regression. The output is also tab delimited and can be easily read to check for problem sensors.

thermocp - This is a relatively old program that allows one to calibrate a thermocouple from data taken in the DAS. Realistically, there are better ways of calibrating thermocouples and it is not used anymore.

psia2kpa - This program takes the output from **fitmon** and assumes that the calibration constants are in psia and converts them to units of KPa. This is mostly a Calspan need since all of our calibrations are done in psia.

cvtcal - This program takes any line of data which contains a channel label and the calibration constants (such as from a spreadsheet) and coverts that into a format readable by the DAS system.

Once the calibration constants have been obtained one must input them into the DAS system. This is done through **ccp** (or Calibration Constant Process). This program allows the user to input the calibration constants in a variety of ways, as well as select the appropriate baseline times and allows the user to activate the standards flag (allowing data reduction).

ccp - This program allows the user to either read in a data file filled with calibration constants or input them manually. The program also writes the calibration values to the calibration files, and will update *.0 files.

Once the calibration constants are entered into the system (i.e. stored into either C_lab, C_pre, C_post, or the thin film properties), the user must select which ones to use. This has been made to be a conscious effort on the part of the engineer, so there is no excuse for using the wrong calibration constant inadvertently.

selcon - This is the main program used to tell the DAS which set of calibration constants need to be placed in C_used. Only the calibration constants in C_used are used to reduce the data. If this step has not been performed, the computer will either not do the reduction (**drp**) or will create only 0's (**reduce** and **automate**). In addition, if data is acquired in relative mode, data reduction will not occur unless a timebase is specified and the standards flag is activated.

Both **ccp** and **selcon** interact with the RCDF and *.0 files to update the RCDRs. They also access the calibration files which provides an ASCII type file which can be used to trace calibration histories.

Utility Programs

There are a whole collection of programs which do not belong in any of the other categories and which provide some functionality to the DAS system. Some of these are just interfaces used to call subroutines which were created during checkout, and have been kept just in case they are needed in the future and will not be discussed here. The other programs can be separated into a few different areas, as discussed below.

Printing/Plotting

plotdas - This program plots DAS files using Tektronix emulators (i.e. does not require an X-terminal). Similar to **drp** but without the data reduction capabilities.

plotraw - This program allows a batch printing job to be executed. One can select the time-range, X axis type (samples or time) and the number of graphs per page. Very useful to get a hard copy of a whole set of files. However, the printer usually takes a while when processing all the data points.

plotvar - creates a plot of a VAR formatted file

draw - Creates on the screen using Tektronix based routines, plots from a file. This is very useful since many of the data analysis programs generate large print files which can be examined on the screen before being sent to the plotter.

stages - This routine plots the blade and vane numbers for any stage. It is useful for figuring out which blade an instrument is on when running ensemble.

Format Conversion

thinit - This program converts any number of DAS files into an ASCII file consisting of a series of multicolumn values. The user can select the number of files, the data range, and how many data points to include. This is very useful for processing data on another machine (such as a PC). As an example, if one would like to display the time lag of a propagating shock on the pressure side of a blade, one might include just those sensors on the blade, and look at 20 ms of data, which might correspond to 10 sensors at 2000 data points each (which is very manageable).

timebase - This program converts a DAS file in one timebase (either synchronous or asynchronous) to the other base. It also allows asynchronous data to be changed from one recording period to another.

v2dfmt - Converts a file from VAR format (readable by the outside world) to a DAS file

d2vfmt - Converts a file from DAS to VAR format

Hardware

hwcheck - This program interrogates the A/D hardware and compares it to the hardware resources file. It is very useful in finding out what the computer thinks it has installed.

fixit - This program reinitializes the system (a similar program is **gronk**, which is a Bob Meyer utility).

Data Reduction Programs

There are many types of data reduction/analysis programs, most of which will be discussed in the next section of this technical note. However there are a few "utility" reduction programs which do not fit well into these categories so they will be reviewed here.

sweep - This program runs through a series of user specified files and generates statistics on the data over a specified time range.

pinned - This program will sweep through an entire run and will generate a list of files with an A/D count outside the specified range. This allows the user to quickly pick-out instruments which may have developed problems during a run.

traverse - Takes as input a file generated by **travset** (usually a *.trav file) and will use that information to convert LED pulses to position

das_filter - Allows the user to create various types of filters to be used on the data.

do_fft - Allows the user to perform fast fourier transforms (fft's) on data files

ensemble - This is a complicated program which allows the user to investigate unsteady phenomenon.

set_tc - This program is used when reducing thermocouple data to set the reference junction channel.

acq - Single or double-sided heat-flux reduction code

preacq - Creates sample time-temperature traces for verifying **acq** operation

Data Reduction Styles:

When version 1 of the DAS system was planned, there was only one data reduction style - fully automatic. This would have caused quite the problem in doing any kind of new research, or diagnosing problems. When version 2 was first delivered, the emphasis was on single file manipulations. This provided the most flexibility, but takes the most amount of time. The full DAS system was always supposed to have more data reduction paths than just this one, but ATARR funding constraints precluded this development. However, as a result of other Calspan programs, the DAS capabilities wave been expanded significantly and have been shared with ATARR. At this time there are four modes which can be used and which will be described below. Many of the same programs can be used in different modes, and it is only important for the engineer to understand the options available to optimize performance based on the specific research experiment.

Single File Manipulations

This is the original system and consists primarily of **drp** and **direct**. **drp** is the X-11 interface which allows the user to load in specific files, plot the results, convert to higher units, or do some basic single file manipulations. There is a menu driven system for doing algorithm conversions based on the algorithm number and sensor type. This provide an excellent format for "smart" reduction, because it is easy to provide the checks in the system to make sure no one tries to reduce a temperature channel with a pressure algorithm (for example). **drp** gets its algorithm information from **direct**. **direct** has one problem, it can only handle a certain number of inputs. The entire system is very useful for engineers examining data in a simple way, or checking the quality of data, or printing out some simple double-Y axis plots.

There are a couple of short comings to this system. The first is that adding algorithms does require a reasonably good programmer, conversant in FORTRAN, C and X-11. And while not impossible, it is not something you want to do on a haphazard basis since it interrupts so many programs. The program is meant to have algorithms added to it, but only after they have been determined to be useful and needed. Secondly, the overhead associated with loading every file, activating the traces, etc. becomes very slow when processing large amounts of data. These two problems lead to two other ways of processing data.

Single File/Complex Manipulations

To alleviate the first problem, a second program was created called **quickmod** which basically allows the user to add a complex algorithm and process the data without the restrictions of **direct**. This is all done in FORTRAN and is well documented in the program **quickmod**. Basically the user creates a simple subroutine, which **quickmod** will access. The user can then experiment with different types of algorithms relatively easily. The main program, **quickmod**, takes care of reading in the files and writing the files so that overhead is eliminated. Thus most engineers should be capable of adding these types of algorithms without any extra support. This is the preferred method for verifying new algorithms.

Multiple Files/Interactive

To alleviate the second problem associated with **drp**, an interface was developed which is on most of the main reduction programs (**do_fft**, **ensemble**, etc.) which allows the user to either load in a series of files, or create a series of files from the RCDF, and then process them all in a similar manner. As an example one can create a 40 Khz low pass filter and use it on a whole selection of files. The new files are named by adding an extension to the input file. For instance, a series of *.1 files processed with a filter may be called *.1.flt. These names are all controllable by the user.

Another program that falls into this category is **reduce**. This program allows the user to reduce all data selected from *.0 to *.1 files. However to do this the proper flags must be set in the RCDF and the proper calibration constants must be loaded into C_used. If these are not done, then the resulting output will always be zero.

Multiple Files/Automatic

This last style allows a full automatic reduction of the data to some specified level. It starts with *.1 files and proceeds from there. The program that controls this is **automate**, and like **quickmod** it is relatively easy to modify. This program is best used with simple data reduction tasks which are the same for multiple tests in a matrix. When configured correctly, this can give almost instantaneous verification of the quality of the data. Presently, there are several algorithms supported by **automate.**

adiabatic and **adiabaticf** - Calculates the adiabatic efficiency as either a scalar or as a time trace based on the upstream and downstream total conditions

differ - Calculates the difference between data files

eff1 - Calculates the adiabatic efficiency based on the difference between the upstream and downstream conditions normalized by a difference reference value

filavg - Calculates the average value of many different input files as a function of time

gam_cp - Calculates the gamma and Cp of a gas based on inlet conditions

machno - Calculates the Mach number based on static and total pressures

ratios - Calculates the ratio of two files or scalars

runavg - Calculates the average value, standard deviation, etc. of a file over a specific time range

runfft - Performs FFT's on input files.

automate is run by providing it with a command line file (as described in the program). The output can be either plots, or ASCII values, and the computer will generate an error file if problems arise. While it does take a little while to generate the first input file, this file can be used on all the test matrices. Major reduction of test data has been reduced to 1-2 hours using this method when appropriate.

2.10 Using "**Traverse**"
(and other notes on position measurements in the DAS)

C. Haldeman
10-19-94

Introduction:

During construction at ATARR, a capability of measuring position relatively easily on many different devices: traversing rings, valves, actuators, etc. developed. Because of the relatively high cost of encoders, and the relatively benign environment that they must operate under, a decision was made to use optical LED's and write a piece of software that could convert recorded pulses to position information. The result of this was a program called **traverse**. In essence this program takes as input either one or two LED pulse traces, some information about the geometry of the grooves which are generating the pulses, and creates a position file (a ***.3 file). Much of the present program interface and capability are the result of the specialized uses that the program has seen over time (main valve motion, traversing ring motion, etc.)

As it presently exists, the geometry data is created by another program called **travset** which is an X-11 interface program used primarily for the traversing rings. The purpose of this memo is to describe in more detail the options of the program **traverse** and discuss how it converts pulses to position.

Traverse Options:

There are really three major input options.

1) The program can accommodate blanked off groove positions. This means that if you want to know the position of your item accurately at some set location, you can place a piece of tape over a particular groove. This type of system is equivalent to having a once/rev marker on an encoder. This option was used extensively early in the design phase of the main valve and with only one LED. But with the increasing usage of two LED systems this is not as important. The information about the number of grooves, space between the grooves, and the blanked-off grooves is stored in the program **travset**.

2) The program can also predict motion based on a set of stored parameters in **travset**. Presently this is limited to trapezoidal motion associated with the traversing ring, but that can be easily increased.

3) The program can accommodate either a rotating or linear frame of reference (also set in **travset**).

In terms of output, the program generates the following files:

profiles.info - This is an ASCII file of all the leading and trailing edges of the pulse trains which is extremely useful in debugging the system
xxx.3.ascii - This is a file based on the LED name (**xxx**) which contains the position and time of each pulse (or for a two LED system, the leading edge of a quadrature pulse).
xxx.3 - This is a DAS format file of position vs. time. The program basically takes the pulses shown in **xxx.3.ascii** and linearly interpolates between them to fill in all the times.

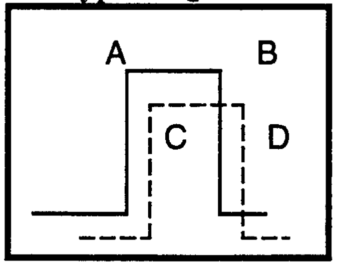
Main Program Calculations:

The program, while quite long is divided into four main sections. The sections are the same for both single and dual LED systems. The dual LED system is more complicated since one can determine direction, and that system will be the one discussed here, since the one LED system is a simplification.

1) The first is the book-keeping section which loads in the ASCII file created by **travset**.

2) The second part is one where the pulse leading and trailing edges are found. This is in reality composed of two smaller pieces, the first of which is the setting of the threshold value, and the second is the scanning of the LED channels to find the leading and trailing edges. The program is robust in that it does not need any particular type of data input (counts, volts, whatever). It only assumes that the LED channels have clearly defined pulses. The program scans the data and finds a high and low value for each channel. Presently the threshold value is set at 20% of the distance between the low value and the high value. When the voltage level is over this value, the signal is considered on, and below it is considered off. Figure 1 shows the typical quadrature pulse.

Figure 1 Typical Quadrature Pulse

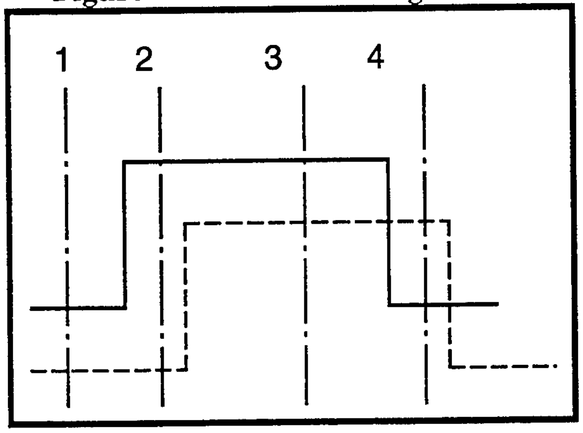


The dark line represents the leading pulse train, and the dashed line is the trailing pulse line. Points A and C are the leading edges and points B and D are the trailing edges. For this type of system to work well, it is important for both sensors to see the same groove. For space reasons, one might think about offsetting the sensors by one or more grooves. When that happens, the manufacturing tolerances of the grooves can lead to false pulses. Basically the file **profiles.info** is a listing of the points A, B, C, and D for each pulse sequence.

3) The next part of the program is used to determine which way the device is first moving and its initial state. If we assume that the mechanics of the device are such that there is no noise, and

that good signals are being generated (which is usually true), then there are four types of initial data (see figure 2).

Figure 2 Initial Starting States



State 1: Both traces start low. This is relatively easy to handle since the next four data points (A, B, C, and D from fig. 1) form one sequence. This makes it is easy to determine which way the system is moving.

State 2 : One trace is high, and the other is low and goes high before the first one goes low. This is also relatively straight forward since the computer recognizes this pattern as a traditional pattern (State 1), since it does not need a low before it can go high.

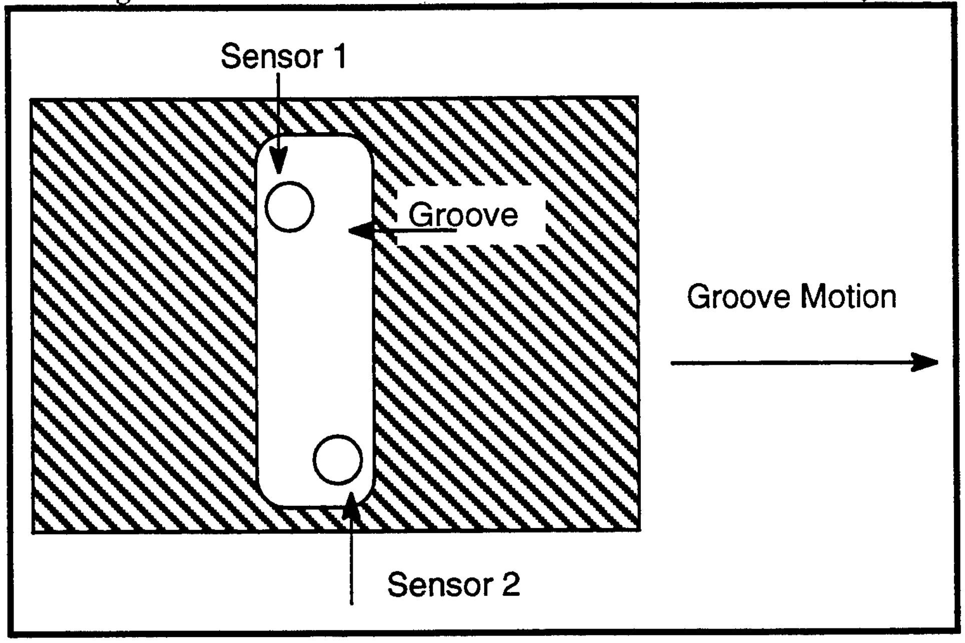
State 3: Both traces are high. The computer cannot figure out the direction until both pulses go low. It then looks at the different times the two traces go low to figure out the motion direction. However, like state 1 and state 2, the computer recognizes this as one sequence.

State 4: In this case one sensor is high and then goes low, and the other sensor never goes high. This case poses the largest problem because the computer deals with four numbers, but there is a mismatch. In this case, points C and D are for the first pulse on the second LED, and A and B are from the second pulse on the first LED (the first not being captured by the A/D). In this case, the computer recognizes that trace two (dashed line) is high and has gone low before anything happens on trace 1 (solid line), indicating that the first pulse is by itself and the computer ignores the first pulse information from trace 2, and looks at the next set of pulse information on trace 2 to see if they team up with the information on trace 1. If it does, the program continues. If the computer does not recognize the initial data as being one of these four cases, it will stop the program and the user should check the quality of the input, and the **profile.info** file for information about where the computer thinks the leading and trailing edges are. This case usually only occurs when there is some initial jitter in the system.

4) The final part of the program counts the pulses, and based on the input data from **travset** generates position data. For single LED systems this is straight forward since there is no direction, every pulse is a new distance increment. For the two LED system, direction can be measured, and

in most applications, the system reverses itself, or shakes, and the computer has to recognize that motion. There are really four ways the LED traces can look when the motion reverses direction, and these depend on where the reversal occurs. For reference, figure 3 shows one type of 2 LED system.

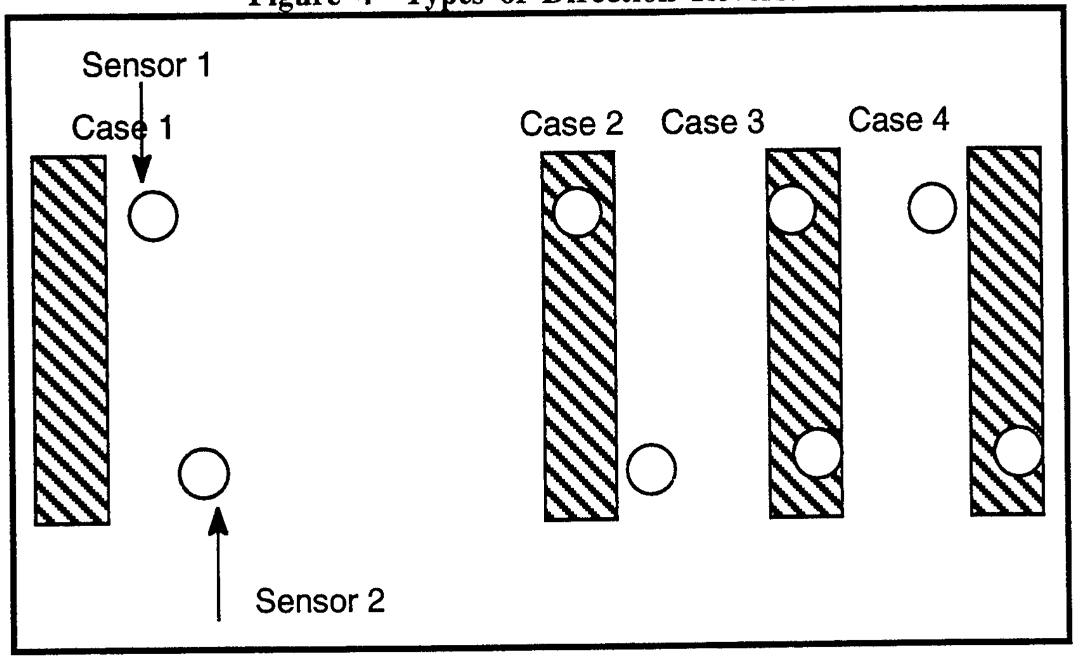
Figure 3 Schematic of Groove and Sensor Geometry



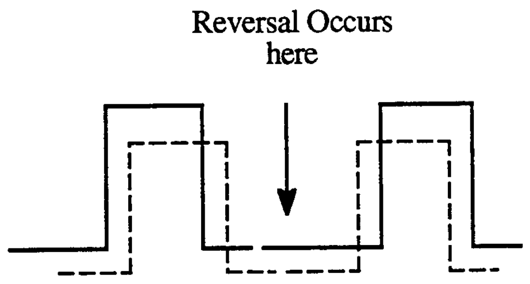
In this case we can assume that the sensors generate signals when the groove is completely underneath it, and that those signals are high. Thus for the motion indicated, the signal will look similar to that shown in figure 1, where the solid line would represent the trace of sensor 1 and the dashed line the trace of sensor 2.

If we assume that the initial motion is in the direction stated, we can represent the right side of the groove as a line in figure 4. Basically motion reversal can occur in four different states, represented by how far the hatched area in figure 4 has moved to the right, when the motion reverses.

Figure 4 Types of Direction Reversals

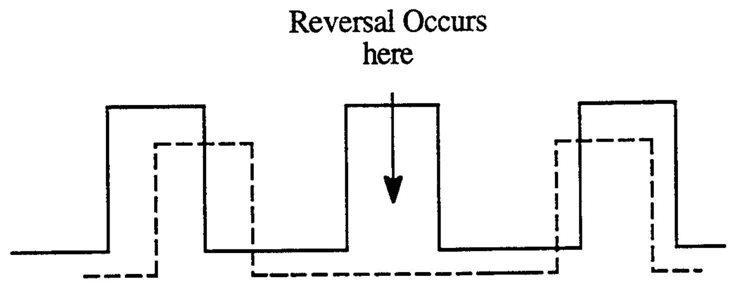


Case 1: In this case the pulse train looks like this:



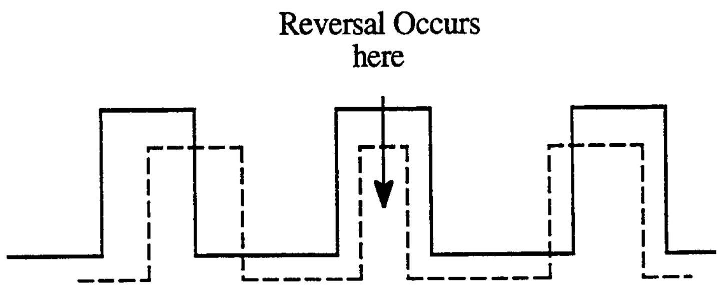
One can see that the pulse train that was leading now follows the other pulse train. This is relatively easy to spot, because one complete pulse sequence has occurred when the motion reverses itself.

Case 2: Here the groove has passed under the first sensor but has not triggered the second sensor. The pulse train in this case looks like:



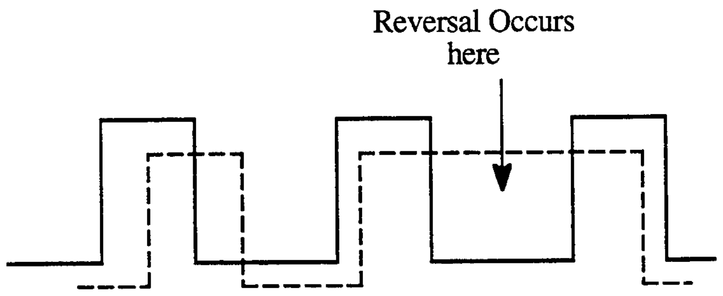
In this case the computer has to sense this type of motion from the fact that there was a leading edge and trailing edge on one pulse without something happening to the second pulse within that time frame. Because the sensors are connected in a quadrature system, this cannot physically happen unless the system changes direction.

Case 3: Here the groove has turned on both the first and second LEDs but has not turned off the first one when motion reverses.



The logic here is similar to that of case 2.

Case 4: The groove has turned on and off the first LED but only turned on the second LED.



This part of the program is set up to recognize these four situations and reverse the motion direction if necessary. If when running the program, one finds that the system is not reversing the way it should, check the raw LED traces to make sure one of these four cases is really existing.

Program Status and Future Work:

Traverse was modified to accept full quadrature signals (such as from an encoder) during the time when we were trying to measure the moment of inertia of a rotor. The old program contained similar logic, but could not account for the initial start up conditions of having one or both LED sensors high. In addition, one of the reversal conditions was not accounted for properly. As a result, the new program should be more versatile and robust.

Presently, most future work will actually involve changing **travset** and some of the checks in **traverse.** These, should not affect the control logic at all and are mostly aimed at easing the units conversion process and the motion controlling process of the traversing rings.

Problems Encountered Using Traverse:

If one is having a problem with the program one should check the following items:

1) Is there good, high quality LED signals coming into the system. If problems are occurring it is most likely due to the LEDs not being positioned correctly. Things to ask yourself:

a) Are these sensors looking at the same groove?

b) Are they positioned to provide a quadrature signal (i.e. one sensor is about 1/4 groove width from the other sensor)

c) Do the LEDs have the proper frequency response (i.e. can they see the pulses)

2) If all the hardware is operating and the input pulse trains look good, then

a) Decide if the problem is in the initial phase (i.e. determining which way the device is moving, or

b) In the device shaking, or reversing.

It is possible that given some type of LED configuration, a behavior other than that specified above may need to be accounted for. If that occurs, the code is easily modifiable to add extra cases; however, that should only need to be done as a last resort.

2.11 The RCDF and its Interactions with DAS Files

By: John Moselle and Charles Haldeman
Date: 5-6-93, Update: 1-18-95

Overview:

This technical note reviews the relationship between the RCDF (Run/Channel Descriptor File and the RDRs and CDRs (Run Descriptor Records and Channel Descriptor Records) stored in each data file. It describes how the data files are organized, how the RCDF interacts with the data files, some programs which interact with the RCDF and those which can be used to modify the RCDF. Attached to this document are the items contained in version 2 of the DAS RDRs and CDRs.

Data File Structure:

DAS data files are important entities within the context of an ATARR experimental run. DAS data files are always direct access, binary (unformatted) files, with fixed length records (1024 bytes). Presently there are three ways which DAS files are organized. For all *.0 files[1], the data is stored as I*2 formats (16 bit, or 2 bytes). Thus there are 512 samples stored in each record. The number of total records is defined by the total number of data points. For all *.1 to *.n files[2], the values are stored as Real*4 (or 32 bits), allowing only 256 samples per record. The third format supports separate x-axis data. The previous two formats basically have the data plotted either as a function of sample number or time. The actual value is based on the following:
a) the sample rate (stored in the CDR)
b) the sample number
Thus, one need not store all the separate sample numbers or times. One can see that this method will work whether one is using data sampled at a fixed time interval (asynchronous data) or a fixed spatial interval (synchronous data).

For the X-axis format, there exist twice as many records as in the other two formats. The first set of records represents the dependent variable (or Y-axis). After the appropriate number of records have been used (defined by the total number of samples), a new set of records is started in the file continuing the independent data (or X-axis).

[1] *.0 files are raw data files, see main documentation for a discussion of the different data types

[2] *.1 are data converted to engineering units, *.2 to *.n are data files which have been post-processed past basic engineering units

All data files have as their first two records, the Run Descriptor Record (RDR) and Channel Descriptor Record (CDR) which contain information needed by the DAS processes to perform their tasks[1]. However, every process accesses the RCDR's at different times. Some times these operations only involve reading information from the RCDRs. Other times, the processes in question will store information in the RCDRs. It is important to realize that some information in the RCDRs can only be determined after a test. Some examples are: the initial gas temperature and pressure in the supply tank and test section, the test gas composition, and the post-test calibrations. Thus the RCDR can be modified at different times, even after data is acquired. Thus a system is needed to keep track of these modifications.

The RCDF:

The system devised is the RCDF (or Run/Channel Descriptor File). This file is initially created by the Test Setup Process (**tsp**) and contains one RDR, and a CDR for every channel used in the run. **tsp** stores all the facility turbine parameters into the RCDF. The Fill and Calibration Process (**fcp**) marks the partial pressure and temperature conditions during the filling of the supply tank and stores these values in the RCDF. Throughout the period where **tsp** and **fcp** are being used to define a run, and prior to the execution of the Data Acquisition Process (**dap**), the user is unrestricted in his choice of Run Channel Descriptor File (RCDF) names. Once data is acquired in a DAS context (which can be done in either **fcp** or **dap**) the RCDF is used to create the data files by combining with the data for every channel, a copy of the RDR and the appropriate copy of the CDR. Subsequently, the RCDF is used as the basis for completing the RDR and CDR of each recorded channel as the data is processed.

There are several computer processes that read the RCDRs and need to modify them. The number of processes will surely increase in the future, but they will probably all behave in a similar manner to the ones listed below.

ccp (FORTRAN code)

This process is used to input calibration constants from outside the DAS environment and place them into one of the three possible calibration arrays: laboratory (C_Lab), pre-test (C_pre), and post test (C_post).

facility (FORTRAN code) or **mm** (X-11 interface)

This code is used to run the main math model which is used to predict the performance of the ATARR facility. It can be run as a stand alone code (**facility**) or can be used with an X-11 interface (**mm**) to create the configuration files needed to run the main program. This program operates in two modes. In the first mode it operates purely as a predictive tool for predicting

[1] See attachment for a listing of the variables contained in the RCDR.

performance of ATARR. In this mode the program operates completely outside of the DAS context (although it does generate DAS readable files) and does not need any information stored in the RCDRs. The second mode of operation is run after the supply tank has been filled. In this case it takes the partial pressure and temperature data stored in the RCDRs and uses it to calculate the test gas properties. For this mode of operation, the model reads data from the RCDRs and writes data to the RCDRs.

selcon (FORTRAN)

This process reads the calibrations constants and allows the user to select which set of constants is placed into C_used. These are the constants used by the data reduction process to create engineering units

standards (FORTRAN)

This process is used to examine the initial conditions of the experiment and place average values for the initial conditions into the RCDRs.

Clearly, once data has been taken, there exists a potentially large number of data files, all of which have RCDR's which must be modified. Instead of modifying each one individually, the RCDF is modified, and those changes are broadcast to the existing data files. In this way, one can always tell the existing status of all channels in a run just by examining the RCDF. Adopting a uniform name for the RCDF of the form *run*.rcdf* (* = run number) makes the programming much more streamlined. Further, this file must reside in the appropriately named turbine run subdirectory (*run**, where * is the run number) together with existing raw and primary data files. To ensure proper results, only one RCDF named *run*.rcdf* can exist in the current working directory (CWD) when any of the DAS processes are invoked.

Each process which updates the RDR portion of an RCDF results in the automatic broadcast of changes to existing raw data files and an entry in *run*.rcdf*. If *run*.rcdf* is not found in the CWD, it is created. Modifications to the CDR portion of an RCDF are presently restricted to the definition or selection of calibration constants which are also changed in the corresponding raw data file.

Viewing and Interacting with the RCDF:

The values in the RCDF can be viewed directly by using either **tsp**, **viewdas**, or **listdas**. **viewdas** allows the user to change values in the RCDF (as well as view the current RCDF) and should only be used in this mode for diagnostic purposes. Listdas can print the channels and can be used for review of all the channel setups before a run. In addition, changes which are made to the RCDF are also listed in the data run log file. **tsp** is the main program used to set-up the run and thus the RCDF, but not all the parameters which are in the RCDF are visible in this program.

There are also two programs which can be used to modify the RDR and CDR of a directory if need be. Caution: these should only be used in an emergency, because once used you destroy the continuity of the data system. These are:

modcdr.F - This program allows a direct change of the CDRs in the *.0 files in a directory. Once used, any *.1 files that were in existence will be out of date.

modrdr.F - This program allows a direct change of the RDRs in all the *.0 files within a directory. Same precautions as listed above.

Since the creation of a fully defined RCDF is so involved and the content so vital, it rivals raw data files in importance. The RCDF, however, unlike raw data files, can be reconstructed. This is accomplished by collecting the CDRs of the raw data files or by re-running certain peripheral processes. Although a means for collecting CDRs does not exist, it represents a routine programming task to a reasonably skilled "C" or FORTRAN programmer.

2.12 Channel Lists and the Keithley Standard

C. Haldeman
2-3-95

This note discusses very briefly the channel list files and their use in the DAS. Part of this discussion will focus on the Keithley multimeter and it's use in the A/D system.

Channel Lists

Channel lists are used in two separate ways in the DAS system. One use for these files is as a list of all the channels that are to be calibrated or displayed in either **fcp** or **monitor**. When the files are used in this manner, only the channels listed in the file will be activated (no matter what the RCDF says). The file is constructed by listing one file on each line such as:

PU1T
PU2T
PU3T
etc.

Note that there do not need to be any extensions, and that the channels names need to agree exactly with the ones listed in the RCDF (do not mix cases). Both **fcp** and **monitor** use the exact same format. These lists can be used for different tasks in the testing process. As an example, at Calspan we would probably have a list of every instrument which is used by **fcp** to examine the quality of the signals before a test. We will generally also have a list of just the pressure transducers in the test section. This list is used with **monitor** for calibration. We might have a third list for transducers in the driven tube which are used for calibrations of the shock tunnel. Once these lists have been created, they are used for the entire test matrix.

A second use for these files is as a list for the interactive batch processing in the data reduction. Most of these programs allow the user to generate the list "on the fly" by selecting channels from the RCDF. But they can also accept input from a file. In this case, the file looks exactly the same as listed above, and the user is allowed to select the appropriate extension to the file.

As an example, if one wanted to perform a FFT on a whole collection of *.1 files. The user could input the file shown above, and then select ".1" as an input extension. The program will then append the ".1" to all the channel names in the file list. This is very helpful since after many processes one could have several different extensions. This allows one base file to be used for many different tasks.

The Keithley Standard

Both **fcp** and **monitor** can perform calibrations. To perform a calibration, there needs to be a standard, something that the other channels are calibrated against. This becomes a particular problem if one is calibrating the A/D system, since one does not want the standard to be part of the system you are calibrating. To alleviate this problem, we set-up a system where the standard is a separate instrument, which is connected to the computer via a GPIB interface. When the data acquisition is being triggered, it is done via a GPIB bus trigger command.

Presently two different devices are supported (although many could be added). These are the Keithley 195A and the 2001 machines. The Keithley devices have the benefit of employing fundamentally different acquisition technology and calibration of these instruments is easy, well documented, and well supported.

This programs know to use this instrument if the following line appears at the top of the channel list:

KEITHLEY**X** **aaaa** **bbbb**

Where **X** can be either "T" or "P". Which stands for a temperature calibration or a pressure calibration. The **aaaa** and **bbbb** are the calibration constants which are used in the conversion of the voltage to an engineering unit. The actual conversion is a little different for every device type. For instance, the Keithley 195A could not display temperature directly (at least not accurately), so the conversion to temperature was done in a separate routine. However, for the 2001, the unit can display temperature very accurately, and thus no conversion constants are needed. For specific information about how the conversions are done in each case, the user is referred to **monitor.c** (the same code is duplicated in **fcp.c**).

2.13 Calibration Constants and Their Path Through the DAS

By: Charles Haldeman
Date: 1-18-95

Overview:

When the facility at WPAFB moved from being T^{3}R to ATARR, in addition to changing the name, the overall data accuracy requirements went up dramatically. And as anyone who has ever worked with complicated instruments knows, most of the "art" of data reduction is in the selection of the proper calibration constants and characterization of instrument quality. This became one of the driving forces behind the redesign of the DAS system. In fact, one way of looking at the entire DAS system (one of many), is to separate the system into three distinct pieces: the set-up of the test, the reduction from recorded counts to primary units, and the reduction from primary units to more useful scientific information. This note will focus primarily on the second part of this process. In fact one could say that the entire process of reducing the data from raw counts to primary units is just a matter of tracing calibration constants.

Background Information:

Calibrations constants can be entered, generated, stored and used all within the DAS context. They can also be created outside of the DAS system and then entered into the DAS context[1]. Calibration constants are traced by serial number and not channel labels. This occurs because the channel labels refer to a position, while the serial number refers to a specific sensor. During a testing matrix, the same sensor may be moved, thus acquiring a new label.

All calibration constants that enter the DAS are stored in three places: the RCDF and the channel RDRs and CDRs, and the ASCII calibration files. The former are clearly needed for data reduction. The latter is just a methodology for allowing the viewer to trace past calibrations. Every time a program which manipulates the calibration constants is used, an entry is sent to one of a series of files and is appended to the end of the file. If a file does not exist, then it is created. The files are all of the same type of name **xxx.pre, xxx.post, xxx.lab, xxx.HT** or **xxx.used** where **xxx** refers to the sensor serial number. These files contain the sensors label, run number, date and calibration constants. Basically the programs **ccp** and **fcp** write to the first four types of files, while **selcon** writes to the last.

[1] The DAS context refers to the type of file structure used to contain data. As one will recall, the DAS structure is to have two 1024 byte header records followed by 1024 byte data records, as opposed to a standard ASCII file.

In the beginning of this note, the entire DAS system was separated into three areas with the major focus of this note being on the second area. For completeness a little needs to be said about the other two sections. Setting up the test is normally done in **tsp**. It is extremely helpful to know how one will process the data before one takes it. This knowledge directly impacts how the test is set up in **tsp** (the reader is referred to the program note on **tsp**, and to the technical note on the recording mode). Once data is converted to primary units (temperature, pressure, strain, etc.) how one combines these to get useful knowledge is purely a scientific question, and not really one involving the A/D system except for questions involving frequency response. Thus it seems logical to say that almost all of the effort in the DAS system revolves around the reduction to primary units.

This part of the process can be dissected one step further into two subtasks. The first is the conversion of counts recorded by the digitizer to volts referenced to the input. This process is purely a function of the A/D system (and not the instruments). While one needs to verify the operation of the A/D and should do so on a regular basis, once done this is an automatic process, completely transparent to the user. The second subtask is the difficult one, converting voltages into engineering units. This is where questions about uncertainty in the measurements arise, which type of constants should be used, etc.

Calibration Types:

Before proceeding, it is worthwhile to establish some basic definitions. There are fundamentally four different calibration "spaces" in the A/D which can contain up to seven constants. There is also an algorithm number which determines how those constants are used[1]. These spaces are called: C_pre, C_post, C_lab, and thin-film. These are generally used to hold the calibration constants for a sensor generated before the test, after the test, in a laboratory setting, and for the thin-film sensors. Readers may remember that part of our conversion has been to place the heat transfer calibrations directly into C_lab. However, in our present software version, the thin-film constants are placed both in C_lab and thin-film.

While those are the general types of calibrations associated with each area, any set of three calibrations can be stored in the RCDF. These calibration constants do nothing in these areas. Only after the user copies one of these to the final calibration constant area C_used, will data reduction occur.

[1] These can be used in any form where EU = F(Ai, V) where EU is the engineering unit, A_i is the calibration constants, and V is the voltage referenced to the input.

Entering Calibration Constants:

There are several ways calibration constants get "entered" into the DAS system. They can be generated in the DAS system (such as done in **fcp**). They can be entered by hand in the **tsp** (but only the lab calibrations). Or, they can be entered through **ccp**. When calibration constants are generated within a program, the program handles the proper placement of the calibrations based upon the user input. **tsp** only allows access to the laboratory calibration area of the RCDR and is better suited to viewing the lab calibration constant than inputting them. It should only be used when one or two channels need to be updated. **ccp** allows far more flexibility and it is much more specific about how the calibration constants are being stored.

Within **ccp** there are basically two ways of entering the data: either by hand or through a file. **ccp** also allow the user to set specific values to be the same for all transducers (which is used when data is recorded in relative mode). **ccp** also allows the user to set some other parameters which may be needed in the RCDF. This program can be thought of as a miniature **tsp** which deals specifically with calibration constants. This program allows the user to enter all of the parameters needed by the RCDF to reduce the data. This program can be run many times during the data reduction cycle. It can be run before a test to put in the laboratory calibrations, it can be run after the test to enter the post-test calibrations. It allows the user to store many different types of calibrations which can be reviewed by different programs.

ccp is only one half of the process though. With all of the information stored the engineer still has to make the conscious decision about which calibration constants will be used in the data reduction. This is done in **selcon**. This program is very straight forward. It asks the user to decide which type of calibration constant should be placed in the specific channels. Once done it copies the calibration constants into C_used and also makes an entry in the ASCII calibration files. At this point, the user is prepared to reduce data.

The Next Step:

Once calibration constants have been placed into C_used, the data needs to be converted to engineering units (*.1 files). This can be done using one of two programs: **drp** and **reduce. drp** will allow plotting as well as reduction, and will provide some basic checks when reduction takes place and will provide the user with some diagnostic messages if an error occurs. This program is very useful for a few channels, but becomes unwieldy when reducing an entire run. **reduce** will do all the reduction automatically. There is actually no input required from the user to reduce the data once the calibration constants are set, because the algorithm number and sensor type are stored in the RCDF. However, this program does not do any specific checking of data validity, and if there is no data in the calibration constants, or if the specific data reduction flags have not been set (which are done in **ccp** and **selcon**) then the resulting time trace will be all 0's.

Once data is reduced to engineering units, a wide range of programs exist to process the data. The benefit of this system, is that for every data file generated, a history of which calibration constants were used is also stored in the file, providing instant tractability.

2.14 The DAS Recording Modes: Absolute and Relative

By: Charles Haldeman
Date: 1-26-95

Overview:

This note reviews the differences between the two types of data recording modes in the DAS system: Absolute and Relative. Presently there is also a third option available within **tsp** (Calibrated), but that option is not fully operational and should not be used. Since the choice of recording style has major implications on data interpretation, it is important that the user understand these differences to best utilize the system.

In general, the differences between these two systems parallel the differences between an AC coupled signal and a DC coupled signal. The differences are manifested in the zeroing of the A/D system and the reduction of the data to primary units (done within **direct.F** or other similar programs).

Zeroing the A/D System:

When the A/D is zeroed (which is done at the beginning of every data acquisition program: **fcp, monitor, dap**), each channel is zeroed in a manner according to how the recording mode is set. For channels which are to be recorded in absolute mode, the amplifier inputs are internally shorted, and the A/D system attempts to set the observed amplifier offset to the value specified in the Channel Descriptor Record (CDR). This is equivalent to setting the zero voltage input of the amplifier to a specified voltage level on the A/D system. While absolute accuracies will vary with the total gain, this value should be correct to within about ±2-5 bits. In relative mode, the digitizers look at the present output of the amplifiers and instruct the amplifiers to move the offset to the value listed in the CDR. This is the same as having the present input of the amplifiers moved to the offset specified in the CDR.

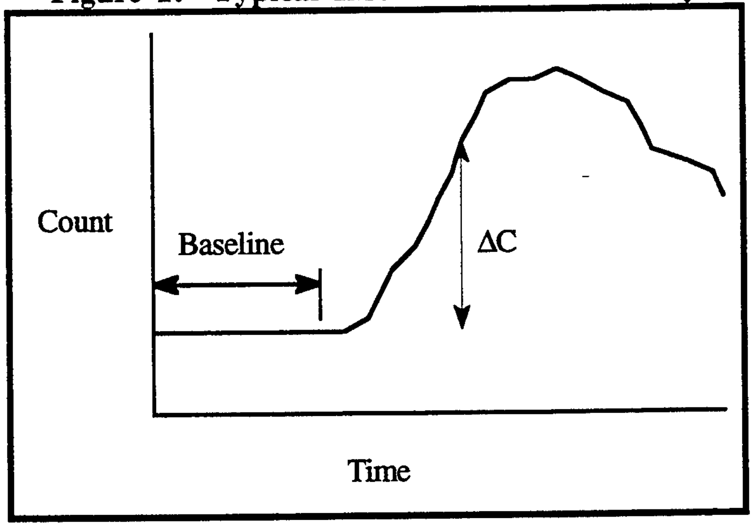
As an example, if there is a 1 VDC signal coming into the amplifiers and the offset is desired to be at -90% (-4.5 VDC), then an absolute system would show a signal entering the digitizers of -3.5 VDC (-4.5 + 1 VDC input). A relative system would have the input to the digitizers reading -4.5 VDC (the 1 VDC signal has been offset by -5.5 VDC to make the signal entering the digitizers -4.5 VDC). One should note that this process takes three passes to accomplish. In each pass the amplifiers set a specific offset, and the digitizers record the position of the signal. Based upon where the signal is supposed to be (the offset value in the CDR) and where it actually is, the amplifiers change their internal offset.

Absolute mode is generally used when the drift in the electronics is not a major issue. It provides the most easily understood and straightforward implementation. Relative mode is useful for systems where the offset values can change over time, but the instrument gains do not. In this way, the change in the instrument offsets will not affect the data. However, for channels to zero properly in relative mode requires the entering signal to be relatively steady. A varying AC signal will cause problems for the system when it calculates the voltage to determine the proper offset.

Reducing to Primary Units:

Once the data is recorded (in counts), there is theoretically a direct conversion to voltage using only the gain and offset of the system. To convert to primary units (temperature or pressure) one needs the calibration constants. Practically, the conversion from raw counts to primary units (either voltage temperature or pressure) is done in one step. Figure 1 shows an example history of a typical sensor and describes how the data reduction scheme differs for the two systems.

Figure 1: Typical Instrument Time History



In an absolutely recorded system, the conversion from counts to volts is direct and is of the form:

$$Volts = (Counts - Offset)C_1 \tag{1}$$

where

$$C_1 = \frac{Full\ Scale\ Voltage\ Range}{(Bit\ Resolution)\ Gain} \Leftrightarrow \frac{10}{(2^{12})Gain} \tag{2}$$

Offset is the recorded offset of the A/D which is the actual value stored in the CDR parameter *recset* (which is very close to what was suggested in the property *offset*). The only difference between the value stored in *recset* and *offset* is that the former is the actual offset of the A/D versus the latter which is the one asked for by the user[1]. For relative recorded data, equations 1 and 2 are still valid, but the offset is taken as the average value of the trace during the baseline. Thus in this system, the user must define the baseline starting and ending times (done in **ccp**).

One can see that for the absolute system there is always a direct correspondence of counts to voltage which is independent of where the A/D was zeroed. However in the relative system, the voltage signal will always go from 0 (at the average of the baseline) to some other voltage. In this system, one is only interested in the variation from the baseline (ΔC).

From voltage to engineering units (pressure or temperature) requires some additional calibrations constants. These can be obtained from many sources, but the user must select the ones desired and place them in *c_used* (done in **selcon**). In an absolute system, a polynomial conversion is of the form:

$$EU = a_0 + a_1 V + a_2 V^2 \ldots \tag{3}$$

where a_i represents the calibration constants. In relative mode, equation 3 is still valid for the linear case only, and the constant a_0 must represent the engineering unit equivalent of the trace at the baseline! This is clear since you are only working on a ΔV system that you must have some reference point. Equation 3 is still valid in its entirety for relative systems, if when the calibration is being done, the baseline is always set to 0 Volts. Other types of calibrations (other than polynomial) can also be supported.

For all practical purposes, there is no separate conversion from counts to volts and then volts to engineering units. This is all done at once depending upon what the user desires. After the reduction to primary units is made, there is no real concern over what mode the data was acquired in.

Summary:

The choice of recording mode really is determined by the experiment. Absolute mode gives one the best chance of tracking errors because it makes them more visible in the form of drift and it makes the use of non-linear sensors much easier. Relative mode reduces ones dependency on well-operating equipment, but does restrict the types of sensors which can be used. In addition, it is critical that one know the engineering unit during the baseline phase of the data. If one does not know this during the actual testing, but derives that from when the A/D is zeroed, it is important that the instrument remain in the same state from initialization to testing. In addition, using relative mode requires the additional input of the baseline time interval and the initial engineering unit.

[1] All of these equations are taken from **direct.F** and the user is referred to that program for more in-depth coverage of all the conversions.

2.15 Data Reduction to Primary Units

By: Charles Haldeman
Date: 2-19-95

Overview:

As mentioned throughout the DAS documentation, before one even starts to set-up the RCDF for a test, one should have a basic idea of how the data will need to be reduced. Sometimes experiments will use standard equipment and this task will seem relatively mundane. Other times more exotic equipment might be used, or a standard type of transducer used in a more novel way. Thus to insure that the reduced data is correct, there is no substitute for reviewing the actual routines that convert the data from raw counts to primary engineering units. Since one of the benefits of the DAS is its extreme flexibility and ease of modification, there is no telling how the reduction programs might have been changed over time.

This note discusses the part of data reduction which occurs between raw counts and basic engineering units. As such, it focuses on the programs **direct** and **reduce**. Because these programs are so easily modified, this note is designed to help the reader understand the code, versus describing exactly what the code does (since it will probably change).

Code Structure:

Both **reduce** and **direct** share the same basic parts which do the conversions from base A/D counts to engineering units. The only difference in the codes is that there is a little more flexibility in calling **direct** than in **reduce**. **direct** is called by **drp** and the user can select some intermediate reduction stages. For instance, a pressure transducer can be reduced to voltages and then to pressure, while in **reduce** there is no chance to reduce to voltage, the reduction process goes directly to the final engineering units.

This occurs because all the information needed to reduce a channel to its engineering units is described by the sensor type and its recording type. If one has multiple kinds of temperature sensors, one which has a polynomial conversion (such as an RTD or a thermocouple) and another which uses an exponential process (such as a thermistor), one only needs to define a new sensor type. As long as the sensor can be described with seven calibration constants one can use the existing DAS structure to reduce the data.

Thus in **drp** the choice of "type of reduction" from the algorithm menu is redundant. The **direct** code contains some ability to allow conversion to voltages if need be, and will inform the user if an error occurs when reducing the data. None of these "benefits" exist in **reduce**. Other than that the code is basically identical.

There are four items (five for **direct**) which are key to the data reduction process.

algnum (direct only) - This is the algorithm number from the **drp** menu list. Presently there are seven listed.

1: Voltage
2: Turbine Speed
3: Pressure
4: Temperature
5: Heat Flux
6: Stanton number
7: Position

The last two have been "disconnected" in **direct** and **reduce** (if they are called, the program will return an error). This is due to the fact that a Stanton number calculation requires many explicit reference properties. This type of conversion is best done in one of the other data reduction programs where the various options can be more easily handled. The position algorithms have been moved to the program **traverse** which can handle the many different options needed.

ctype - This is the current data type of the file being operated on (***.0, ***.2, ***.3, etc.). This is just a check to make sure no one re-reduces data (i.e. tries to convert from pressure to pressure for instance).

deltav - This is the flag that describes whether the data was acquired in relative (1) or absolute (0) mode.

type - This is the number which describes the sensor type (and thus how the calibration constants are used). Presently, almost all of the conversions are polynomial based, although that does not always need to be the case.

1: Kulite
2: RTD
3: Thin film heat flux
4: Baratron
5: Thermocouple

standard - This is a flag which is set only after the baseline time interval is set (in **ccp**) if data is acquired in relative mode. If this flag is not set, **direct** will return an error, and **reduce** will return all zeros.

Basically the code is just a series of branch statements. It first checks to make sure that one is processing the proper data, and that one has all the proper information. Then based upon whether it is absolute or relative, it will convert the data to primary units. Thus the code is very simple to modify and new algorithms should be easy to incorporate.

2.16 Software Flowchart Notes

C. Haldeman
9-20-95

The following section contains "flowchart outlines" of some of the main programs. This was done primarily for the programs involved in data acquisition, although it could be expanded in the future. The goal of these outlines was not to flowchart every aspect of the program, since that would be a gigantic effort, and would require a great deal of updating over time as small changes are made in the programs. Rather it was designed to show where the major connections are between pieces of code, where the code is located, and provide a quick synopsis of what each piece of the code does. This becomes more important if one is considering changing the system at all.

To aid in this process the following nomenclature is used:

1) Program flow proceeds from left to right and top to bottom with the main program always being in the top left side. The program always has its main source code and location referenced.

2) The $\Rightarrow$ means that the following subprogram is called. This program will be in bold and may have the following information on the same line with it [xxxxxx] **{yyyy,** ***zzzzz*** **,[O]}**. The Values in the [] are major arguments that are passed from the calling routine to the subprogram. If these are not included then there are no major arguments. The { } contain the file which the subprogram is in. Usually the file is the same name as the subprogram **but not always**. The italics indicate where in the original file system the subprogram is. If there are no names in italics the assumption is that the program resided in the same directory as the main program. If there is a [O] in this are it means that this subprogram has an outline also. An example from **tsp** may help. $\Rightarrow$ **getchaninfo {getchaninfo.c,** ***src/util*** **,[O]}** implies that tsp calls getchaninfo which is in a file called getchaninfo.c residing in an area called src/util and that this

3) The $\Leftrightarrow$ implies that the item is in a loop

4) For applications using the X-11 interface, the XXXX $\rightarrow$ **YYYY** describes a button calling as subroutine within the program. These in turn usually call other sub-programs. The XXXX is the button displayed on the screen while the **YYYY** is the subroutine in the program (such as Go to Channel $\rightarrow$ **cgoto**).

5) After the outline, if it is appropriate, a list of calling programs of the subprogram may also be given. This is usually done for the low level routines which one may be interested in modifying. This provides an idea of how many programs one may affect if one makes a change.

Listing of Program Outlines:

tsp

monitor

dap

fixit.c

getchaninfo

zerochannel

2.16.1 tsp Flowchart

C. Haldeman

9-20-95

TSP is the program used to set up the "RCDF" file for a run. The program name stands for "Test Setup Process" and is invoked by typing "tsp". The program is stored in */usr/atarr/src/tsp.*

tsp {tsp.c, ***/usr/atarr/src/tsp*** **}**

⇒ **getchaninfo {getchaninfo.c,** ***src/util*** **, [O]}**

(This checks the hardware resources file and provides the default values for the RCDF file)

⇒ **fillwindow {tsp.c}**

(this takes information from channel i and puts it in the display window, CDR)

⇒ **fillrun {tsp.c}**

(This takes the default RDR and places it in the Run Descriptor Window)

<u>Window Buttons</u>

Quit → **quit {commands.c}**

(Exits the program)

Load RCDF→ **load {commands.c}**

⇒ **loadrcdf {rcdf.c,** ***src/util*}**

⇒ **fillrun {tsp.c}**

⇒ **fillwindow{tsp.c}**

(Opens the selected RCDF and puts the current CDR in the display and the RDR in the Run display)

Save RCDF→ **save {commands.c}**

⇒ **updater {tsp.c}**

(Updates the RDR with information from display)

⇒ **update {tsp.c}**

(This puts information into the CDR from the display and does all the special conversions, such as for the Datalabs, to make the CDR uniform)

(This saves the RCDF to the user selected file)

Edit Run Parameters → **sw2des {tsp.c}**

⇒ **update {tsp.c}**

(Switches the display window from the RDR display to the CDR display and vice-versa)

Update Calibrations → **updallsensor {tsp.c}**

(This gets the latest reading from the lab coefficient file of all the sensor calibrations)

⇒ **update {tsp.c}**

⇒ **fillwindow{tsp.c}**

Go to Channel → **cgoto [current_channel]{tsp.c}**

⇒ **update {tsp.c}**

(First it updates the current channel)

⇒ **fillwindow{tsp.c}**

(Then it goes to the new channel and fills the display with that information)

Copy to Channel → **ccopy [current_channel]{tsp.c}**

⇒ **update {tsp.c}**

(First it updates the current channel, then it sets the current channel number to the new channel number, and recalls update)

⇒ **update {tsp.c}**

Next Channel → **cincrement [current_channel]{tsp.c}**

(increments the current channel number)

⇒ **cgoto {tsp.c}**

Copy Thru → **cthru[current_channel]{tsp.c}**

⇔ **update {tsp.c}**

(Loops from current channel to the final channel number, calling update)

⇒ **fillwindow{tsp.c}**

Sensor Serial number → **setsensor{tsp.c}**

⇒ **updsensor{tsp.c}**

(Looks at SENSORFILE, and updates sensor number laboratory calibration constants)

2.16.2 monitor Flowchart

C. Haldeman

9-20-95

This program acquires data over a long time period of time. It sets up the A/D system once and then goes through a process of arming, triggering, reading, and re-arming the system. This program also is capable of talking to a Keithley multimeter (either a K195A or a K2001) which can be used as a standard. The program has several different operation modes.

```
monitor {monitor.c, /usr/atarr/src/dap}
    ⇒ getchaninfo {getchaninfo.c, /usr/atarr/src/util , [O]}
    ⇒ getrcdf [infile] {monitor.c}
        ⇒ getchaninfo {getchaninfo.c, /usr/atarr/src/util , [O]}
        ⇒ If hwok || Debug
        (this is a hardware debugging flag which allows the software to be run without any
        hardware connected)
            retval = loadrcdf {rcdf.c}
                ⇒ getrdr {rcdf.c}
                (This reads in the RDR record and checks for an ATARR magic
                number and the DAS version number)
                ⇒ getcdr {rcdf.c}
                (This gets all the CDRs in the file and performs an fseek and an
                fpoke on them)
        ⇒ else
            retval = errhw {monitor.c}
                ⇒ checkhw {dataacq.c}
                (This actually checks the hardware resources file and make sure that
                the crates and amplifiers agree with the RCDF)
        (This entire set of routine is to check the validity of the RCDF with the hardware
        resource file)
    ⇔ initproc{monitor.c}
        If | Debug
        ⇒ open_crate {dataacq.c}
        ⇒ if channels are active call set_amp {dataacq.c}
        ⇒ zerochannel [0] {zerochannel.c}
        (This zeros the absolute channels)
        ⇒ zerochannel [1] {zerochannel.c}
```

(This zeros the relative channels)

⇒ **zerochannel [2] {zerochannel.c}**

(This zeros the calibrated channels)

(This loop awaits user input to either run again or to stop the zeroing process and continue)

⇒ **inittime {timer.c}**

⇒ **snooze_init {timer.c}**

⇔ **data acquisition loop**

⇒ **acqdata {monitor.c}**

(This acquires the data)

⇒ **storedata {monitor.c}**

(This stores the data in a file

⇒ **snooze {timer.c}**

(This makes the system wait for a specified period of time)

(This loop runs until the specified number of time intervals passes or until the program stops the program.)

2.16.3 dap Flowchart

C. Haldeman

9-25-95

This program is the main data acquisition program.

```
dap {dap.c, /usr/atarr/src/dap}
    ⇒ getchaninfo {getchaninfo.c, /usr/atarr/src/util , [O]}
    Load RCDF → FileSelect {fileselect.c, /usr/atarr/src/fileselect}
    (Loads the RCDF)
    Load Profiles → FileSelect {fileselect.c, /usr/atarr/src/fileselect}
    (Loads the profiles for the traversing ring, if required)
    Initialize → inithw {dap.c}
        ⇒ open_crate {dataacq.c}
        ⇒ if channels are active call set_amp {dataacq.c}
        ⇒ zerochannel [0] {zerochannel.c}
        (This zeros the absolute channels)
        ⇒ zerochannel [1] {zerochannel.c}
        (This zeros the relative channels)
    ARM DAS → armhw {dap.c} ⇒ armproc {dataacq.c}
        ⇒ clear_aflags {dataacq.c}
        ⇒ clear_cflags {dataacq.c}
        ⇔ Loops through all channels
            ⇒ set_channel {dataacq.c}
        ⇔ Loops through all channels
            ⇒ arm_channel {dataacq.c}
        If there are Profiles, loop through all Profiles
            ⇒ initcont {travcont.c}
    Record Data → recordproc {dap.c}
        ⇔ Loops through all channels
            ⇒ read_channel {dataacq.c}
    Trigger Das→ triggerproc {dap.c}
        ⇔ Loops through all channels
            ⇒ trigger_channel {dataacq.c}
    ABORT→ abortproc {dap.c}
```

2.16.4 fixit Flowchart

C. Haldeman

9-20-95

This routine basically checks the validity of the hardware resources by examining the hardware resources file and making sure that the computer can open all the required devices. The program accepts a full device command (i.e. device, slot number, function, address, and data) and will write the command directly to the device.

```
fixit {fixit.c, /usr/atarr/src/dap}
    ⇒ getchaninfo {getchaninfo.c, /usr/atarr/src/util, [O]}
    (Checks validity of HWRF)
    ⇒ open_crate [crate]{dataacq.c}
    (This open individual GPIB devices based upon the type in the crate structure)
    If function >= 16
        ⇒ camwrt16 {camac.c}
        (writes the data to a camac type crate)
    Else
        ⇒ camrd16 {camac.c}
        (reads data from a camac type crate)
```

2.16.5 getchaninfo Flowchart

C. Haldeman

9-20-95

This routine serves two main purposes. The first is that it provides the hardware consistency checks between what is stored in an RCDF and the actual state of the hardware resources file at the time it is called. This checks to make sure that no one has changed the hardware resources file from when the RCDF was created to when it will be run. The second purpose of this program is load default values into the RCDF for use by **tsp**.

getchaninfo {getchaninfo.c, *_/usr/atarr/src/util_*}

⇒ **readhwconfig {hwconfig.c}**

(This will return a 0 if the hardware resources file is valid. It reads in all the variables stored in hwconfig.h and any program with that in its include statement can access those)

⇔ **Loop through all channels**

⇒ **getamp {hwconfig.c}**

(this program returns the type of amplifier in the hardware resources file. This is a variable which is used to switch default settings in the RCDF.

getchaninfo. is called from the following programs:

/dap -dap.c

-fcp.c

-fixit.c

-hwcheck.c

-gronk.c (This program can be eliminated, it is an earlier version of hwcheck.c)

-monitor.c

/dsp -dap.c (This entire directory can probably be eliminated. It is probably an earlier version of all the dsp code. One should check to make sure there is not anything super useful in there first)

/tsp -tsp.c

readhwconfig is called from:

/dap -menu1402.c (This is a diagnostic program for the amplifiers)

-getchaninfo

2.16.6 zerochannel Flowchart

C. Haldeman

9-25-95

This program is called by the main data acquisition programs and is used to zero the channels and set all the appropriate switches in either the Datalab system or the DSP system.

```
zerochannel {zerochannel.c, /usr/atarr/src/dap}
    ⇒ clear_aflags {dataacq.c}
    ⇒ clear_cflags {dataacq.c}
    (This sets the armed and configured flags to false)
    ⇔ Loops through all channels
        ⇒ If !rezero
            ⇒ get_amp_offset {dataacq.c}
                ⇒ getchannel{hwconfig.c, /usr/atarr/src/util}
                ⇒ get1402offset {1402abh.c}
            (This routine basically obtains the proper offset value)
        ⇒ set_amp{dataacq.c}
            ⇒ getchannel{hwconfig.c, /usr/atarr/src/util}
            ⇒ getamp{hwconfig.c, /usr/atarr/src/util}
            Switch type =1402
                ⇒ set1402{1402abh.c}
                (Sets the 1402 module values using base camac commands)
            Switch type =2904
                ⇒ set2904{2904.c}
                (Sets the 2904 module values using base camac commands)
        (Entire routine is designed to figure out which amplifier is being addressed and call
        the correct hardware routine).
        ⇒ Switch Level
            ⇒ set_cal{dataacq.c}
                ⇒ getchannel{hwconfig.c, /usr/atarr/src/util}
                ⇒ set1402offs{1402abh.c}
                (Sets the 1402 module offset using base camac commands)
        (This routine just sets, based on the value of level, how the set_cal is called)
        ⇒ set_channel {dataacq.c}
            ⇒ getchannel{hwconfig.c, /usr/atarr/src/util}
            ⇒ getamp{hwconfig.c, /usr/atarr/src/util}
```

```
                    Switch type =4012
                            ⇒ set4012{1402abh.c}
                            (Sets the 4012 module values using base camac commands based
                            upon what type of amplifier is associated with it)
                    Switch type =6020,6023,6025
                            ⇒ dlset6010{datalab.c}
                            (Sets the datalab timer module values using base datalab crate
                            commands)
                            ⇒ dlsetchan{datalab.c}
                            (Sets the datalab amplifier/digitizer modules values using base
                            datalab crate commands)
⇔ Loops through all channels
            ⇒ arm_channel {dataacq.c}
                    ⇒ getchannel{hwconfig.c, /usr/atarr/src/util}
                    Switch type =4012
                            ⇒arm4012{1402abh.c}
                            (arms the 4012 module values using base camac commands based
                            upon what type of amplifier is associated with it)
                    Switch type =6020,6023,6025
                            ⇒ dlarm{datalab.c}
                            (Arms the datalab timer module values using base datalab crate
                            commands)
            (This routine arms all the channels and returns a flag array telling the computer that
            the channels are armed)
⇔ Loops through all channels
            ⇒ trigger_channel {dataacq.c}
                    ⇒ getchannel{hwconfig.c, /usr/atarr/src/util}
                    Switch type =4012
                            ⇒trigger4012{1402abh.c}
                            (triggers the 4012 module values using base camac commands)
                    Switch type =6020,6023,6025
                            ⇒ dl_trigger{datalab.c}
                            (triggers the datalab timer module values using base GPIB
                            commands)
            (This routine triggers all the channels and returns a flag array telling the computer
            that the channels are no longer armed)
```

```
⇔ Loops 3 times
    ⇔ Loops through all channels
        ⇒ get_amp_offset {dataacq.c}
        ⇒ read_channel{dataacq.c}
            ⇒ getchannel{hwconfig.c, /usr/atarr/src/util}
            ⇒ getamp{hwconfig.c, /usr/atarr/src/util}
            ⇒ getcrate{hwconfig.c, /usr/atarr/src/util}
            Switch type =4012
                ⇒ read4012{1402abh.c}
            (This routine does some special things if the 4012 is used with the
            spincoder)
            Switch type =6020,6023,6025
                ⇒ dl_trigger{datalab.c}
                (treads the datalab modules )
        (reads the data from all the digitizers)
        ⇒ set_amp_offset {dataacq.c}
    ⇒ clear_aflags {dataacq.c}
    ⇒ clear_cflags {dataacq.c}
    ⇔ Loops through all channels
        ⇒ set_channel {dataacq.c}
    ⇔ Loops through all channels
        ⇒ arm_channel {dataacq.c}
    ⇔ Loops through all channels
        ⇒ trigger_channel {dataacq.c}
    ⇔ Loops through all channels
        ⇒ read_channel {dataacq.c}
        ⇒ set_cal {dataacq.c}
```

Zerochannel is called from the following programs:

```
/acquire - acquire1.c
/dap    -dap.c
        -fcp.c
        monitor.c
/dsp
```

Section 3. Version I Manual

This manual contains a great deal of useful information about some of the FORTRAN subroutines that are used, the file naming convention, the channel naming convention and other items that are either glossed-over or not dealt with extensively in other sections. However, if information described in this manual conflicts with the other two sections of this report, those sections should take precedence.

This manual is included only as a reference to past work. As such it is not an integral part of this report. There are two figures which were deleted because they directly conflicted with the newer system design. However the authors of this report caution the reader against using the data contained in the version I manual without relating it to the previous two sections of this report to check for conflicts.

APPENDIX A

INTRODUCTION

TO THE

ATARR DATA ACQUISITION SYSTEM

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. Introduction to the ATARR Data Acquisition System (DAS)

The purpose of this document is to give users of the Data Acquisition System (DAS) the ability to understand, use and where necessary, modify the software to analyze experimental data recorded from instrumentation in the Advanced Turbine Aerothermal Research Rig (ATARR). Reference will be made to concepts and abbreviations introduced in the Critical Design Review document[1] in order to avoid unnecessary duplication. However, lengthy descriptions will be repeated wherever clarity may be enhanced. The DAS resides on a machine called "atarr". This term will be used to mean the computer wherever it appears.

The atarr operating system is SunOS 4.1.1 which is the Sun Microsystems implementation of UNIX. It represents an effective merger of BSD and System V UNIX. If the user is acquainted with UNIX, working with the DAS should present no problems. Those with little experience are advised to get access to a copy of *The UNIX Programming Environment*[2] or some equally respected reference to compensate.

In the course of implementation of the DAS design, the system overview has undergone considerable revision. It is presented in Figure 1 and should be fully understood before a new user hopes to effectively operate the DAS. Although the hardware resources (HRF) and sensor calibration (SCF) files do not appear, their importance in the overall scheme is unchanged.

In general, the user accesses the numerous programs and routines which comprise the DAS formal processes through the X11 window interface. Since there are so many routines available to the DAS, it is easy to conceive of problem areas where one or more of them would be helpful. Toward that end, several libraries of predominantly Fortran routines are provided along with examples of how to use several of the most useful entries.

Recorded voltages and time histories which are produced as a consequence of DAS processes are stored in direct access, binary files according to some very specific arrangement. The need to access this information in other than the DAS prescribed context has been provided for by the creation of the "VAR" file format, a concept which is introduced in this report. Again, the user is given a number of examples detailing the bi-directional DAS <=> VAR file conversion formalism.

As stated above, the primary goal here is to present the informed user with a practical tool for interacting with the DAS. Hopefully, there is enough content to enable him to easily incorporate changes to account for unanticipated requirements arising during the installation of the system. Any questions pertaining to the DAS may be confidently directed to the authors.

Figure Deleted to Avoid Confusion with Changes in System Design

Figure 1. ATARR DAS System Overview

2. Using the DAS

Figure 1 summarizes the major tasks which the DAS has to perform. It shows the relationships between the processes needed to perform a turbine experiment in the atarr. This chapter will delineate all the elements which need to be in place in order to accomplish this end. All descriptions which reference the screen, keyboard or mouse refer to the atarr console or one of its two X-terminals.

There will be a number of people who will be authorized to access the system. Each will logon to atarr with his username and password. The username is usually the surname but a limit of eight characters may require some users to choose a reasonable alternative. Passwords may be changed, as often as desired, using the UNIX "passwd" command.

The system manager will provide each new user with default set of resource files to ensure a successful voyage into the world of UNIX and X11. Experienced users will have already established preferences and may define their resources accordingly.

Once a user has successfully connected with atarr, the full range of system capabilities are available to him. It is not necessary to be a UNIX guru since the mouse and keyboard provide very effective means for user interaction.

2.1 Selecting a DAS Process for Execution

There are a few simple steps required to execute each DAS process. First, it is necessary to define what is meant by the terms "root background" and "pointer". The former refers to the area of the screen which borders the window created during logon. The latter is the symbol associated with the mouse wherever it is located on the screen.

To select one of the DAS processes move the pointer to the root background. An "X" should be displayed on the screen. If not, move the mouse until this requirement is satisfied. Now, if the right mouse button is pressed and held, "dragging" the mouse slowly downward will highlight each root menu item in sequence. Stop moving the mouse when the required DAS process is highlighted. Releasing the right button will start program execution by placing a "ghosted" window on the screen. The mouse is used to position the window which is "dropped" by pressing the left button. Once a process has been successfully started the user controls the pace and direction the session will take mostly with the mouse and, where necessary, with the keyboard.

2.2 Directory Structure

All activities in the DAS center about information contained in files. The file browser, designed specifically for the DAS, provides the ability to locate and access any active file on the system. Figure 2 shows the straightforward approach used in the DAS to determine where its various files reside.

Operational Portion:

```
HOME -- Home directory of the facility (/usr/data/atarr)
          HRF --     Hardware resources file
          SCF --     Sensor calibration file
          TURBINE -- Turbine subdirectory containing current RCDF
                     RUN_1
                     RUN_2       Run subdirectories containing :
                       .         A. Run/Channel Descriptor File (RCDF)
                       .         B. Raw data files (RDF)
                       .         C. Physical units data files (PUDF)
                       .         D. Derived channel files (DCF)
                     RUN_n
```

Software Portion:

```
TREE --   atarr software tree (/usr/atarr)
          /lib --      DAS libraries
          /include --  DAS header (include) files
          /src --      Source directory tree
                       /dap          Data acquisition software
                       /drp          Data reduction software
                       /tsp          Test setup software
```

Figure 2. DAS Directory Structure

Some of the file resources needed by DAS processes require no user intervention to be brought online. The hardware resources file (HRF) and sensor calibration file (SCF) are two examples of facility resources dealt with automatically. In most other instances the user is required to know the exact name of each file to be accessed. Where channel time histories are concerned, the gauge label is the filename and the extension is a single digit indicating its evolution through the data reduction process.

Whenever a DAS process is invoked, the user will be able to access and create files owned by atarr. However, only users who can login as atarr may remove such files from the system. This policy is instituted to prevent one user from removing files painstakingly created by another while executing the data reduction process. The voltage counts recorded by each channel during a turbine experiment are singularly important. Consequently, once the information has been downloaded to raw data files at the completion of the data acquisition process, the files are copied to 9-track tape.

3. Data Files

All files used in conjunction with the DAS are either internal or external. Internal files are those created by and for the DAS as an integral part of its formal processes. External files communicate data to and from the DAS for peripheral applications.

All internal binary data files contain a copy of the run descriptor record (RDR) and the pertinent channel descriptor record (CDR) from the run/channel descriptor file (RCDF) corresponding to the turbine experiment being considered. The elements in the RDR and CDR provide the means for the DAS to carry out prescribed computational steps and are accessible to Fortran routines via structures defined in file ´dstruc.inc´. Appendix A contains the type, length, name and description of the individual RDR structure entries. The CDR structure is similarly described in Appendix B.

3.1 Internal Files

Internal files serve the variety of tasks performed by the DAS and, therefore, exist in a variety of forms. The complete list of internal files follows:

File Designation	Form	Content
Hardware Resources (HRF)	Ascii	Current configuration of acquisition hardware
Sensor Calibration (SCF)	Ascii	Calibrations for each atarr sensor
Sensor History (SHF)	Ascii	Chronology of sensor calibration changes
Run/Channel Descriptor (RCDF)	binary	Acquisition and reduction info (RDR+nCDRs)
Raw Data (RDF)	binary	RDR, CDR and recorded A/D samples
Physical Units Data (PUDF)	binary	Produced from calibration algorithm on RDF
Derived Channel (DCF)	binary	Result of further processing on PUDF
Variable Origins (VOF)	binary	From PUDF and/or DCF or external source

The binary files have been given new designations which are reflected in the CTYPE parameter of the CDR. Briefly, the values of CTYPE are: 0 for RDF; 1 for PUDF; 2 for DCF; and, 3 for VOF. Zero level files will still be referred to as raw data but levels one, two and three will now be designated primary, secondary and tertiary files in the DAS documentation. The single digit in CTYPE is used as the extension to the filename for the associated channel time history.

A primary file is one obtained directly from raw voltages. The timer channel and the time histories for temperature and pressure are the only primary files. The timer channel has double precision values to provide required computational accuracies. Secondary files are derived directly from primary files and include heat rate, average temperature and pressure, and rotor speed time histories. All other files are considered to be tertiary and are either derived from primary and/or secondary data or converted to DAS internal format from an external source.

By convention, all primary, secondary and tertiary files will have the RDR and one CDR as the first two records followed by as many records as necessary to contain the data time histories. Any data from external files required by DAS processes will be converted to internal structures which conform to this conventional pattern.

It is expected that Fortran will be the primary high level language used whenever it is necessary to access information on these files in non-DAS applications. In such cases, the file should be treated as unformatted, direct access, with a fixed record length of 1024 bytes. Except for the first two records, there will usually be 256 floating point (R*4) values written. Raw data files contain 512 fixed point (I*2) entries and the shaft encoder (Indexer) has only 128 double precision (R*8) values per record. It is important to note that a negative time is inserted in the shaft encoder file each time a "zero crossing" of the rotor occurs. This artifice is the mechanism used in the DAS to isolate each turbine revolution and permit ensemble averaging of the individual blade passages.

3.2 External Files : VAR File Format

In a research environment it is imperative that maximum flexibility is provided with regard to the ability to process information. In the course of reducing the data recorded by the DAS, there will be many occasions when information will be needed from or by other processes. In consideration of this need, all files of this nature must conform to some standard. The convention used by the DAS for these external files, for Ascii text, is outlined in the table appearing below and will subsequently be referred to as the "VAR" file format. A modified input version is also supported and will be presented at the end of this section.

#	Contents	Description
1	Story	Alphanumeric text (80 characters max)
2	X_Typ, Y_Typ, Units	X,Y types (e.g. 1=Time), Units (0=SI,1=ENG)
3	N_X, N_Y	# of Xs and Ys (N_X=1 or N_X=N_Y only)
4	(X_Lab(i), i=1,N_X)	X label(s) (8 characters max)
5	(Y_Lab(i), i=1,N_Y)	Y label(s) (8 characters max)
6	N_Dat	Number of data points in the X,Y arrays
7a	(X(j,1), [Y(j,i),i=1,N_Y], j=1,N_Dat)	X,Y data if N_X=1
7b	([X(j,i),Y(j,i),i=1,N_Y], j=1,N_Dat)	X,Y data if N_X=N_Y

All text and labelling information must be enclosed within single quotes (apostrophe) with at least one blank separating multiple entries. The value appearing under the "#" heading, except for 7, is the line number if the file were to be viewed in a text editor. 7a,b actually refer to N_Dat separate lines (records) of x,y information. Although convenient for text editing, Ascii text files are wasteful of disk space where numerical information is concerned. Therefore, VAR data may also be stored in binary variable length record files. For binary VAR files, items 1, 4 and 5 do not have single quote delimiters nor are there blank separators. Each item does, however, contain the maximum number of characters of information provided for each type specified. Routines which read input from VAR files are able to distinguish how the data are organized automatically. On the other hand, routines which create VAR files require the user to choose between Ascii text and binary output.

Two utility programs (v2dfmt and d2vfmt) are available to perform the bi-directional conversions needed to support these external files. Any DAS internal file may be converted to an external file. However, only tertiary data may be created from an external source. In general, VAR files are not restricted as to the content or number of independent variables. This is not the case for VAR <=> DAS conversions. The independent variable must be time and a single dependent variable must be specified to be consistent with DAS requirements.

3.2.1 Modified VAR File Format

There are many times when it would be convenient to have a shorthand means for specifying the variation in an independent variable without having to enter all its values into a file. Within the context of the VAR file format it is possible to provide such a capability and still preserve its overall appearance. The means chosen to accomplish this involves the N_X parameter defined above. If it is specified as a negative integer, its magnitude represents the number of distinct X regions, having a fixed spacing, to be provided.

Given an initial X location, Xref, it is then only necessary to provide the spacing and number of points to completely define the X domain. The effect on the VAR file appearance for this modification is large enough to warrant the following table:

#	**Contents**	**Description**
1	Story	Alphanumeric text (80 characters max)
2	X_Typ, Y_Typ, Units	X,Y types (e.g. 1=Time), Units (0=SI,1=ENG)
3	N_Xreg, N_Y	# X regions (negative) and # Ys (positive)
4	X_Lab	X label (8 characters max)
5	(Y_Lab(i), i=1,N_Y)	Y label(s) (8 characters max)
6	Xref, (Del_X(i),N_Pts(i), i=1,N_X)	Initial X and spacing,#points per X region (N_X = Magnitude of N_Xreg)
7	([Y(j,i),i=1,N_Y], j=1,N_Dat)	Y data at implied X(j) values (N_Dat = Sum N_Pts(k) for k=1,N_X)

This form provides an especially convenient analog to the channels which are recorded asynchronously. By way of example, suppose there is a channel whose sample rate source is the clock. This would mean that CDR parameter LSS would contain "C" and LSR the corresponding sampling frequency in Hz. Again, let us assume LSR = 10000, or 10kHz. Since the DAS works in time units of seconds, a sample spacing of 10^{-4} seconds would be implied. If we specify CDR parameter NSAMP1=2001, a context which can be recreated in a modified VAR file format exists.

Assuming input lines 3 and 6 in a modified VAR file contained "-1,1" and "0,1e-4,2001", the resulting X array would be equivalent to the times at which data were recorded for the hypothetical channel discussed above. A representative piece of Fortran code used to construct the X array is presented below for the edification of the user.

```
parameter    (maxreg = 3)
real*4       Del_X(maxreg), X(65536), Xbegin, Xref
integer*4    i, j, nptsum, N_Pts(maxreg), N_Xreg
nptsum = 0
Xbegin = Xref - Del_X(1)
do i = 1, iabs(N_Xreg)
  do j = 1, N_Pts(i)
    X(j + nptsum) = Xbegin + j * Del_X(i)
  enddo
  Xbegin = Xbegin + N_Pts(i) * Del_X(i)
  nptsum = nptsum + N_Pts(i)
enddo
```

By implication: N_Xreg=-1, Xref=0.0; Del_X(1)=.0001; and, N_Pts(1)=2001.

4. Non-Data (SOIL) Files

In the course of implementing the DAS design, many source, object, include and library files were created. Henceforth the acronym "SOIL" will be used as a collective reference to these four distinct types. The user id under which all such files are stored is ′/usr/atarr′ with an appropriately named subdirectory based on usage. The following list relates each SOIL file with its extension and subdirectory:

Ext	Subdirectory	File Usage
c	src	C source code
f	src	Fortran language source code
o	src	Compiled object code (C or Fortran)
h	include	C "include" file
inc	include	Fortran "include" file
a	lib	Library of DAS routines

All SOIL files have the form filename.extension. The extensions used are listed above but filenames are chosen mnemonically to suggest the general or specific areas to which they apply.

4.1 Include Files

Most of the routines which comprise the capabilities of the DAS rely wholly or in part on the information contained in the so-called "include" files. These files are used to type, allocate and collect variables which share one or more characteristics and provide a shorthand means for inserting the related statements into source code modules. The C language include files have the extension "h" while Fortran has "inc" appended to its filenames.

If the user hopes to effectively use the Fortran routines provided, it is likely that he will have to include code from some or all of the DAS include files (i.e. bstruc.inc, dasres.inc, dstruc.inc, and revdat.inc) with his source statements. The form of the statement required is include ′?.inc′, within columns 7 thru 72, with ? = any of the filenames listed above. Within these files, Fortran structures and labelled commons are used to communicate information between the routines containing them. The language reference manual gives satisfactory examples for both concepts if the user needs clarification on their meaning.

4.2 Modifying DAS Software

Although a copy of the executable files representing the individual DAS processes is placed in /usr/local/bin, all of the pertinent source and object code reside in /usr/atarr/src as does Makefile. The UNIX make utility uses the contents of Makefile, which defines all the file dependencies, to create the DAS executables. If changes in the software affecting any of the formal DAS processes are to be made, they should be fully checked out before the system manager copies them to /usr/local/bin for general usage.

4.3 DAS Resources File : dasres.inc

The DAS supports a number of capabilities which refer to numbered lists of items. For instance, there are lists of sensor calibration, physical units, digital filter and algorithm types maintained in the DAS software. During test setup an item number from each of these lists must be specified for each channel to be recorded. The file 'dasres.inc' contains the labelled common variable names which link each item number, for a given type, with its pertinent value. The list definitions are established via data initializations in the Fortran block data subprogram resident in file 'define.F'.

The manner in which the DAS relates item numbers in a CDR to stored information is straightforward. For instance, suppose the CDR to a source array of data values has UNITYP=I and UNISYS=J. Then, UN_TYP(I) and UN_LAB(I,J) are the character array elements which contain descriptions for the respective data type and measurement units to be used by the DAS in conjunction with plotted or tabulated output. Consult Table 1 at the end of the chapter to examine the combinations of data type and units label descriptions which I and J offer.

The DAS limit for I is determined by the parameter MAXPUN which is set by block data subprogram DEFINE. At the time of DAS delivery, MAXPUN was 23. This number may be extended to 99 by making appropriate changes in the source code and performing the steps necessary to create new executable files. The only measurement systems supported are SI and English (J=0 and 1). The system manager can easily extend the number of unit types but adding a measurement system is more complicated due to the intersystem conversions required.

4.4 DAS Data Reduction Algorithms

The user accesses the various data reduction algorithms through a pulldown menu option in the data reduction process (DRP). The number of algorithms supported by the DRP is currently limited to six. There is no limit to the number which can be incorporated but some level of programming skill is needed for each addition. A description of the precise methodology is beyond the scope of this document but some explanation is warranted.

The DRP consists of two essential parts: the User Interface (UI), written in "C"; and, the Computational Background (CB), predominantly Fortran routines. The UI communicates with the CB via subroutine DIRECT which is passed pertinent run and channel structures for the timing data and the source and target arrays. An entry in the CDR structure for the target array designates the algorithm number to be used, if any, to produce the required output. Subroutine DIRECT uses the number to make the appropriate calculation(s) or subroutine call. The following list enumerates the algorithms in effect at the time of DAS delivery:

#	**Algorithm**
1	Converts counts to volts
2	Turbine speed (RPM) computed from shaft encoder times
3	Kulite, Endevco and Baratron counts or volts to pressure
4	RTD, ThinFilm and Thermocouple voltage counts to temperature
5	Temperature to heat rate using Cook-Felderman (ThinFilm gauges)
6	Stanton # from heat rate, temperature and system values

The user will have to keep a separate list to account for each new algorithm to be added to the DAS data reduction capabilities.

	Type	**SI Label**	**English Label**
i	**un_typ(i)**	**un_lab(i,0)**	**un_lab(i,1)**
0	Nondimensional	blank	blank
1	Time	seconds	seconds
2	Power	Watts	BTU/second
3	Energy	Joules	BTU
4	Enthalpy	Joules/kg	BTU/lbm
5	Heat Rate	Watts/m^2	BTU/ft^2-s
6	Force	Newtons	lbf
7	Pressure	Pascals (N/m^2)	psi
8	Acceleration	meters/s^2	feet/s^2
9	Mass	kilograms	lbm
10	Density	kg/m^3	lbm/ft^3
11	Molecular Wt.	kg/mole	lbm/mole
12	Gas Constant	Joules/mole-K	lbf-ft/mole-R
13	Specific Heat	Joules/kg-K	BTU/lbm-R
14	Temperature	Kelvin	Rankine
15	Mass Flow Rate	kg/s	lbm/s
16	Length	meters	feet
17	Area	square meters	square feet
18	Volume	cubic meters	cubic feet
19	Speed	meters/second	feet/second
20	Rotation Rate	RPM	RPM
21	Stanton Number	STANTON#	STANTON#
22	Nusselt Number	NUSSELT#	NUSSELT#
23	Voltage	Volts	Volts

TABLE 1. Physical Units Labels in SI and English Measurement Systems

5. Gauge Labels

The DAS will deal with measurements from a large number of gauges placed strategically within the atarr. Although gauge serial numbers are fixed, labels are assigned on a run to run basis. If a systematic approach to naming each gauge is taken, its label will uniquely define its purpose in any turbine experiment.

Since the primary measurements being made are pressure and temperature, gauge labelling conventions will only apply to them. Gauges for other types of measurements have no naming restrictions except for the eight alphanumeric character limit.

5.1 Temperature and Pressure

Only capital letters will be used to define temperature and pressure gauge labels. The first character will contain a "T" or "P" depending on which measurement applies. The second is dependent on the axial position of the gauge from the following list:

S	==	Supply Tank	X	==	Dump Tank
U	==	Upstream Rake	D	==	Downstream Rake
V	==	Vane	R	==	Rotor
T	==	Test Section			

For the test section or either of the tanks a gauge sequence number is appended to complete the label. While the stage components next require a blade number, the rakes need a letter corresponding to the gauge's circumferential position [A-Z]. The relationship between a specific letter and its clockwise, angular, separation from a prescribed reference point is defined by the test engineer prior to each test and is maintained as part of the facility run log.

The next label position is used to indicate spanwise location. A number is entered for the rakes but the vane and rotor need a single letter (H for hub, M for midspan and T for tip). Finally, vane and rotor gauges require a sequence number for variations along the pertinent foil surface.

When the DAS is extended to support capton gauges, the top and bottom temperatures will be denoted by placing "T" or "B" in the character position immediately following the gauge sequence number. The DAS does not require a test engineer to use the gauge naming convention profiled here but some structured approach to defining labels is strongly recommended. Since gauge labels are used as filenames, it is clear that the files named "TS1.0" and "TS1.1" contain the respective voltage and temperature time histories recorded on a gauge in the supply tank.

5.2 Gauge Labels for Standards Measurements

The convention outlined above is equally applicable to gauges which measure pressure and temperature standards. Such channels are denoted by prefixing an "S" to a label name which was created using the same methodology. Applying this rule would designate a channel with the label "STTn" as a temperature standards gauge which is installed somewhere other than on a rake or stage component in the test section. Schematically, the gauge labelling convention might look like this:

```
                               S | X | T         Number
Prefix   +   P | T   +         V | R     +   Blade Number   +   H | M | T   +   Number
                               U | D     +       [A-Z]      +    Number
```

where the operator "|" indicates a choice between the connected items.

6. Data Reduction

For a given experiment the data reduction process may be invoked at any time subsequent to data acquisition. Fundamentally, the DRP gives the user complete control over the evolution of a channel time history from raw voltage counts to engineering units and so forth. Each step in the process creates a unique data file which is named according to DAS conventions described elsewhere in this document. All such files are available for plotting (listing) or further calculation from a displayed list of options.

6.1 Sensor Calibration Files

Each sensor used in the atarr is identified by a unique serial number. The file containing the laboratory standard calibrations for all the sensors is contained in the sensor calibration file (SCF) which resides in the home directory of the facility. The SCF is an Ascii file consisting of a single line for each sensor which starts with an integer corresponding to one of the following entries:

1	==	Kulite or Endevco	5	==	Thermocouple
2	==	RTD	6	==	Position
3	==	Thin Film Heat Transfer	7	==	Tip Clearance
4	==	Baratron	8	==	Torque

The serial number (up to 12 alphanumeric characters) is entered next followed by the gauge sensitivity. A single digit, zero for absolute voltage (V) or one for relative voltage (ΔV) plus seven fit coefficients, in increasing powers of V or ΔV, complete the entry. Seven coefficients are required to support a maximum degree of fit equal to six. If a second degree polynomial was to be specified, the first three coefficient entries would be relevant and the final four entries would have to be zeroes.

To properly operate the DAS requires online calibrations immediately before and after data has been taken for a turbine experiment. Most pressure gauges in the supply tank and test section will record voltages over a range of set points (pressures) controlled by valves operated by the test engineer. The sequence of operations required to arrive at a set of pre-test calibrations is described in Figure 3. The voltage time histories taken at each set point are stored, if the user requests, in a file named using the gauge serial number with an extension of "t_pre" or "t_post" dependent on when the process takes place relative to the actual test. These files are binary with the first record containing, in order, the run number (I*4), date (A*9), number of set points (I*4), and the number of time samples (I*4). For each set point taken a record is written containing the pressure (R*4) and the sampled voltages (I*2).

At the completion of the fill/calibration process, the set point pressures are fit as a function of the mean voltage in the recorded time interval. The coefficients of the polynomial fit are appended to an Ascii file also named using the gauge serial number but with the extension "online". Each entry consists of the run#, date, ´ pre´ or ´post´, and the seven coefficients written with the format ´(i3,2x,a9,2x,a4,2x,1p7e14.6)´. The files created during the process all reside in the appropriate run directory for the turbine being investigated. The fit coefficients are placed in either the "c_pre" or "c_post" arrays in the RCDF for the sensors considered.

The pressure versus voltage fit used to convert the recorded volts during the actual test may be determined at any time during the data reduction process. The pre-test calibration coefficients in array c_pre will be the defaults in the absence of a user determined polynomial fit. If the option to alter the calibration coefficients is chosen in the DRP, then graphs of $p(V)$, over the full scale voltage range, will be displayed using the pre-test, post-test and laboratory standard calibration coefficients. A weighted average of the three existing fits is the means the user will have to select a revised fit. A least squares approximation to the resulting curve defines the preferred polynomial coefficents to be placed in the RCDF. The RCDF array "c_used" will contain a copy of these coefficients. An Ascii file named for the gauge number with the extension "actual" will be created or appended to whenever array "c_used" is defined. Each file entry contains the run#, date, and seven coefficients according to format ´(i3,2x,a9,2x,1p7e14.6)´.

Figure Deleted to Avoid Confusion with Changes in System Design

Figure 3. Pre-Test Calibration Procedure

7. DAS Utility Programs

Overview

The DAS formal processes give users the ability to setup, acquire and reduce atarr test data. The DAS design permits the extension of the processes to incorporate even more capabilities. However, there are a number of routine tasks which serve the user better if they are kept separate.

Individual programs have been developed to perform the tasks which have been identified to date. This chapter provides brief descriptions of each program's purpose and how it is used. All the DAS utility programs have usage information which will be displayed on the screen if its name is entered at an atarr terminal.

7.1 Program bdplt

By entering the command *bdplt* the user can generate Tekronix plots of information contained in file "BD.PLT" that was output from the *mm* or *facility* programs. As with all Tektronix plotting, the UNIX environment parameter "GTYPE" must be defined as "xterm" (the default on atarr). Otherwise, whenever a plot is requested the screen will be filled with nonsense. If "BD.PLT" is not in the current working directory, program bdplt terminates with an informative message.

7.2 Program ccp

Computes channel calibration constants to be saved in the CDR of user selected DAS raw data files. The current working directory is searched for a single RCDF name of the form "run*.rcdf" (* = integer run number). An informative message is printed and the program stopped if this requirement is violated.

The user selects the channel(s) to be calibrated and responds to program prompts for an action to be taken. All activity which results in changes to a channel calibration are recorded in a) the CDRs of the RCDF and raw data file; b) file run*.log; and, c) in directory /usr/atarr/files in a file of the form "#.lab" (# = pertinent hardware serial number).

7.3 Program d2vfmt

Places a time history from a DAS internal file onto a file in VAR format. The program currently accepts CDR values lsr and nsampl as the only valid sampling parameters. The program is invoked with a command line of the form "d2vfmt [-x] file1 [file2...filen]". The optional "x" parameter dictates the form of the VAR output: "a" for Ascii text; or "b" for binary data (default). Output filenames are derived from the input files simply by appending ".var".

7.4 Program plotvar

Create Tektronix plots from VAR format input data files. The program is invoked as follows: "plotvar file1 [file2...filen]". Each file specified is processed in the given order and the user may interactively create a finished plot to suit his needs. All information necessary to effectively use the program is provided by clearly worded prompts at the bottom of the screen display.

7.5 Program standards

Compute standards channels for the DRP. Although it is usually run by the DRP, standards can also be run standalone. The program is invoked using the syntax "standards filename" and the user responds to various reverse video prompts appearing at the bottom of the 80 column by 24 row text display. The current working directory must contain the given filename (an RCDF) and all associated DAS data files for the turbine run which was being investigated at the time the program was executed. The DRP takes these factors into account before invoking standards and the user is returned to the DRP upon program completion.

7.6 Program viewdas

Permits the inspection of information in the RDR and/or CDR of DAS data files based on the option selection parameter in the command line used to invoke the program. The program is executed by entering "viewdas -x file1 [file2...filen]" at one of the atarr terminals. Replace "x" with "c" for CDR; "r" for RDR; and, "b" (the default) for both CDR and RDR access.

Pertinent run or channel descriptor parameters are displayed for each file specified by the user. The opportunity to alter them will be offered by the program only if certain internal criteria are met.

In practice it is expected that only channel dependent information will be modified. However, the ability to alter run dependent information is available and, if required, the following steps should be observed to ensure continuity:

1. All primary, secondary and tertiary files for the run should be deleted;
2. Invoke program with the command line "viewdas -r *.0";
3. Make the required correction(s);
4. Exit the program;
5. Regenerate required primary, secondary and tertiary files.

Any correction(s) will automatically appear in the RDR for all channels subsequently reduced using the DRP.

Whenever a channel's CDR is altered the impact of change on subsequent data reduction must be considered. No practical means exists to safeguard data from inadvertent error caused by using viewdas. The importance of having the capability it provides hopefully outweighs the occasional inconvenience resulting from its use. Also, most errors can be corrected and the affected DAS files regenerated using the DRP.

7.7 Program v2dfmt

Converts a VAR data file to DAS format. Only data with fixed time spacing for the abscissa may be converted to DAS format. In the case of implicit x data only the first region will be stored. Explicit x data will be checked for nominal fixed spacing and only qualifying samples will be stored.

The program is executed by entering the command "v2dfmt filein fileout". Where, filein is the name of an existing VAR file and fileout is a unique filename of up to eight characters with a ".3" extension. It is important to note that at least one raw data file must exist in the current working directory for v2dfmt to work properly.

7.8 Program waveforms

Permits the bulk plotting of DAS data files for the intended purpose of analyzing individual wave forms. Presently, the abscissa is restricted to sample number. The files to be plotted are defined on the command line used to invoke the program. For example, to plot the data from all raw data files in the current working directory, the command line "waveforms *.0" is used. Plots of voltage counts versus sample number would be produced on the system printer/plotter. Similarly, "waveforms -s *.0" would cause the same plots to appear on the screen.

If the medium chosen is the screen, information pertinent to the file currently being processed is placed on the screen for user action. This usually means the the keyboard becomes *hot*. When this condition exists any key touched (except Alt, Ctrl and Shift) is sensed by the program and the appropriate action is taken. In general, the Esc, Q or X keys will cause all processing to end. Whenever a prompt requesting the user to enter data is displayed, the "Enter" key (<CR>) must be struck to terminate the input line.

A word of caution is in order regarding the *hot* keyboard. A simple "tap" is recommended since a "push" may result in multiple occurrences of the key touched if the keyboard's sensitivity is high.

8. Source Routine Summaries

Overview

This chapter defines the formalism required to incorporate various elements of the DAS libraries into user defined programs. Except where noted, each entry is a Sun Fortran subroutine, function or block data subprogram. Each routine (function) is a part of a file containing one or more source entities. Abstracts are listed in alphabetical filename order with a capsulized description of each routine's purpose followed by, in most instances, a listing of the formal parameters in the proper calling sequence. Each parameter listing consists of the following five headings:

Heading	**Description**
Note	Number corresponding to an explanatory note following list
Parameter	Variable name
Type	Variable type
I|O|B	Parameter is input (I), output (O) or both (B)
Description	Brief explanation of content or usage

Appendix C lists all subroutine and block data and function subprograms mentioned in this chapter in alphabetical order.

8.1 adjlab.F {Subroutine ADJLAB}

Routine to left adjust the input characters in RLABEL such that NCOUT is the count from the first to the last nonblank character.

Note	**Parameter**	**Type**	**I|O|B**	**Description**
	RLABEL	C*?	B	Character string to be adjusted
	NCOUT	I*d	O	Output number of contiguous characters

8.2 axdata.F {Subroutine AXDATA}

Routine to permit the specification of axes dependent information via the screen and keyboard. Logical unit 19 is used for this purpose and is opened if it doesn't already exist.

Note	Parameter	Type	I|O|B	Description
	DALTER	L*d	B	T if XST,XFIN,DELTAX,SFDOFX or NXPPTD change; else, F.
	LOGOFX	L*d	B	T if log axis; else, F.
	XMIN	R*4	B	Minimum of actual data.
	XMAX	R*4	B	Maximum of actual data.
	XST	R*4	B	Scaled plot axis minimum (min decade for log axis).
	XFIN	R*4	B	Scaled plot axis maximum (max decade for log axis).
1.	DELTAX	R*4	B	Scaled plot axis interval (<0 for descending ticks).
1.	SFDOFX	I*d	B	Scale factor decade (scaled=actual/10^{SFDOFX}).
2.	NXPPTD	I*d	B	Number places past decimal in tick mark labels.
	AXIS	C*1	I	'X' or 'Y' as applicable.
	NUMAX	I*d	I	Subscript for axis being treated (0, 1 or 2). (If zero, no subscript is indicated in prompts.)

AXDATA Notes:

1. Does not apply for log axis;
2. Negative of number of cycles for log axis.

8.3 beta.F {Subroutine BETA}

Computes $\rho c k$ at a given temperature for the substrate material prescribed in block data subprogram DEFINE. At the time of DAS delivery the material for thin film heat transfer gauge substrates was defaulted to pyrex.

Note	Parameter	Type	I|O|B	Description
	T	R*4	I	Temperature (degrees K)
1.	ANS	R*4	O	$\rho c k$

BETA Note:

1. ρ = density; c = specific heat; and, k = thermal conductivity.

8.4 convrt.F {Subroutine CONVRT}

Converts input values from SI to English or vice versa dependent on the mode parameter. The data stored in the labelled commons defined in file dasres.inc provide the required factors to convert the unit types supported by the DAS. The initialization of these resources is performed by the block data subprogram DEFINE.

Note	Parameter	Type	I|O|B	Description
	UNITYP	I*d	I	Integer from zero to MAXPUN defining conversion
	MODE	I*d	I	0 = SI => English; 1 = English => SI.
	INPRAY	R*4	I	Array of values to be converted
	OUPRAY	R*4	O	Array of converted values
	N	I*d	I	Number of elements in INPRAY
	UNILAB	C*15	O	Character string containing units label
	*	N/A	O	Alternate return for errors

8.5 cook.F {Subroutine COOK}

Uses the Cook-Felderman method to compute the heat rate from an input N-point temperature time history. The version implemented uses the recorded variable Δt and is computationally burdensome due to the $N^2/2$ square root calculations required. The turbine data used to checkout the software had 12k samples. A parallel version, which performed the same calculation with a fixed Δt, is faster because the solution equation could be factored so that only N square root calculations were needed. For the ten temperature profiles compared the difference in results produced by the two methods was barely discernible. Further analysis is required before it can be confidently substituted in future experiments.

Note	Parameter	Type	I|O|B	Description
	NPNTS	I*d	I	Number of points in all arrays
	TTQ1	R*8	I	Array of time points
	TTQ2	R*4	I	Array of temperature values
	TTQ3	R*4	O	Computed heat rate values

8.6 declab.F {Subroutine DECLAB}

Routine to place a decade label of the form 10^i at specified pixel coordinates. Used primarily by subroutine LOGAXS but may be of some use for low level plot development.

Note	Parameter	Type	I|O|B	Description
	ICYCX	I*d	I	X pixel coordinate at lower left hand corner
	ICYCY	I*d	I	Y pixel coordinate at lower left hand corner
	ICYCNO	I*d	I	Power of ten to be drawn

8.7 define.F {Block Data Subprogram DEFINE}

Initializes the parameters contained in the labelled commons associated with the include files ´dasres.nc´, ´revdat.inc´, and ´tfdata.inc´.

8.8 deflim.F {Subroutine DEFLIM}

This routine provides the parameters needed to specify a coordinate axis where the user has specified the tick mark minimum, maximum and spacing to be used. If the values are consistent, the axis parameters returned will reflect this fact. Otherwise, the "best" set of parameters will be determined based on an interpretation of the user's specifications. Where subroutine PLMNMX is used to auto-scale axis values, DEFLIM is constrained to operate within the specified endpoints and accept the given tick mark spacing if its internal criteria are met.

Note	Parameter	Type	I|O|B	Description
	TICMIN	R*4	I	Minimum axis value
	TICMAX	R*4	I	Maximum axis value [If TICMIN > TICMAX, axis tick marks will be in decreasing order. If TICMIN = TICMAX, subroutine PLMNMX will be used.]
	TICSPC	R*4	I	Tick mark spacing between ticmin and ticmax [Used only if (TICMAX-TICMIN)/TICSPC is an integer. If TICSPC = 0, tick mark spacing is computed based on internal criteria. Tick marks will be drawn in descending order if TICSPC < 0.]
	START	R*4	O	Minimum scaled axis value
	FINISH	R*4	O	Minimum scaled axis value
	DELTA	R*4	O	Tick mark spacing between START and FINISH
	NINTS	I*d	O	Total number of tick mark intervals
	POW	R*4	O	Scaled value = POW * actual value
	SFD	I*d	O	Scale factor decade [POW = 10**(-SFD)]
	NPLCS	I*d	O	Number of decimal places in tick mark labels

8.9 derivn.F {Subroutine DERIVN}

Computes the derivative of an equally spaced function using a 3 or 5-point Lagrangian formulation.

Note	Parameter	Type	I|O|B	Description
	F	R*8	I	Array of input values
	DFDT	R*4	O	First derivatives at each point
	DT	R*8	I	Spacing between input points
	N	I*4	I	Number of points in F and DFDT arrays
	NPOINT	I*4	I	Number of Lagrangian points (3 or 5)
	*	N/A	O	Alternate return for errors

8.10 direct.F {Integer Function DIRECT}

Interprets requests from the DAS data reduction process user interface and directs all computational activity based on the contents of the CDR and RDR. The value of I = DIRECT(...) is one if successful and zero if the request could not be processed.

The source array(s) of input values must be operated on to produce a target array of output values. The DAS accesses a library of algorithms which are numbered for use by data reduction routines. The number associated with a particular algorithm is used here to perform pertinent calculations or subroutine call(s). Physical unit types are also enumerated and the number corresponding to the target array is placed in the output CDR. Most DAS internal calculations are done in SI and all primary data files are stored in that system. However, since users may work exclusively in ENGLISH, the status of the units flag is used to output secondary and tertiary target values in that system.

Note	Parameter	Type	I|O|B	Description
1.	IRDES	REC	I	RDR for INIDAT or INRDAT array
1.	ICDES	REC	I	CDR for INIDAT or INRDAT array
2.	INIDAT	I*2	I	Source array 1 (voltage counts)
3.	INRDAT	R*4	I	Source array 1 (engineering units)
1.	JRDES	REC	I	RDR for R4DATA array
1.	JCDES	REC	I	CDR for R4DATA array
4.	R4DATA	R*4	I	Source array 2 (engineering units)
1.	TRDES	REC	I	RDR for TIMDAT array
1.	TCDES	REC	I	CDR for TIMDAT array
	TIMDAT	R*8	I	Times (sec) at indexer pulses
1.	ORDES	REC	B	RDR for OUTDAT array
1.	OCDES	REC	B	CDR for OUTDAT array
	OUTDAT	R*4	O	Target array
	WKAREA	R*4	B	Work area for incidental calculations

DIRECT Notes:

1. REC (record) size is 1024 bytes
2. Only if ICDES.CTYPE = 0
3. Only if ICDES.CTYPE > 0
4. Undefined unless JCDES.CTYPE > 0

8.11 evlsp3.F {Subroutine EVLSP3}

To evaluate cubic spline at specified values of independent variable (XEV) using the coefficients determined by a prior call to subroutine SPLIN3. Spline may be evaluated for function value, derivatives or integral (see KODE).

For the integral mode (KODE=-1), XEV(1) is taken as the origin for integration, and YEV(1)=0 is correspondingly set to zero. The output value YEV(K) will be the integral from XEV(1) to XEV(K). This mode provides a method of numerical integration. For XEV values outside the original X range (X(1) to X(N)), extrapolation is based upon values in the first interval or last interval.

Program does not check that X and XEV values are in ascending order. If they are not, errors are probable, or else an infinite loop may occur. An alternate return is provided to bypass using results if the KODE value is not valid and either N or NEV is less than 1.

Note	Parameter	Type	I|O|B	Description
	N	I*d	I	Number of points at which original fit made (Length of X, Y, A, B, C arrays)
	X	R*4	I	Array of X-values for original fit (Values must be in ascending order)
	Y	R*4	I	Array of Y-values for original fit
	A,B,C	R*4	I	Arrays of spline coefficients from SPLIN3
	NEV	I*d	I	Number of points at which evaluations to be made (Length of XEV and YEV arrays)
	XEV	R*4	I	X value(s) at which evalution(s) to be made (Values must be in ascending order)
	KODE	I*d	I	Code value for what YEV output is to be -1 Integral (see remarks above) 0 Y 1 Y' (first derivative) 2 Y" (second derivative) 3 Y"' (third derivative)
	YEV	R*4	O	Output value(s) per KODE value
	*	N/A	O	Alternate return for errors

8.12 galpha.F {Subroutine GALPHA}

Routine to draw a given character string in some orientation beginning at specified pixel coordinates. Similar to subroutine GLABEL but not as restricted in orientation possibilities. GALPHA provides the capability of positioning with respect to the string's center in addition to the usual lower left corner frame of reference. Formal parameters are input only and output is placed on the graphics medium.

Note	Parameter	Type	I|O|B	Description
1.	IXC,IYC	I*d	I	Pixel coordinates for placement of LABEL
2.	ANGLED	R*4	I	Angle of LABEL
	INCHES	R*4	I	Size of each character in inches
	LABEL	C*n	I	String to be placed at IXC,IYC (n=NCHARS)
	NCHARS	I*d	I	Number of characters in LABEL

GALPHA Notes:

1. If IXC>0, the string will be centered and rotated about (IXC,IYC); else, (-IXC,IYC) are at the lower left corner of the string;
2. Counter clockwise degrees with respect to the horizontal.

8.13 getray.F {Integer Function GETRAY}

This integer function provides a general means for dynamically allocating space for any array associated with a pointer. The value returned is always unity indicating a successful allocation.

Note	Parameter	Type	I|O|B	Description
	NBYTES	I*d	I	Number of bytes to be allocated
	P	POINTER	I	Pointer to array for which space is needed

8.14 getres.F {Integer Functions GET_I2, GET_R4 and GET_R8}

Contains the integer function subprograms GET_I2, GET_R4 and GET_R8 named to suggest the kind of DAS resource file to which each is designed to provide access. All raw data files contain recorded I*2 voltage counts and need GET_I2 to provide access to them. Except for the shaft encoder time history contained in 'INDEXER.1', all other DAS data files require GET_R4 to access the stored R*4 information. Of course, GET_R8 extracts the R*8 times from 'INDEXER.1'.

Note	Parameter	Type	I|O|B	Description
	RDES	RECORD	O	DAS file RDR
	CDES	RECORD	O	DAS file CDR
	FILNAM	C*?	I	DAS filename to open
	P	POINTER	I	Pointer to data output array

Usage:

```
I = GET_??(RDES,CDES,FILNAM,P)
```

Where, ?? are the two-characters (I2, R4 or R8) of the function; the parameters are as described above; and, I an integer which is nonzero if successful and zero, otherwise.

The RDES and CDES parameters must be included in record statements in the calling routine which are referenced to the structures in file 'dstruc.inc'. This is accomplished by placing the following three statements in the source file:

```
INCLUDE 'dstruc.inc'
RECORD /RUNDES/ RDES
RECORD /CHDES / CDES
```

Data output arrays will be dynamically allocated and require the use of pointers and pointer based arrays. Casual Fortran users may not be acquainted with them but can use them confidently if the first of the following statements appears in the source file with the appropriate type statement:

```
POINTER    (P,V)
INTEGER*2 V(*)   [For use with GET_I2.]
REAL*4    V(*)   [For use with GET_R4.]
REAL*8    V(*)   [For use with GET_R8.]
```

Caution! Each pointer is associated with a single data array. If more than one array is needed for an application, each must be paired with a unique pointer and appear in a separate "type" statement. For instance, if two R*4 data files are needed for an application which also requires the shaft encoder times, the source statements might include:

```
RECORD /CHDES / CDES1, CDES2, CDEST
POINTER    (P1, ARRAY1), (P2, ARRAY2), (P3, TIMES)
REAL*4     ARRAY1(*), ARRAY2(*)
REAL*8     TIMES(*)
```

8.15 get1c.c {Integer Function GET1C}

Integer function to return the Ascii character number corresponding to the key pressed. Used primarily to process the user response to a screen prompt. It is important to note that the interrupt processing routine INITSIG is disabled since Ctrl-C is interpreted as a control character (i.e Ascii numbers < 32). Therefore, some provision must be made to terminate execution, since even Ctrl-Z will not work. To use GET1C in a Fortran context, simply declare I and GET1C as integer types and execute the command "I=GET1C()". The value of I will reflect the Ascii number of any key struck.

8.16 glabel.F {Subroutine GLABEL}

Routine to draw a given character string in some orientation beginning at specified pixel coordinates. Subroutine GALPHA provides the same functionality, with a different calling sequence, and can be used in place of GLABEL which is retained primarily because it is referenced by other library subroutines (e.g. LINAXD). All formal parameters are input only and output is placed on the graphics medium.

Note	Parameter	Type	I|O|B	Description
	INX,INY	I*d	I	Pixel coordinates at lower left corner of the first character to be drawn
	IORIEN	I*d	I	Multiple of 90 degrees character orientation (Magnitude of IORIEN < 4)
	LABEL	C*n	I	String to be placed at INX,INY (n=NCHARS)
	NCHARS	I*d	I	Number of characters to be drawn

8.17 grdlin.F {Subroutine GRDLIN}

Draws grid overlays in x and y directions. All formal parameters are input integers only. Output is to the existing graphics medium.

Note	Parameter	Type	I|O|B	Description
	IUPD	I*d	I	IUPD =0 for x; =1 for y direction
	NINTVS	I*d	I	Number of grid intervals to be drawn
	IX0,IXN	I*d	I	Pixel extent of the x axis
	IY0,IYN	I*d	I	Pixel extent of the y axis

8.18 interrupt.c {Subroutine INITSIG}

Permits the activation of a block of code whenever Ctrl-C is encountered. The interrupt flag, an integer, must be initialized anywhere prior to a program segment where it is queried. In the example below, the user will be given the opportunity to stop or continue anytime Ctrl-C is encountered during the execution of loop 100. It is necessary to reset the interrupt flag to zero before calculations continue.

```
      character   c
      integer     intflag
      call initsig(intflag)
      do 100 i=1,100000
        if(intflag.ne.0) then
          write(*,'(a,i6,a)') 'Loop index=',i,'.  Continue (Y/N)? '
   90     read(*,'(a)') c
          if(c.eq.'y' .or. c.eq.'Y') then
            intflag = 0
            go to 100
          elseif(c.eq.'n' .or. c.eq.'N') then
            stop
          else
            write(*,'(a)') '(Y/N)? '
            go to 90
          endif
        endif
        if(mod(i,100).eq.0) write(*,*) i
  100 continue
      end
```

8.19 linaxd.F {Subroutine LINAXD}

Routine to draw and label a linear x or y coordinate axis with labelled tick marks on the existing plot medium.

Note	Parameter	Type	I|O|B	Description
	NINTVS	I*d	I	Number of tick intervals to be drawn
	TICKMN	R*4	I	Tick value at left for X-axis and bottom for Y-axis
	TICKMX	R*4	I	Tick value at right for X-axis and top for Y-axis
	NDP	I*d	I	Number of decimal places in tick labels (0 to 4)
	IUPD	I*d	I	Axis orientation: 0=X at bottom; -1=X at top; 1=Y at left; 2=Y at right.
	IX0,IXN	I*d	I	X-axis pixel boundaries (left, right)
	IY0,IYN	I*d	I	Y-axis pixel boundaries (bottom, top)

8.20 lnblnk.F {Integer Function LNBLNK}

Provides same capability as V77 "LNBLNK" function. For example, the statement I = LNBLNK('String length.') would result in I = 14.

8.21 logaxs.F {Subroutine LOGAXS}

Routine to draw and label a log x or y coordinate axis with labelled decade markers.

Note	Parameter	Type	I|O|B	Description
	NCYCS	I*d	I	Number of complete cycles (decades) to be drawn
	MINCYC	I*d	I	Minimum cycle #
	IUPD	I*d	I	Axis orientation: 0=X at bottom; -1=X at top; 1=Y at left; 2=Y at right.
	IX0,IXN	I*d	I	X-axis pixel boundaries (left, right)
	IY0,IYN	I*d	I	Y-axis pixel boundaries (bottom, top)

8.22 mulplt.F {Subroutine MULPLT}

Routine to draw multiple plots to scan for features. If he chooses, the user may select a time interval for baseline averaging or perform a weighted average over several channels to produce a single supply tank or test section value.

Note	Parameter	Type	I|O|B	Description
	XPDATA	R*4	I	Array of x values
	YPDATA	R*4	I	Array of y values
	NDAPTS	I*d	I	Number of points in XPDATA & YPDATA
	YLABIN	C*8	I	Array of labels for each trace (channel)
	XLEFT	R*4	I	Minimum x value to be considered
	XRIGHT	R*4	I	Maximum x value to be considered
	NTRACE	I*d	I	Number of traces (channels) in XPDATA,YPDATA arrays
	NPLPTS	I*d	I	Array containing number of points per trace
	UTYPIN	C*15	I	Data type string (e.g. 'Pressure')
	ULABIN	C*15	I	Data label string (e.g. 'Pascals (N/m^2)')
	BL_AVG	L*d	I	Logical to select baseline averaging option
	BEGTIM	R*4	B	Beginning time for baseline averaging
	ENDTIM	R*4	B	Ending time for baseline averaging
	BL_NEW	L*d	O	Logical indicating BEGTIM,ENDTIM changed
	SUPPLY	R*4	B	Supply tank value for data type in UTYPIN
	TSTSEC	R*4	B	Test section value for data type in UTYPIN
	NWIDE	I*d	B	Number of plots per screen (page) horizontally
	NHIGH	I*d	B	Number of plots per screen (page) vertically

8.23 ndplcs.F {Subroutine NDPLCS}

Determine the optimum number of decimal places, NOUT, to represent DXIN. If DXIN is considered to be the spacing between tick marks on a coordinate axis to be drawn by subroutine LINAXD, then each tick will be labelled using a format of Fw.d. In this context NOUT is completely analogous to d. If NOUT = 0, it is recommended that a format of Iw be used for tick mark labels. Note that the maximum value of NOUT permitted is 4.

Note	Parameter	Type	I|O|B	Description
	DXIN	R*4	I	Number to be examined
	NOUT	I*d	O	Optimum number of decimal places

8.24 numsin.F {Subroutine NUMSIN}

Routine to input number ranges from logical unit = UNITNO. All formal parameters are INTEGER*d. Input is in the form "N1,N2-N3,N4,N5-N6" where the Ni are integers. A blank may be used in place of a comma and two successive blanks or a slash (/) will terminate further processing of the input line. The number of intervals will be returned in INTVLS and the min, max for each interval will be in the arrays MINNUM, MAXNUM, respectively. The sample input line above would result in INTVLS = 4 and values of MINNUM and MAXNUM as follows:

i	MINNUM(i)	MAXNUM(i)
1	N1	N1
2	N2	N3
3	N4	N4
4	N5	N6

Care must be taken that arrays MINNUM and MAXNUM be dimensioned sufficiently in the calling program. INTVLS contains the number of valid intervals encountered before two successive blanks, a slash, a read error, or end-of-file condition occurred in the input line.

Note	Parameter	Type	I|O|B	Description
	UNITNO	I*d	I	Logical unit number for data input
	INTVLS	I*d	O	Number of intervals found
	MINNUM	I*d	O	Array of starting values for each interval
	MAXNUM	I*d	O	Array of ending values for each interval

8.25 numsrt.F {Subroutines I2SORT, I4SORT, R4SORT and R8SORT}

"NUMSRT" is a collective term which refers to the numerical sorting capability provided for columnar arrays of type I*2, I*4, R*4 and R*8. The subroutine names for the respective types are I2SORT, I4SORT, R4SORT and R8SORT. Except for the names, the calling sequence is the same for the four routines. For example, to numerically sort the first 500 values in the R*8 array "A" the statement "CALL R8SORT (A, INDEXA, 500)" would be used. The array "INDEXA" would have to be allotted at least 500 in a dimension or type statement in the calling routine or the results would be unpredictable. After the call to R8SORT has been executed, the minimum value in array "A" is to be found at location INDEXA(1) and the maximum at INDEXA(500).

The source array is undisturbed by the sort activity and multiple entries are assigned unique indices in indexa but are not necessarily arranged as they were at the start of the process. For instance, if the source array contains 3,1,2,2,2, then the resulting indexa array, ideally, would contain, 2,3,4,5,1. However, any permutation of the three interior values of indexa is equally correct, for a total of six possible solutions.

8.26 num2ch.F {Subroutine NUM2CH}

Routine to convert an I*d value to its character equivalent.

Note	Parameter	Type	I|O|B	Description
	NUMBER	I*d	I	Number to be converted
	CHRNUM	C*?	O	Character string representation
	NCHOUT	I*d	O	Number of characters (≤?)

8.27 pfonts.F {Block Data Subprogram PFONTS}

Stores the vectors (I4VECT) and pointers to initial vector (LOCVEC) used to draw Ascii characters 33 thru 126 listed below. The vectors are "packed" in integer*4 words in array I4VECT of labelled common fntdat which is defined in file "fntdat.inc". For example, Ascii character N's vectors are stored in I4VECT(LOCVEC(N-32)) thru I4VECT(LOCVEC(N-31)-1).

Each entry in I4VECT contains a single vector defining the coordinates of a line to be drawn within a window which is 50 units wide by 70 units high. The window is mapped onto the pixel window for the current graphics character size. The entry 50000070 describes a line segment from (50,0) to (0,70) or a line starting at the lower right corner of the window and extending to the upper left. Five characters (g, j, p, q and y) have descenders requiring each ordinate to be shifted 20 units downward. Since negative numbers cannot be supported conveniently in I4VECT, Ascii characters 103, 106, 112, 113 and 121 must be checked for and their ordinates suitably adjusted before they are drawn.

The vectors defining the character set were developed by W.D.Fryer of Calspan whose contribution is gratefully acknowledeged. The following tabulation lists the parameters required

#	C	P	#V	I	#	C	P	#V	I	#	C	P	#V	I	#	C	P	#V	I
33	!	1	22	1	57	(	25	10	175	81	Q	49	10	340	104	h	72	5	452
34	"	2	2	23	58	:	26	16	185	82	R	50	7	350	105	i	73	4	457
35	#	3	4	25	59	;	27	10	201	83	S	51	11	357	106	j	74	7	461
36	$	4	12	29	60	<	28	2	221	84	T	52	2	368	107	k	75	3	468
37	%	5	17	41	61	=	29	2	223	85	U	53	5	370	108	l	76	3	471
38	&	6	12	58	62	>	30	2	225	86	V	54	2	375	109	m	77	9	474
39	'	7	4	70	63	?	31	13	227	87	W	55	4	377	110	n	78	5	483
40	(	8	5	74	64	@	32	18	240	88	X	56	2	381	111	o	79	8	488
41	)	9	5	79	65	A	33	6	258	89	Y	57	3	383	112	p	80	6	496
42	*	10	3	84	66	B	34	10	264	90	Z	58	3	386	113	q	81	8	502
43	+	11	2	87	67	C	35	7	274	91	[	59	3	389	114	r	82	5	510
44	,	12	12	89	68	D	36	6	281	92	\	60	1	392	115	s	83	11	515
45	-	13	1	101	69	E	37	4	287	93	]	61	3	393	116	t	84	5	526
46	.	14	8	102	70	F	38	3	291	94	^	62	2	396	117	u	85	5	531
47	/	15	1	110	71	G	39	9	294	95	_	63	1	398	118	v	86	2	536
48	0	16	8	111	72	H	40	3	303	96	`	64	5	399	119	w	87	4	538
49	1	17	3	119	73	I	41	3	306	97	a	65	8	404	120	x	88	2	542
50	2	18	6	122	74	J	42	5	309	98	b	66	6	412	121	y	89	2	544
51	3	19	8	128	75	K	43	3	314	99	c	67	5	418	122	z	90	3	546
52	4	20	3	136	76	L	44	2	317	100	d	68	6	423	123	{	91	8	549
53	5	21	8	139	77	M	45	4	319	101	e	69	8	429	124	|	92	1	557
54	6	22	10	147	78	N	46	3	323	102	f	70	5	437	125	}	93	8	558
55	7	23	2	157	79	O	47	8	326	103	g	71	10	442	126	~	94	9	566
56	8	24	16	159	80	P	48	6	334										

Key to the Ascii listing:

#	==	Ascii Number
C	==	Ascii Character
P	==	Position in Storage
#V	==	Number of Vectors to Draw Character
I	==	Index of Initial Vector in LOCVEC

8.28 plmnmx.F {Subroutine PLMNMX}

Given a min,max of values to be plotted, this routine will return a reasonable set of values to use for a coordinate axis.

Note	Parameter	Type	I|O|B	Description
	AIN	R*4	I	Minimum data value
	BIN	R*4	I	Maximum data value
	XST	R*4	O	Min axis value
	XFIN	R*4	O	Max axis value
	DELTAX	R*4	O	Spacing between tick marks
	NXINTS	I*d	O	Number of tick intervals
	POW	R*4	O	Factor for scaling data (i.e. Plot value = Data value * POW)
	IPOW	I*d	O	Scale factor decade
	NXPPTD	I*d	O	Number of places past the decimal in tick labels

8.29 prep19.F {Subroutine PREP19}

Routine to read an input character string which is later to be "reread" via a list-directed read on unit 19. If the input string contains all blanks, the alternate return is used for individual error handling. Otherwise, a slash (/) is appended to the string in the islash-th character position to ensure the list-directed read on logical unit 19 will be terminated. When control is returned to the calling program, unit 19 is positioned for the list-directed read.

Note	Parameter	Type	I|O|B	Description
	STRING	C*?	I	String to be input via keyboard
	ISLASH	I*d	I	Last character position permitted in the string

8.30 props.F {Subroutine PROPS}

Computes required properties of thin film gauge substrate at a given temperature.

Note	Parameter	Type	I|O|B	Description
	TK	R*4	I	Temperature (degrees K)
	N	I*4	I	Determines content of ANS
	ANS	R*4	O	ANS=k (N=1) or $k/\rho/c$ (N=2)

8.31 qfrtmp.F {Subroutine QFRTMP}

Processes request for heat flow calculations. Calls either subroutine COOK to calculate heat flow from the Cook-Felderman method where thermal properties are assumed constant; or, RAETAU which employs the Rae-Taulbee method where thermal properties may be temperature dependent and the gauge substrate is assumed infinitely thick. In both cases heat flow can be corrected (after the fact) for variation of thermal propeties with temperature using an empirical relationship.

Note	Parameter	Type	I|O|B	Description
	TRDES	REC	I	RDR for temperature in TTQ2
	TCDES	REC	I	CDR for temperature in TTQ2
	TTQ1	R*8	I	Input time array
	TTQ2	R*4	I	Input temperature array
	TTQ3	R*4	O	Output heat flow array
	IVAR	I*d	I	=0 to correct for varying thermal properties; =1 to suppress correction
	COOKMETHOD	L*d	I	T for Cook-Felderman; F for Rae_Taulbee

8.32 quickm.F {Subroutine QUICKM}

Routine to place a quick plot on any medium (presumed open) given only the essential information needed. The Tektronix set of plotting routines in the Fortran libraries provided are used for this purpose.

Note	Parameter	Type	I|O|B	Description
	XPDATA	R*4	I	Array of abscissas
	YPDATA	R*4	I	Array of ordinates
	NPTS	I*d	I	Number of points in arrays XPDATA and YPDATA
	N1D	I*d	I	First dimension of YPDATA array
	NTRACE	I*d	I	Number of traces in array YPDATA
	PTITLE	C*?	I	Plot title
	STITLE	C*?	I	Subtitle
	XLABEL	C*?	I	X-axis label
	YLABEL	C*?	I	Y-axis label
	TRCLBL	C*?	I	String containing NTRACE labels of NTRCLB bytes
	NPTITL	I*d	I	# characters in PTITLE
	NSTITL	I*d	I	# characters in STITLE
	NXLABL	I*d	I	# characters in XLABEL
	NYLABL	I*d	I	# characters in YLABEL
	NTRCLB	I*d	I	# characters in each trace label
	IFGRID	L*1	I	Logical indicator for a plot grid overlay
	IFKBOX	L*1	I	Logical indicator for a key box
	IFKSET	L*1	I	Logical indicator for a symbol key
	SAMEXV	L*1	I	Logical indicator for shared abcissa array
	INTSYM	I*d	I	Sample interval for symbol plotting
	PLTYPE	C*1	I	Single character for line (L), symbol (S) or both line & symbol (B) plot.
	NXPIXL	I*4	I	Screen width in pixels
	NYPIXL	I*4	I	Screen height in pixels
	NPIXPI	I*4	I	Pixels per inch

8.33 quickp.F {Subroutine QUICKP}

Routine to place a quick plot on any medium given only the essential information needed. The Tektronix set of plotting routines in the "tekgr" library are used for this purpose. The full extent of the plot window will be used. The user must provide the call to openplot to define the medium and the needed plot window pixel parameters.

Note	Parameter	Type	I|O|B	Description
	XPDATA	R*4	I	Array of abscissas
	YPDATA	R*4	I	Array of ordinates
	NPTS	I*4	I	Number of points in arrays XPDATA and YPDATA
	IFGRID	L*1	I	Logical indicator for a plot grid overlay
	PLTYPE	C*1	I	Single character for line (L), symbol (S) or both line & symbol (B) plot.
	NTITLE	I*4	I	Number of characters in the "TITLE" string for describing the plot. Its value must be in the range [0,80].
	TITLE	C*?	I	String of descriptive info, containing "NTITLE" characters, placed at the top of the plot.
	NXPIXL	I*4	I	Number of pixels in the x direction.
	NYPIXL	I*4	I	Number of pixels in the y direction.
	NPIXPI	I*4	I	Number of pixels per inch.

8.34 raetau.F {Subroutine RAETAU}

Performs the solution of the one-dimensional heat conduction equation with variable thermal properties using the method of Rae and Taulbee. The source file contains some internal documentation which may be helpful to the user.

Note	Parameter	Type	I|O|B	Description
	JMX	I*4	I	Number of points in data arrays
	TTQ1	R*8	I	Input time array
	TTQ2	R*4	I	Input temperature array
	TTQ3	R*4	O	Output heat flow array
	IVAR	I*d	I	0=constant; 1=variable temperature gauge

8.35 rnd.F {Subroutine RND}

Routine to generate uniform random numbers in the interval [0,1]. Usage = CALL RND(X) for X (R*4) the output except for the initial call when it is the input seed (also in [0,1]).

8.36 rotspd.F {Subroutine ROTSPD}

Calculates rotor speed given angular sample interval and the indexer time array under the assumption that time can be fit to angular displacement by a Lagrange polynomial. A five-point formula has been selected to provide the required accuracy.

Note	Parameter	Type	I|O|B	Description
	R1DES	REC	I	RDR for indexer times
	C1DES	REC	I	CDR for indexer times
	TIME	R*8	I	Array of positive time values
	RSPEED	R*4	O	Angular speed in RPM
	*	N/A	O	Alternate return for errors

8.37 splin3.F {Subroutine SPLIN3}

To evaluate coefficients for a cubic spline fit through N points. The output arrays A, B, and C contain the coefficients for the type of spline indicated in the master code selector MCODE. The functional form for the k-th interval is: f=Y(k)+u•[A(k)+u•{B(k)+u•C(k)}], where u=xx-x(k), for xx the desired abscissa value. The subroutine EVLSP3 is provided separately to compute the results of the process at user specified values of the independent variable.

Program does not check for validity of the X values in ascending order. Output will not be valid if this requirement is not met, even if no system diagnostics appear. An alternate return is provided to bypass using results only if N is zero or negative; MCODE is invalid; or, KL or KR are invalid for MCODE=5.

N values of 1 and 2 are considered valid, but the program currently provides only the "minimal solution," ignoring any specifications. For N=1, the result is a constant with A(1)= B(1)= C(1)=0.0; N=2, linear with A(*)=slope and B(*)= C(*)=0.0.

The case N=3 involves a few special cases, depending upon input specifications. If those specifications do not uniquely define the spline, a "reasonable interpretation" is made.

Note	Parameter	Type	I|O|B	Description
	N	I*d	I	Number of points (X,Y,A,B,C array lengths)
	X	R*4	I	Array of X-values (ascending order)
	Y	R*4	I	Array of Y-values
	MCODE	I*d	I	Master select code: 1=Smooth spline; 2=Natural spline; 3=Not-a-knot spline 4=Periodic spline; 5=Specs to be set with KL, KR.
1.	KL	I*d	I	Left-end constraints code: 0=None; 1=yL'=VALL; 2=yL"=VALL; 3=Not-a-knot (left); 4=yL'=yR' (partial periodicity; y').
1.	KR	I*d	I	Right-end constraints code: 0=None; 1=yR'=VALR; 2=yR"=VALR; 3=Not-a-knot (right); 4=yL"=yR" (partial periodicity; y").
1.	VALL	R*4	I	Left end value for y' (KL=1) or y" (KL=2)
1.	VALR	R*4	I	Right end value for y' (KR=1) or y" (KR=2)
	A,B,C	R*4	O	Arrays of spline fit coefficients
	*	N/A	O	Alternate return for errors

SPLIN3 Note:

1. KL,KR,VALL,VALR meaningful only if MCODE = 5, otherwise not used.

8.38 stripc.F {Subroutine STRIPC}

Routine to produce strip charts of the data in the X,Y arrays provided. The medium (i.e screen; hardcopy [8.5X11 or 11X17]) selected by the user is transparent to this routine.

Note	Parameter	Type	I|O|B	Description
	NXPIXL	I*4	I	Number of pixels in the x direction
	NYPIXL	I*4	I	Number of pixels in the y direction
	NPIXPI	I*4	B	Number of pixels per inch (If NPIXPI=0, it will be calculated assuming the proportion NXPIXL/10.24" = NYPIXL/7.8" and NPIXPI will be the average of the two ratios.)
	X(*)	R*4	I	One dimensional array of x values for all charts to be drawn
	Y(*,*)	R*d	I	Two dimensional array of y values for all charts to be drawn (Ordinates for chart i are in Y(j,i),j=1,NPTS.)
	NPTS	I*d	I	Number of points in each chart
	N1D	I*d	I	The value of the first dimension used to allocate space for the Y array in the calling routine (N1D.GE.NPTS)
	NCHART	I*d	I	The total number of strip charts to be drawn
	LYAXIS	R*4	B	The physical length (inches) of the y-axis (If the plot page cannot handle the user's request, LYAXIS is adjusted to use the maximum extent.)
	MULTIC	L*2	I	Logical variable to permit as many charts per page as size wil permit
	CLABEL	C*?	I	String containing labels for all charts in the specified order, where, ? is the string length (NCHART*NCPERL)
	NCPERL	I*d	I	Number of characters per chart label
	MEDIUM	I*d	I	1 for screen, 2 for file or plotter

8.39 tekdat.F {Block Data Subprogram TEKDAT}

Stores the initial values for all parameters in labelled common TEKUPG, defined in file "tekupg.inc", for use by the Tektronix plot routines TEKGRF and TEKPLT.

8.40 tekgrf.F {Subroutine TEKGRF}

Exactly like tekplt.F except lines or symbols may be drawn but not both; and, successive points are not drawn if their pixel coordinates are the same. Refer to tekplt.F for a description of the input formal parameters.

8.41 tekplt.F {Subroutine TEKPLT}

Routine to produce a Tektronix compatible plot of y versus x independent of the medium chosen. The descriptions below apply equally to subroutine TEKGRF in source file tekgrf.F. The only differences deal with how individual traces on a plot are handled. TEKGRF can only draw connected lines or individual symbols but not both. It also contains logic to minimize the number of plot vectors drawn and should be used in situations where large amounts of data are involved to minimize file and system space requirements.

Communication with the calling routine is accomplished using formal parameters and the labelled commons in files 'gencom.inc', 'grafix.inc' and 'tekupg.inc'. Subroutines ADJLAB, DECLAB, GRDLIN, LINAXD, LOGAXS and all Tektronix plot routines are required.

The plot medium must be defined before calling TEKPLT or TEKGRF. This is accomplished by invoking openplot (e.g. IPLOTF = OPENPLOT(NXPIXL,NYPIXL,NPIXPI,?)). The fourth argument (?) may be 'screen', 'plotter', or the name of a file. The plot medium remains open until subroutine CLOSEPLOT is called (no arguments).

Labelled common /GENCOM/ must be included in the calling routine and appear as follows:

```
      LOGICAL     KEYSET, GRIDON
      COMMON /GENCOM/ KEYSET,GRIDON,IXL,IXR,IYB,IYT,SYMSIZ,
     +                            IX0,IXN,IY0,IYN
```

Labelled common /TEKUPG/ may be included in the calling routine to override the default plot conventions defined in block data subprogram TEKDAT. The pertinent parameters are described elsewhere in this text and their default values are listed for your convenience (NCPTRL=8, SYMSPC=1, COMONX=.F., KEYBOX=.F., XLENIN=0.0, YLENIN=0.0). The user may override these values by including labelled common /TEKUPG/ in the source code as follows:

```
      INTEGER     NCPTRL, SYMSPC
      LOGICAL     COMONX, KEYBOX
      REAL        XLENIN, YLENIN
      COMMON /TEKUPG/ NCPTRL, SYMSPC, COMONX, KEYBOX, XLENIN, YLENIN
```

TEKPLT or TEKGRF may now be called as the user requires. They each try to draw and label the coordinate axes subject to the size of the plot window and its internal considerations. A clean plot page is generated by executing the statement CALL NUPAGE.

TEKGRF and TEKPLT Parameter Descriptions

Note	Parameter	Type	I|O|B	Description
1.	ICSIZE	I*d	I	Integer related to character size.
	NTRACE	I*d	I	Number of traces, n, to be drawn.
	XRAY(*)	R*4	I	Array of x data.
	YRAY(*)	R*4	I	Array of y data.
	IYAXIS(n)	I*d	I	IYAXIS(i) = y-axis # for trace i.
	LOCXY(n)	I*d	I	LOCXY(i) = 1-st point of trace i in XRAY,YRAY.
	NXYPTS(n)	I*d	I	NXYPTS(i) = # data points in trace i.
	NYAXES	I*d	I	Number of y axes to be drawn (1 or 2).
2.	NXPPTD, NYPPTD(NYAXES)	I*d	I	Integer specifying the respective x and y axis types shown below.
	NPTITL	I*d	I	Number of characters in PTITLE,
	PTITLE,STITLE, XLABEL, YLABEL(NYAXES)	C*?	I	Plot title, subtitle and x,y labels. ? is 50 for YLABEL and 80 for the rest.
3.	TRCLBL	C*?	I	String containing labels for all traces.
4.	XMIN,XMAX, DELTAX	R*4	I	Scaled min, max, spacing for the x-axis.
5.	IDSFX	I*d	I	Scale factor decade for x values.
4.	YMIN(NYAXES), YMAX(NYAXES), DELTAY(NYAXES)	R*4	I	Scaled min, max, spacing for the y-axis.
5.	IDSFY(NYAXES)	I*d	I	Scale factor decade for y values.
6.	NUMSYM(n)	I*d	I	Integer symbol # for trace n.
7.	NUMLIN(n)	I*d	I	Integer line # for trace n.

8.42 teksym.F {Subroutine TEKSYM}

Routine to use Tektronix graphics calls to draw a symbol at a specified (x,y). All parameters are input and output is a symbol drawn on the plot medium. The pen is left at (IXC,IYC) after the required symbol is drawn.

Note	Parameter	Type	I|O|B	Description
	IXC	I*d	I	X pixel coordinate of symbol center
	IYC	I*d	I	Y pixel coordinate of symbol center
	ISYMNO	I*d	I	Symbol # (see list below)
	SIZE	R*4	I	Size of symbol in inches ($0 < \text{SIZE} \leq 3$)

#	Symbol	#	Symbol
1	Arrow up	10	5-point star
2	X	11	Hexagon
3	Arrow down	12	6-point star
4	Square	13	Asterisk
5	Arrow right	14	Plus
6	Diamond	15	Y
7	Arrow left	16	Inverted Y
8	Circle	17	Both 1&3
9	Pentagon	18	Dot

List of Available Symbols in TEKSYM (Zero for no Symbol)

8.43 tofphi.F {Subroutine TOFPHI}

Finds T using a Newton-Raphson iteration to solve a polynomial expression containing $\Phi(T)$. Used in conjunction with heat flow calculation performed in subroutine RAETAU.

Note	Parameter	Type	I|O|B	Description
	PHI	R*4	I	Heat conduction parameter, $\Phi = \Phi(T)$
	TANS	R*4	B	T corresponding to value of Φ
	AKRF	R*4	I	Thermal conductivity at reference temperature
	TREF	R*4	I	Reference temperature

TEKGRF and TEKPLT Notes:

1. Graphics character sizes (pixels) based on ICSIZE:

ICSIZE	Width	Height
1	8	10
2	10	13
3	12	18
4	14	20

2. Coordinate axes types based on N_PPTD (_ = X or Y):

N_PPTD	Axis	Comment
Negative	Log	# log cycles = -N_PPTD
Zero	Linear	Integer tick labels
Positive	Linear	# places past decimal in tick labels (≤4)

3. Trace label conventions: There are NCPTRL characters per label per labelled common TEKUPG. Labels for individual traces are stored in substrings of TRCLBL. The label for trace i is in TRCLBL(m:n), for, m = 1+(i-1)•NCPTRL and n = i•NCPTRL.

4. Coordinate axes relationships: For a linear axis (MAX-MIN)/DELTA must be an integer; else, MIN and MAX are the respective min,max log cycles and DELTA does not apply. Further, DELTA < 0 implies descending tick mark labels.

5. Scale factor decade (SFD) definition: The integer power of ten used to convert MIN, MAX and DELTA to actual values. That is, actual value = scaled value • 10^{SFD}.

6. Refer to list of available symbol numbers in the section describing filename teksym.F (subroutine TEKSYM). NUMSYM(n) positive results in connected symbols while a negative value causes only symbols to be drawn.

7. List of line types (negative # indicates a double width line): 1=Solid; 2=Dotted; 3=Dot-Dash; 4=Short Dash; 5=Long Dash.

8.44 ufdate.F {Subroutine UFDATE}

Routine to get system date and time in CTIME(3) format shown below. The date will be returned in the first nine bytes of DATSTR and the time in the first eight bytes of TIMSTR. If the length of DATSTR is greater than eleven bytes, the date will be returned in the first twelve.

CTIME(3) format: Tueb̸Decb̸31b̸11:17:45b̸1991 (N.B. b̸ --> blank)

Note	Parameter	Type	I|O|B	Description
	DATSTR	C*?	O	Character string to contain the date
	TIMSTR	C*?	O	Character string to contain the time

8.45 varplt.F {Subroutine VARPLT}

Routine to permit interactive plot activity in a Sun Fortran context with the environment variable "GTYPE" set to xterm. Similar to vryplt.F but formal parameters are more extensive. Requirements: use of logical 19; proper environment (e.g. xterm, ibmpc, vt240).

Note	Parameter	Type	I|O|B	Description
	ntrace	I*d	I	# of distinct traces in x,y data (1 to 7)
1.	XPDATA(*)	R*d	I	Input array of x values
1.	YPDATA(*)	R*d	I	Input array of y values
2.	IYAXIS(j)	I*d	I	Y axis number for trace j
2.	LOCXY (j)	I*d	I	X,Y array element # at start of trace j
2.	NPLPTS(j)	I*d	I	Total number of points in trace j
	NYAXES	I*d	I	# of y axes to be drawn (1 or 2)
	NXPPTD	I*d	I	# places past decimal in x tick labels
3.	NYPPTD(i)	I*d	I	# places past decimal in y tick labels
	PTITLE	C*80	I	Plot title (up to 80 characters)
	STITLE	C*80	I	Subtitle (up to 80 characters)
	XLABEL	C*80	I	Label for X axis (up to 80 characters)
3.	YLABEL(i)	C*50	I	Label for Y axis i (up to 50 characters)
2.	TRCLBL(j)	C*8	I	Label for trace j (up to 8 characters)
	XST	R*4	I	Minimum x tick value (log cycle)
	XFIN	R*4	I	Maximum x tick value (log cycle)
	DELTAX	R*4	I	X tick spacing (minus for descending)
	SFDOFX	I*d	I	Scale factor decade for x tick values
3.	YST(i)	R*4	I	Minimum y tick value (log cycle) for axis i
3.	YFIN(i)	R*4	I	Maximum y tick value (log cycle) for axis i
3.	DELTAY(i)	R*4	I	Y axis i tick spacing (minus for descending)
3.	SFDOFY(i)	I*d	I	Scale factor decade for y axis i tick values
2.	NUMSYM(j)	I*d	I	Symbol type for trace j
2.	NUMLIN(j)	I*d	I	Line type for trace j

VARPLT Notes:

1. Index varies from 1 to sum NPLPTS(j) for j=1,NTRACE;
2. Index j varies from 1 to NTRACE;
3. Index i varies from 1 to NYAXES.

8.46 vdtopt.F {Subroutines CLRNGE, CLRROW, CLRTXT, PUTCUR, PUTEXT and PUTNUM}

Contains subroutines CLRNGE, CLRROW, CLRTXT, PUTCUR, PUTEXT and PUTNUM which cause specific things to happen on the display screen. The formal parameters to all the routines (CLRTXT has none) are described before the subroutine calling sequences.

Note	Parameter	Type	I|O|B	Description
	COLUMN	I*d	I	Screen column (normally 1 to 80)
	AVALUE	R*4	I	Numeric value to be placed on screen
	D	I*d	I	Designates format (sign) and number of places after the decimal (magnitude) for displaying AVALUE (Minus=E format; Zero=Integer; Plus=F format.)
	ENDOPT	I*d	I	End of text line options: 1 -- Clear remainder of line, don't advance cursor; 2 -- Clear and advance; 3 -- Don't clear and don't advance; 4 -- Don't clear but advance.
	NCHARS	I*d	I	Number of characters in TEXT
	ROW	I*d	I	Screen row (effective for 1 to 23 only)

	VIDOPT	I*d	I	Video options for TEXT:
				Minus -- Blinking;
				Zero -- Normal;
				Plus -- Reverse.

	W	I*d	I	Field width for AVALUE
	TEXT	C*?	I	Literal data to be placed on console.

Subroutine	Purpose (Calling Sequence)
CLRNGE	Place blanks from ROW,COLUMN thru ROW,COLUMN+NCHARS-1 (ROW,COLUMN,NCHARS)
CLRROW	Clear specified ROW from COLUMN to end of display (ROW,COLUMN)
CLRTXT	Clear entire text display (None)
PUTCUR	Place cursor at ROW,COLUMN for a screen read (ROW,COLUMN)
PUTEXT	Place NCHARS characters of TEXT at ROW,COLUMN in the video mode specified by VIDOPT and end line as per ENDOPT (VIDOPT,ROW,COLUMN,TEXT,NCHARS,ENDOPT)
PUTNUM	Starting at ROW,COLUMN place the number in AVALUE on the screen in the form designated by W and D (VIDOPT,ROW,COLUMN,AVALUE,W,D,ENDOPT)

8.47 vryplt.F {Subroutine VRYPLT}

Routine to permit interactive plot activity in a Sun Fortran context with the environment variable "GTYPE" set to xterm. Similar to varplt.F but formal parameters are not as extensive. Requirements: use of logical 19; proper environment (e.g. xterm, ibmpc, vt240).

Note	Parameter	Type	I|O|B	Description
1.	XPDATA(*)	R*d	I	Input array of x values
1.	YPDATA(*)	R*d	I	Input array of y values
	NTRACE	I*d	I	# of distinct traces in x,y data (1 to 7)
	NYAXES	I*d	I	# of y axes to be drawn (1 or 2)
2.	IYAXIS(j)	I*d	I	Y axis number for trace j
2.	LOCXY (j)	I*d	I	X,Y array element # at start of trace j
2.	NPLPTS(j)	I*d	I	Total number of points in trace j
2.	NUMLIN(j)	I*d	I	Line type for trace j
2.	NUMSYM(j)	I*d	I	Symbol type for trace j
	LOGOFX	L*d	I	Logical to plot log of x
3.	LOGOFY(i)	L*d	I	Logical to plot log of y for axis i
	PTITLE	C*80	I	Plot title (up to 80 characters)
	STITLE	C*80	I	Subtitle (up to 80 characters)
2.	TRCLBL(j)	C*8	I	Label for trace j (up to 8 characters)
	XLABEL	C*80	I	Label for X axis (up to 80 characters)
3.	YLABEL(i)	C*50	I	Label for Y axis i (up to 50 characters)

VRYPLT Notes:

1. Index varies from 1 to sum NPLPTS(j) for j=1,NTRACE;
2. Index j varies from 1 to NTRACE;
3. Index i varies from 1 to NYAXES.

Appendix A Run Descriptor Record

Presented below are the elements comprising the RUNDES structure defined in file ′dstruc.inc′ and referenced by a Fortran "INCLUDE" statement in the DAS source files. If the statement "RECORD/RUNDES/R" is present, then the RDR element named X is uniquely defined by "R.X" throughout the source routine.

Type	Parameter	Description
I*4	MAGIC	Specific integer to identify a DAS atarr data file
I*4	RUN	Run/Experiment sequence number
I*4	NCHAN	Number of channels recorded this run
I*4	VERSION	DAS version number
I*4	CTIME	RCDF creation time in system format
I*4	RTIME	Run time (empty in RCDF)
I*4	FAC_TURB	**Facility Turbine Parameters Data Toggle**
I*4	NSTAGES	Number of turbine stages
I*4	NBLADES[2]	Number of blades on each turbine stage
I*4	NVANES[2]	Number of vanes on each turbine stage
R*4	NGVA	Cross sectional area of nozzle guide vane (NGV) inlet
R*4	CVA	Choke valve orifice Area
R*4	BLBA	Boundary layer bleed area
R*4	KCV	Choke valve discharge coefficient
R*4	KBL	Boundary layer bleed discharge coefficient
R*4	DTIP	Diameter of rotor
R*4	VSUPPLY	Volume of supply tank
R*4	ANGTO1	Anglular offset of blade 1 relative to indexer (degrees)
I*4	FILL_DAT	**Fill Parameters Data Toggle**
R*4	MOLEF[2]	Mole fractions of the test gas components
R*4	MW	Molecular weight of test gas (kg/kg-mole)
R*4	ACP[3]	Test gas specific heat coefficients (i=1..3)
R*4	AGAM[3]	Test gas specific heat ratio coefficients (i=1..3)
R*4	APR[3]	Test gas Prandtl number coefficients (i=1..3)
R*4	AMU[3]	Test gas dynamic viscosity coefficients (i=1..3)
R*4	GCONS	Gas constant (Ru/MW)
R*4	PPGAS1	Partial pressure of test gas component #1
R*4	TGAS1	Temperature of test gas component #1
R*4	PFINAL	Final fill pressure

R*4	TFINAL	Final fill temperature

Appendix A Run Descriptor Record (Continued)

Type	Parameter	Description
I*4	DES_COND	**Design Conditions Data Toggle**
R*4	TSTAG	Upstream design stagnation temperature
R*4	PSTAG	Upstream design stagnation pressure (kPa)
R*4	WDTOTAL	Total design weight flow of test gas (kg/sec) from user
R*4	WDTURB	Design weight flow of test gas through turbine (kg/sec)
R*4	PRANDTL	Design Prandtl number of test gas
I*4	STANDARD	**Standards Data Toggle**
R*4	TSUPPLY0	Supply tank temperature
R*4	PSUPPLY0	Supply tank pressure
R*4	TINIT	Test section temperature at t0
R*4	PINIT	Test section pressure at t0
R*4	BEGTIM	Beginning time of interval used to determine standards
R*4	ENDTIM	Ending time of interval used to determine standards
I*4	ERRORS	**Error Analysis Data Toggle**
R*4	E_TSUPP0[3]	Supply tank temperature uncertainty
R*4	E_PSUPP0[3]	Supply tank pressure uncertainty
R*4	E_TINIT[3]	Test section temperature uncertainty
R*4	E_PINIT[3]	Test section pressure uncertainty
C*80	RDESC	Description of experiment
C*80	TDESC	Description of turbine
C*10	GASTYP[2]	Labels describing the test gas components
C*576	RUNPAD	**Padding to 1024 bytes**

Appendix B Channel Descriptor Record

Presented below are the elements comprising the CHDES structure defined in file ´dstruc.inc´ and referenced by a Fortran "INCLUDE" statement in the DAS source files. If the statement "RECORD/CHDES/C" is present, then the CDR element named X is uniquely defined by "C.X" throughout the source routine.

Type	Parameter	Description
I*4	ACTIVE	Channel status: 0=Inactive; 1=Active.
I*4	CTYPE	Data type: 0=>raw data; 1=>from CTYPE=0; 2=>from CTYPE=1; 3=>from CTYPE=1&2 or other source.
I*4	SEQ	Sequence number of this channel
I*4	TYPE	Sensor calibration type
R*4	SCALE	Calibration constant (e.g. Volts / Pa)
R*4	C_LAB[7]	Coefficients for laboratory voltage calibration (i=1..7)
R*4	C_PRE[7]	Coefficients for pre-test voltage calibration (i=1..7)
R*4	C_POST[7]	Coefficients for post-test voltage calibration (i=1..7)
R*4	C_USED[7]	Coefficients for calibration used (i=1..7)
R*4	RCALIB	Resistance (Ω) of the gauge at room temperature
R*4	RLINE	Resistance (Ω) of the lines from the amps to the gauge
R*4	RS	Resistance (Ω) of the large series resistor in gauge
R*4	RPRER	Resistance (Ω) of the gauge prior to a test
R*4	TPRER	Temperature of the gauge prior to a test
R*4	IZERO	Transducer excitation: plus=current (mA); minus=voltage (V).
R*4	GAIN	Amplifier voltage gain
I*4	DELTAV	0=Absolute; 1=Relative; 2=Calibrated voltage measurement
I*4	AOFFSET	Amplifier register offset value
I*4	VOFFSET	Desired offset from midrange of A/D (±100% of FSVV/2)
I*4	BOFFSET	A/D representation of VOFFSET
I*4	RECSET	Recorded A/D offset
I*4	ADMODE	Recording mode of A/D: 0=2s complement; 1=binary offset.
R*4	AALIAS	Antialiasing filter Cutoff Frequency (Hz)
I*4	AFTYPE	Amplifier filter type: 0=none; 1=Bessel 4-pole; 2=Bessel 8-pole; 3=Butterworth 4-pole; 4=Butterworth 8-pole.

Appendix B Channel Descriptor Record (Continued)

Type	Parameter	Description
I*4	DFTYPE	Digital filter type: 0=none; 1=low pass; 2=high pass; 3=band pass; 4=band stop.
R*4	LOF	Low frequency end of band (DFTYPE=3) or notch (DFTYPE=4) in Hz
R*4	HIF	High frequency end of band (DFTYPE=3) or notch (DFTYPE=4) in Hz
R*4	FSVV	Full scale A/D voltage range
I*4	BITRES	Number of bits of A/D resolution
I*4	LSR	Low sample rate (Hz) or indexer pulses per revolution
I*4	HSR	High sample rate (Hz) or indexer pulses per revolution
I*4	NSAMP1	Number of startup transient samples to acquire
I*4	NSAMP2	Number of high speed samples to acquire
I*4	NSAMP3	Number of additional low speed samples to acquire
I*4	FIRST_AT	Sample number of first sample of the first full revolution
I*4	LAST_REV	Last full revolution in the channel
I*4	CHASSIS	A/D chassis identification number
I*4	SLOT	A/D slot number in chassis
I*4	CHANNEL	A/D channel number on device
I*4	ACHASSIS	Amplifier chassis identification number
I*4	ASLOT	Amplifier slot number in chassis
I*4	ACHANNEL	Amplifier channel number on device (device dependent)
I*4	GLOCTYP	Gauge location (e.g. blade, vane, rake)
I*4	GLOCNO	Gauge location number (e.g. 1 for blade1, vane1, rake1)
I*4	GSIDE	Gauge location - side (e.g. low or high pressure side)
R*4	GXPOS	X-coordinate of gauge
R*4	GYPOS	Y-coordinate of gauge
R*4	GZPOS	Z-coordinate of gauge
R*4	BLSTART	Start time for base line averaging (secs)
R*4	BLEND	Ending time for base line averaging (secs)
R*4	TZERO	Offset to be added to time samples (secs)

Appendix B Channel Descriptor Record (Continued)

Type	Parameter	Description
I*4	UNISYS	Units System (0=SI, 1=ENGLISH) (0 for CTYPE= 0&1)
I*4	UNITYP	DAS physical units number type (e.g. 0=none, 1=Time, 2=Power, etc.)
I*4	ALGNUM	DAS algorithm number from which data was derived (0 for CTYPE= 0&3)
I*4	X_TYPE	If LSS=X, X_TYPE is integer for x-axis descriptor (e.g. 1 --> Blade Passing)
R*4	DELT_X	Applies only if LSS=X: X(i) appended if zero; else, X(i)=X_ZERO+(i-1)*DELT_X
R*4	X_ZERO	Applies only if LSS=X and DELT_X nonzero
C*15	UNILAB	Physical units label (e.g. "BTU" if UNISYS=1 & UNITYP=3)
C*80	STORY	User supplied descriptive information pertaining to data
C*1	VALID	Is the record for this channel complete (0=no, 1=yes)
C*8	LABEL	Channel Identifier (strict DAS convention for CTYPE= 0&1)
C*12	SERIAL	Sensor calibration serial number
C*1	LSS	Low sample rate source (C=clock, I=indexer)
C*1	HSS	High sample rate source (C=clock, I=indexer)
C*8	DTYPE	Digitizer (A/D) device type name
C*8	ATYPE	Signal conditioner (amplifier) device type name
C*8	GLOC	Gauge location (e.g. blade1, vane2, rake2)
C*61	X_DESC	X-axis descriptor if LSS=X (e.g. Blade Passing, Degrees RDZ, etc.)
C*15	E_UNIT	Corresponding English unit of measure for Y axis data.
C*482	CHNPAD	**Padding to 1024 bytes**

Appendix C Alphabetic Routine Listing

Name	Purpose	Filename
ADJLAB	Adjusts labels for centering in available plot space	adjlab.F
AXDATA	Permits specification of axis information for plotting	axdata.F
BETA	Computes *ρck* of thin film gauge substrate	beta.F
CLRNGE	Blanks a range of columns in a given row of screen text	vdtopt.F
CLRROW	Clears screen text row starting with a given column	vdtopt.F
CLRTXT	Clears the entire text display	vdtopt.F
CONVRT	Converts data from SI to English or vice versa	convrt.F
COOK	Computes heat flow using Cook-Felderman method	cook.F
DECLAB	Places label of form 10^1 at given pixel (x,y)	declab.F
DEFINE	Initializes (defines) major portion of DAS resources	define.F
DEFLIM	Defines all coordinate axis parameters given Min,Max[,Spacing]	deflim.F
DERIVN	Lagrangian n-point derivative of equally spaced function	derivn.F
DIRECT	Processes requests from DRP user interface	direct.F
EVLSP3	Evaluates spline fit at specified values of independent variable	evlsp3.F
GALPHA	Plots character string given any angle and pixel (x,y)	galpha.F
GETRAY	Allocates space for the array associated with the given pointer	getray.F
GETRES	Opens DAS files and loads RDR, CDR and channel data array	getres.F
GET1C	Returns Ascii number of key struck by user	get1c.c
GLABEL	Plots character string given 90° multiple and pixel (x,y)	glabel.F
GRDLIN	Plots grid overlay for the coordinate direction specified	grdlin.F
INITSIG	Permits Ctrl-C interrupt handling in applications	interrupt.c
I2SORT	Numerically sorts I*2 array of values	numsrt.F
I4SORT	Numerically sorts I*4 array of values	numsrt.F
LINAXD	Draws and labels a linear coordinate axis (X or Y)	linaxd.F
LNBLNK	Returns position of last nonblank character in input string	lnblnk.F
LOGAXS	Draws and labels a log coordinate axis (X or Y)	logaxs.F
MULPLT	Draws multiple plots and performs weighted channel averages	mulplt.F

Name	Purpose	Filename
NDPLCS	Determines optimum "d" to list given number in Fw.d format	ndplcs.F
NUMSIN	Translates input form "N1,N2-N3,N4" for random order processing	numsin.F
NUM2CH	Converts an integer to its character string equivalent	num2ch.F
PFONTS	Stores vectors for drawing most Ascii characters	pfonts.F
PLMNMX	Returns "reasonable" coordinate axis values from given Min,Max	plmnmx.F
PREP19	Allows "reread" of screen inputs on logical unit 19	prep19.F
PROPS	Computes k or $k/\rho/c$ of thin film gauge substrate	props.F
PUTCUR	Put cursor at specified text (x,y)	vdtopt.F
PUTEXT	Place given character string at text (x,y)	vdtopt.F
PUTNUM	Put floating point # at text (x,y) in Iw, Fw.d or Ew.d format	vdtopt.F
QFRTMP	Processes heat flow calculation requests	qfrtmp.F
QUICKM	Draws a multi-trace plot with minimum user effort	quickm.F
QUICKP	Plots a single curve needing only essential information	quickp.F
RAETAU	Computes heat flow using Rae-Taulbee method	raetau.F
R4SORT	Numerically sorts R*4 array of values	numsrt.F
R8SORT	Numerically sorts R*8 array of values	numsrt.F
RND	Generates uniform random numbers in the interval [0,1]	rnd.F
ROTSPD	Computes rotor speed from the Indexer time history	rotspd.F
SPLIN3	Evaluates coefficients for a cubic spline fit through N points	splin3.F
STRIPC	Produces strip charts of data in X,Y arrays provided	stripc.F
TEKDAT	Initializes parameters for TEKGRF & TEKPLT routines	tekdat.F
TEKGRF	Produces Tektronix compatible plot of y vs x (limited)	tekgrf.F
TEKPLT	Produces Tektronix compatible plot of y vs x (unlimited)	tekplt.F
TEKSYM	Uses Tektronix calls to draw required symbol at pixel (x,y)	teksym.F
TOFPHI	Inverts Φ(T) polynomial to obtain T	tofphi.F
UFDATE	Derives strings with date and time from system CTIME(3) format	ufdate.F
VARPLT	Permit interactive plotting in Fortran context (28 parameters)	varplt.F
VRYPLT	Permit interactive plotting in Fortran context (16 parameters)	vryplt.F

REFERENCES

1. Moselle,J.R., et al: Proposed Data Acquisition System fo WPAFB Advanced Turbine Aerothermal Research Rig (ATARR). Critical Design Review Document, DEC,1989.

2. Kernighan, Brian W. and Pike, Rob, "The UNIX Programming Environment". Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1984.

Facility Model Parameters

Program *facility* has two basic forms of supplying input data. The first consists of a file containing facility configuration parameters. There are seven distinct groups of data which are entered using the FORTRAN namelist convention.

NAMELIST	CONTENTS
FD1	Facility Volumes
FD2	Flow Path Orifice
VD1	Valve Geometry Variables
VD2	Impact Spring Data
VD3	Actuator Cylinder Geometry
VD4	Actuator Gas Supply Geometry
BRK	Brake Constants

Each namelist has one or more parameter names which are used to enter the associated input data. The namelist parameters are presented for the user's convenience:

FD1	Facility Volume Data (Cubic Feet)
V0	SUPPLY TANK
V1	TURBINE INLET
V2	TURBINE EXHAUST
V3	DSV INLET
V4	DUMP TANK

FD2	Facility Flow Path Orifice (Square Inches)
ADSO	EXIT PIPES TO DUMP, A34: 2-20"DIA, CD=.62

VD1	Valve Geometry Variables
ISS	INTEGER INDICATOR [0=MIT COWCATCHER; 1=ATARR SLIDING SLEEVE]
RO	VALVE ANNULUS, OUTER RADIUS (INCHES)
RI	VALVE ANNULUS, INNER RADIUS (INCHES)
RM	MAIN SEAL RADIUS (INCHES)
XM	MAIN SEAL BREAKAWAY (INCHES)
RS	BACK SEAL RADIUS (INCHES)
XS	BACK SEAL BREAKAWAY (INCHES)
THETA	ENTRANCE ANGLE FROM VERTICAL (DEGREES)
MASS	TOTAL MOVING MASS (KG)

Facility Model Parameters

VD2	**Impact Spring Data**
XKSC	ZERO FORCE POSITION, MAIN VALVE CLOSING SPRING (INCHES)
KSC	SPRING CONSTANT [CLOSING SPRING] (LBF/IN)
XKSO	ZERO FORCE POSITION [VALVE OPENING SPRING] (INCHES)
KSO	SPRING CONSTANT [OPENING SPRING] (LBF/IN)
CSF	COEFFICIENT OF STATIC FRICTION
	COEFFICIENTS OF DYNAMIC FRICTION FOR SLIDER RANGES SHOWN:
CD1	0 TO X1
CD2	X1 TO X2
CD3	X2 TO X3
CD4	X3 TO END
	SLIDER POSITIONS FOR DYNAMIC FRICTION COEFFICIENTS:
X1	LOCATION 1 (INCHES)
X2	LOCATION 2 (INCHES)
X3	LOCATION 3 (INCHES)

VD3	**Actuator Cylinder Geometry**
DB	BORE DIAMETER (INCHES)
DR	ROD DIAMETER (INCHES)
NRODS	NUMBER OF RODS [1 FOR MIT; 2 FOR ATARR]
SMAX	MAXIMUM ACTUATOR STROKE (INCHES)
SP	PISTON THICKNESS (INCHES)
S0	STROKE OFFSET, S @ X=0 (INCHES)
S1	OFFSET TO START OF TRANSFER PORT (INCHES)
S2	END OF TRANSFER PORT (INCHES)
AV67	TRANSFER PORT AREA (IN**2)
AF67	ACTUATOR FIXED BLEED AREA (IN**2)
ALEAK	LEAKAGE AROUND PISTON (IN**2)
SSW	SWITCH POSITION (INCHES)

VD4	**Actuator Gas Supply Geometry**
V5	OPENING GAS SUPPLY VOLUME (M**3)
VF6	OPENING LINE VOLUME (IN**3)
VF7	CLOSING LINE VOLUME (IN**3)
V8	CLOSING GAS SUPPLY VOLUME (M**3)
IVNT	VENT INDICATOR [0=DUMP; 1=ROOM]
AO56	OPENING GAS SUPPLY ORIFICE (IN**2)
AO7V	CLOSING GAS SUPPLY ORIFICE (IN**2)
AC87	CYLINDER CLOSING END VENT ORIFICE (IN**2)
AC6V	CYLINDER OPENING END VENT ORIFICE (IN**2)
VDEAD	DEAD VOLUME IN CYLINDER HEAD (IN**3)

BRK	**Brake Constants**
KB	BRAKE CONSTANT (W-S**2 / TESLA**2)
NB	BRAKE SPEED @ MAXIMUM TORQUE (RPM)
MDRUM	BRAKE DRUM MASS (KG)
CPD	BRAKE DRUM SPECIFIC HEAT (J/KG-K)
TAUB	BRAKE TIME CONSTANT (SECONDS)

Guidelines for entering each namelist name and parameters:

First line contains a dollar sign ($) in column 2 and the namelist name starting in column 3. For each namelist parameter, open a new line and enter its name anywhere beyond column 1 followed by an equal sign (=), the value to be assigned the parameter, and a terminating comma (,). For example, the line " V3 = 250," appearing in the FD1 namelist input stream would specify a DSV inlet volume of 250 cubic feet. To terminate input for a namelist, its last line should contain the four characters "$END" starting in column 2.

The second form of input to program *facility* consists of the responses to screen prompts for input data. These responses may be entered interactively or placed in an ASCII file to be read by the program. Since the data are order dependent, there is no recovery from input errors when a response file is used. Even considering this limitation, it is preferable to use this form of input because it provides an automatic record and a convenient platform for changes. For the most part, a response file consists of lines of single numbers whose meanings are explained in the outline presented below.

Outline of Response File Entries for the Facility Model Program

Parameter	**Usage in Facility Model Program**
IFRCDF	MODE INDICATOR [0=PREDICTIVE; 1=GENERATE RCDF VALUES]
IPNTOUT	GENERATE FULL PRINTOUT INDICATOR [0=NO; 1=YES]
ISD	START-UP DYNAMICS INDICATOR [0=NO; 1=YES]
ITURB	TURBINE# [0=INPUT LIST; 3=TURBINE 3; 7=MIT ACE; 8=FULL ACE]

```
If ITURB=0, insert the following turbine parameters (one per line).
PIDP         DESIRED PRESSURE RATIO
ETADP        ADIABATIC EFFICIENCY
GDP          SPECIFIC HEAT RATIO
AT           EFFECIVE CHOKED AREA (SQUARE INCHES)
IR           ROTOR MOMENT OF INERTIA (KG M**2)
DTIP         TIP DIAMETER (INCHES)
TGTWDP       GAS TO WALL TEMPERATURE RATIO
REDP         DESIGN REYNOLDS NUMBER
```

NS0	DESIGN OPERATING SPEED [RPM]
IBEX	EDDY-BRAKE OPTION [1-COMPUTED BY MODEL, 2-PREPROGRAMMED]

```
If IBEX=2, insert the following parameters (one per line).
B0           INITIAL EXCITATION LEVEL
DB_DT        SECOND ORDER EXCITATION CONSTANT
D2B_DT2      THIRD ORDER EXCITATION CONSTANT
```

P0PSI	SUPPLY TANK PRESSURE (PSIA)
T0IN	SUPPLY TANK TEMPERATURE (+K I -F)
P4PSI	DUMP TANK PRESSURE (PSIA)
T4IN	DUMP TANK TEMPERATURE (+K I -F)
GD	DESIRED TEST GAS GAMMA

```
If IFRCDF=1, input a single line with the following four values:
PFILL(1)      PARTIAL PRESSURE OF GAS #1 (PSIA)
TFILLK(1)     TEMPERATURE OF GAS #1 (K)
PFILL(2)      FINAL FILL PRESSURE (PSIA)
TFILLK(2)     FINAL FILL TEMPERATURE (K)
```

Outline of Response File Entries for the Facility Model Program (continued)

Parameter	**Usage in Facility Model Program**
NTGCMP	NUMBER OF TEST GAS COMPONENTS [NTGCMP = 1 OR 2]
ITGCMP	TEST GAS INDICATOR(S) [1=AIR; 2=ARGON; 3=FREON12; 4=N2; 5=CO2]
P5PSI	OPENING RESERVOIR PRESSURE (PSIA)
P8PSI	CLOSING RESERVOIR PRESSURE (PSIA)
PVRPSI	VENT RESERVOIR PRESSURE (PSIA)
PVA0PSI	INITIAL ACTUATOR PRESSURE (PSIA)
NAGCMP	NUMBER OF ACTUATOR GAS COMPONENTS [USUALLY 1]
IAGCMP	ACTUATOR GAS INDICATOR [1=AIR; 2=ARGON; 3=FREON12; 4=N2; 5=CO2]
TSIM	SIMULATION RUN TIME (SECS)
NPMS	NUMBER OF STEPS PER MILLISECOND
TOPEN	VALVE OPENING TIME (SECS)
TCLOSE	VALVE CLOSING TIME (SECS)
TON	BRAKE TURN-ON TIME (SECS)
TOFF	BRAKE TURN-OFF TIME (SECS)

Hopefully, this document provides an accurate summary of the input data required to successfully use program *facility*. It is important to note that when the program is run in the mode to generate RCDF values, it is essential that a file of the form "run*.rcdf" (* = run number) be present in the current working directory. No such restriction is imposed when the program is run in predictive mode.

Changing RDR & CDR in DAS Files

DAS data files contain all the information necessary to access and represent their contents. Data always reside on a direct access, binary (unformatted) file, with fixed length records (1024 bytes). The first two records, the Run Descriptor Record (RDR) and Channel Descriptor Record (CDR), contain information arranged in a specific order. In the course of the ATARR's continued usage the form of either of these records may require changes to accommodate the evolution of the facility. Whenever such a need arises, all DAS processes must be updated to operate only on files containing the current level of revision (version) of the RDR and CDR. A complete listing of all run and channel parameters is provided at the end of this document.

The Test Setup Process (TSP) is the preferred tool for creating a Run Channel Descriptor File (RCDF). The RCDF incorporates a single RDR and the CDR's of all channels to be considered by the processes governing the creation/acquisition of DAS raw data files (e.g. FCP and DAP). If revisions to the RDR and CDR are needed, the RCDF must be converted in addition to the individual DAS data files. This requirement preserves the functionality of some of the DAS utility programs that use the RCDF as a resource for locating channels to be processed. The term *channel* refers to a DAS file containing sensor information at any level of data reduction.

When it is determined that the form of the RDR or the CDR must be altered the ATARR system manager will provide a conversion utility program named *newcdr*_**m***to***n** to convert DAS version **m** files to version **n**. During the shakedown of the orginal version of the DAS software it was determined that several parameters should be added to the CDR. Accordingly, the need to include the pertinent DAS version number among the RDR parameters was established.

The DAS software is dependent on two files that reside in */usr/atarr/include* to define the individual run and channel related structures. Using the same syntax to "include" files, "C" language routines use *runchan.h* and Fortran uses *dstruc.inc*. The resulting impact on software performance resulting from any change(s) in these files must be fully assessed before final implementation. In particular, files *listdas.F*, *rdr2das.F* and *viewdas.F* in */usr/atarr/src/fortran* require considerable attention to successfully incorporate any modification(s) made to the RDR or CDR.

Files *runchan.h* and *dstruc.inc* are not changed by direct means. They are constructed by *makerunchan* a utility program written in perl script and residing in */usr/atarr/src/runchan*. It processes the changes in file *runchan.dat* to produce the required syntactically correct files. *runchan.dat*, itself, is an ascii text file with each parameter represented by a single line of tab separated entries. This methodology provides a convenient context for making changes and reduces the possibility of user error with regard to building the run and channel structures used in the DAS software suite of programs.

A good example of the steps involved in changing the form of the RDR or CDR is provided by the conversion from the original version (i.e. 1) to version 2. The change to the RDR was accomplished simply by adding a four byte integer variable for the DAS version number as the fourth parameter.

The CDR was extended to include the English units equivalent of the SI units label as well as parameters which define DAS files which are not time-based. The requisite software now recognizes "X" as a legitimate sample source in addition to the previous clock ("C") and indexer ("I") sources. Three four byte parameters were inserted after the DAS algorithm number that pertain to "X" files. The first is X_TYPE, an integer to select the x-axis descriptor. It is followed by two floating point values: DELT_X, which, if nonzero, defines the fixed abscissa spacing; and, X_ZERO, the abscissa value at the initial point. The final two additions were character strings inserted just before the CDR "padding": X_DESC, 61 bytes to accommodate the selection denoted by X_TYPE (see above); and, E_UNIT, the 15 byte English units equivalent to the SI units label in the existing UNILAB parameter.

The program *newcdr_1to2* was developed to change existing DAS data files remaining from version 1. Its usage syntax is readily obtained by entering its name from the ATARR console or its supported terminals. Future conversion programs, except for the name, should operate in the same way. The mechanics of the underlying programming for the present example were quite simple. An existing record of information was input according to the version 1 arrangement and output in the version 2 format. With the exception of the version number in the RDR, all new numerical parameters were set to zero. String variables are blank filled to contain the appropriate number of characters. It is intended that all future conversions be handled in the same manner.

In most instances an ATARR user will not be aware that a DAS file is unacceptable to a process until an attempt to access it is made. A clear version incompatibility message is usually presented whenever this occurs. Using the assumption that such a warning was issued while executing a future version of the DAP, we can proceed with the likely solution.

The user may exit the DAP or transfer to another window. He then must change the current working directory (cwd) to where the file(s) to be converted reside. Let us further assume that all resident version 1 raw and primary DAS files are to be converted to version 3. First, execute the command "newcdr_1to2ƀ*.0ƀ*.1" (interpret ƀ as one or more blanks). If no problems are encountered, all pertinent version 1 files will be permanently altered. Next, execute the command "newcdr_2to3ƀ*.0ƀ*.1". At the completion of the two steps all raw and primary files in the cwd could be successfully accessed by any DAS version 3 program.

The remainder of this document lists the run and channel parameters contained in the current program structures.

Parameters in the Structure for the Run Descriptor Record

Type	Parameter	Description
I*4	MAGIC	Specific integer to identify a DAS atarr data file
I*4	RUN	Run/Experiment sequence number
I*4	NCHAN	Number of channels recorded this run
I*4	VERSION	DAS version number
I*4	CTIME	RCDF creation time in system format
I*4	RTIME	Run time (empty in RCDF)
I*4	FAC_TURB	**Facility Turbine Parameters Data Toggle**
I*4	NSTAGES	Number of turbine stages
I*4	NBLADES[2]	Number of blades on each turbine stage
I*4	NVANES[2]	Number of vanes on each turbine stage
R*4	NGVA	Cross sectional area of nozzle guide vane (NGV) inlet
R*4	CVA	Choke valve orifice Area
R*4	BLBA	Boundary layer bleed area
R*4	KCV	Choke valve discharge coefficient
R*4	KBL	Boundary layer bleed discharge coefficient
R*4	DTIP	Diameter of rotor
R*4	VSUPPLY	Volume of supply tank
R*4	ANGTO1	Anglular offset of blade 1 relative to indexer (degrees)
I*4	FILL_DAT	**Fill Parameters Data Toggle**
R*4	MOLEF[2]	Mole fractions of the test gas components
R*4	MW	Molecular weight of test gas (kg/kg-mole)
R*4	ACP[3]	Test gas specific heat coefficients (i=1..3)
R*4	AGAM[3]	Test gas specific heat ratio coefficients (i=1..3)
R*4	APR[3]	Test gas Prandtl number coefficients (i=1..3)
R*4	AMU[3]	Test gas dynamic viscosity coefficients (i=1..3)
R*4	GCONS	Gas constant (Ru/MW)
R*4	PPGAS1	Partial pressure of test gas component #1 (kPa)
R*4	TGAS1	Temperature of test gas component #1 (°K)
R*4	PFINAL	Final fill pressure (kPa)
R*4	TFINAL	Final fill temperature (°K)

Parameters in the Structure for the Run Descriptor Record (Continued)

Type	**Parameter**	**Description**
I*4	DES_COND	**Design Conditions Data Toggle**
R*4	TSTAG	Upstream design stagnation temperature (°K)
R*4	PSTAG	Upstream design stagnation pressure (kPa)
R*4	WDTOTAL	Total design weight flow of test gas (kg/sec) from user
R*4	WDTURB	Design weight flow of test gas through turbine (kg/sec)
R*4	PRANDTL	Design Prandtl number of test gas
I*4	STANDARD	**Standards Data Toggle**
R*4	TSUPPLY0	Supply tank temperature (°K)
R*4	PSUPPLY0	Supply tank pressure (kPa)
R*4	TINIT	Test section temperature at t_0 (°K)
R*4	PINIT	Test section pressure at t_0 (kPa)
R*4	BEGTIM	Beginning time of interval used to determine standards (secs)
R*4	ENDTIM	Ending time of interval used to determine standards (secs)
I*4	ERRORS	**Error Analysis Data Toggle** (uncertainties to be determined)
R*4	E_TSUPP0[3]	Supply tank temperature uncertainty
R*4	E_PSUPP0[3]	Supply tank pressure uncertainty
R*4	E_TINIT[3]	Test section temperature uncertainty
R*4	E_PINIT[3]	Test section pressure uncertainty
C*80	RDESC	Description of experiment
C*80	TDESC	Description of turbine
C*10	GASTYP[2]	Labels describing the test gas components
C*576	RUNPAD	**Padding to 1024 bytes**

Parameters in the Structure for the Channel Descriptor Record

Type	Parameter	Description
I*4	ACTIVE	Channel status: 0=Inactive; 1=Active.
I*4	CTYPE	Data type: 0→raw data; 1→from CTYPE=0; 2→from CTYPE=1; 3→from CTYPE=1&2 or other source.
I*4	SEQ	Sequence number of this channel
I*4	TYPE	Sensor calibration type
R*4	SCALE	Calibration constant (e.g. Volts / Pa)
R*4	C_LAB[7]	Coefficients for laboratory voltage calibration (i=1..7)
R*4	C_PRE[7]	Coefficients for pre-test voltage calibration (i=1..7)
R*4	C_POST[7]	Coefficients for post-test voltage calibration (i=1..7)
R*4	C_USED[7]	Coefficients for calibration used (i=1..7)
R*4	RCALIB	Resistance (Ω) of the gauge at room temperature
R*4	RLINE	Resistance (Ω) of the lines from the amps to the gauge
R*4	RS	Resistance (Ω) of the large series resistor in gauge
R*4	RPRER	Resistance (Ω) of the gauge prior to a test
R*4	TPRER	Temperature of the gauge prior to a test
R*4	IZERO	Transducer excitation: plus=current (mA); minus=voltage (V).
R*4	GAIN	Amplifier voltage gain
I*4	DELTAV	0=Absolute; 1=Relative; 2=Calibrated voltage measurement
I*4	AOFFSET	Amplifier register offset value
I*4	VOFFSET	Desired offset from midrange of A/D (±100% of FSVV/2)
I*4	BOFFSET	A/D representation of VOFFSET
I*4	RECSET	Recorded A/D offset
I*4	ADMODE	Recording mode of A/D: 0=2s complement; 1=binary offset.
R*4	AALIAS	Antialiasing filter Cutoff Frequency (Hz)
I*4	AFTYPE	Amplifier filter type: 0=none; 1=Bessel 4-pole; 2=Bessel 8-pole; 3=Butterworth 4-pole; 4=Butterworth 8-pole.
I*4	DFTYPE	Digital filter type: 0=none; 1=low pass; 2=high pass; 3=band pass; 4=band stop.
R*4	LOF	Low frequency (Hz) end of band (DFTYPE=3) or notch (DFTYPE=4)
R*4	HIF	High frequency (Hz) end of band (DFTYPE=3) or notch (DFTYPE=4)
R*4	FSVV	Full scale A/D voltage range
I*4	BITRES	Number of bits of A/D resolution (normally 12)

Parameters in the Structure for the Channel Descriptor Record (Continued)

Type	Parameter	Description
I*4	LSR	Low sample rate (Hz) or indexer pulses per revolution
I*4	HSR	High sample rate (Hz) or indexer pulses per revolution
I*4	NSAMP1	Number of startup transient samples to acquire
I*4	NSAMP2	Number of high speed samples to acquire
I*4	NSAMP3	Number of additional low speed samples to acquire
I*4	FIRST_AT	Sample number of first sample of the first full revolution
I*4	LAST_REV	Last full revolution in the channel
I*4	CHASSIS	A/D chassis identification number
I*4	SLOT	A/D slot number in chassis
I*4	CHANNEL	A/D channel number on device
I*4	ACHASSIS	Amplifier chassis identification number
I*4	ASLOT	Amplifier slot number in chassis
I*4	ACHANNEL	Amplifier channel number on device (device dependent)
I*4	GLOCTYP	Gauge location (e.g. blade, vane, rake)
I*4	GLOCNO	Gauge location number (e.g. 1 for blade1, vane1, rake1)
I*4	GSIDE	Gauge location - side (e.g. low or high pressure side)
R*4	GXPOS	X-coordinate of gauge
R*4	GYPOS	Y-coordinate of gauge
R*4	GZPOS	Z-coordinate of gauge
R*4	BLSTART	Start time for base line averaging (secs)
R*4	BLEND	Ending time for base line averaging (secs)
R*4	TZERO	Offset to be added to time samples (secs)
I*4	UNISYS	Units System (0=SI, 1=ENGLISH) (default = 0)
I*4	UNITYP	DAS physical units number type (e.g. 0=none, 1=Time, 2=Power, etc.)
I*4	ALGNUM	DAS algorithm number from which data was derived (0 for CTYPE= 0&3)
I*4	X_TYPE	If LSS=X, X_TYPE is integer for x-axis descriptor (e.g. 1 → Blade Passing)
R*4	DELT_X	Only if LSS=X: X(i) appended if zero; else, X(i)=X_ZERO+(i-1)*DELT_X
R*4	X_ZERO	Only if LSS=X and DELT_X nonzero
C*15	UNILAB	Physical units label (e.g. "Joules" for UNITYP=3)
C*80	STORY	User supplied descriptive information pertaining to data
C*1	VALID	Is the record for this channel complete (0=no, 1=yes)
C*8	LABEL	Channel Identifier (strict DAS convention for CTYPE= 0&1)
C*12	SERIAL	Sensor calibration serial number

Parameters in the Structure for the Channel Descriptor Record (Continued)

Type	Parameter	Description
C*1	LSS	Low sample rate source (C=Clock, I=Indexer)
C*1	HSS	High sample rate source (C=Clock, I=Indexer)
C*8	DTYPE	Digitizer (A/D) device type name
C*8	ATYPE	Signal conditioner (amplifier) device type name
C*8	GLOC	Gauge location (e.g. blade1, vane2, rake2)
C*61	X_DESC	X-axis descriptor if LSS=X (e.g. Blade Passing, Degrees RDZ, etc.)
C*15	E_UNIT	Corresponding English unit of measure for Y axis data.
C*482	CHNPAD	**Padding to 1024 bytes**

Data Reduction Process Parameters

The Data Reduction Process (DRP) does not require that all parameters in the Run Descriptor Record (RDR) be defined in order to function. However, it is recommended that the Standards Process (SP) and Facility Model (FM) be run as soon as possible after a run to define parameters essential to the DRP and the ability to progress beyond raw and primary data files. The parameters are divided into meaningful sections in the RDR headed by an integer toggle which is nonzero only if its associated parameters are defined. The definitions are incorporated at different points in the course of performing an experiment in the ATARR facility. The list below shows where in the evolution of a run that each parameter is defined. In some instances, a parameter is multiply defined by the Test Setup Process (TSP) and Facility Model. The values given when the Facility Model is run in RCDF generation mode takes precedence over previous definitions. As a point of clarification, the subroutine in which a parameter is defined in the FM is indicated to aid future software maintenance efforts. Designcond (DC), getgas (GG) and gasparam (GP) are the only three subroutines appearing in the list. All dimensional parameters are specified in SI units.

Parameter	Description	Where Set
FAC_TURB	FACILITY TURBINE PARAMETERS DATA TOGGLE	FM
NSTAGES	Number of Turbine stages	TSP
NBLADES(2)	Number of blades on each turbine stage	TSP
NVANES(2)	Number of vanes on each turbine stage	TSP
NGVA	Cross sectional area of NGV inlet	TSP
CVA	Choke Valve orifice Area	TSP
BLBA	Boundary layer bleed area	TSP
KCV	Choke valve discharge coefficient	TSP
KBL	Boundary layer bleed discharge coefficient	TSP
DTIP	Diameter of rotor	TSP&FM
VSUPPLY	Volume of supply tank	TSP&FM
ANGTO1	Anglular offset of blade 1 wrt indexer (deg)	TSP
STANDARD	STANDARDS DATA TOGGLE	SP
TSUPPLY0	Supply tank temperature	SP
PSUPPLY0	Supply tank pressure	SP
TINIT	Test section temperature at time zero	SP
PINIT	Test section pressure at time zero	SP
BEGTIM	Beginning time of interval for standards calculation	SP
ENDTIM	Ending time of interval for standards calculation	SP

Data Reduction Process Parameters

Parameter	Description	Where Set
FILL_DAT	FILL PARAMETERS DATA TOGGLE	FM
MOLEF(2)	Mole fractions of the test gas components	FM_GG
MW	Molecular weight of test gas	FM_GG
ACP(3)	Test gas specific heat coefficients	FM_GP
AGAM(3)	Test gas specific heat ratio coefficients	FM_GP
APR(3)	Test gas Prandtl Number coefficients	FM_GP
AMU(3)	Test gas dynamic viscosity coefficients	FM_GP
GCONS	Gas constant (Ru/MW)	FM_GG
PPGAS1	Partial pressure of first test gas component	FM_GG
TGAS1	Temperature of first test gas component	FM_GG
PFINAL	Final fill pressure	FM_GG
TFINAL	Final fill temperature	FM_GG
DES_COND	DESIGN CONDITIONS DATA TOGGLE	FM
TSTAG	Upstream design stagnation temperature	TSP
PSTAG	Upstream design stagnation pressure	TSP
WDTOTAL	Total design weight flow of test gas	FM_DC
WDTURB	Design weight flow of test gas through turbine	FM_DC
PRANDTL	Design Prandtl number of test gas	FM_DC