OPEN FILE REPORT 1984-28

Real-time Laboratory System (RLS)

Vol II: Programmer's Guide

Karin K.R. Tremaine

Chemistry Department
Alberta Research Council
11315-87 Ave.
Edmonton, Alberta
T6G 2C2

July, 1984

Alberta Research Council Open File Report

ABSTRACT

This document is the Programmer's Manual for the Real-time Laboratory System (RLS). This manual is intended for advanced RLS users and maintenance support staff, who are required to enhance and support the system. After a brief overview of the system, there is a step-by-step description of how to interface additional commands in the RLS system. A more detailed description of the RLS system discusses the design and operation. Support modules and RLS commands are summarized before the discussion of the system support. Several appendices are given, describing RLS file formats and the build procedures for the system.

Acknowledgments

The initiative for the RLS development came from the Chemistry Department and the Oil Sands Analytical group, mainly through the efforts of C. Richmond and C. Reichert. An external consultant, Lorne Bechtel, was hired to design and write the software, and produce the user documentation for the system. Testing of the software has been done at both the Campus and Clover Bar laboratories.

CHAPTER 1

INTRODUCTION

1.1 LABORATORY SYSTEM REQUIREMENTS

When chemical laboratory systems for the Alberta Research Council's Campus and Clover Bar sites were first being discussed, it was decided that a number of features had to be available, whether the resulting system was a purchased package, or an in-house development. The following characteristics were considered to be essential:

1. The command language had to be common to many applications. This would eliminate the training time and nuisance factor involved for people using more than one data analysis method. As well, it should remove the problem of requiring researchers to familiarize themselves with different computer systems.

2. There should be an easy mechanism to modify important parameters. This would reduce the operator time, would eliminate costly programming time and would allow technical people control over their own processing.

3. It should be easy to modify the sequence and type of processing applied to the experimental data, allowing the user to tailor the data analysis to varying requirements and conditions. A number of general modules could be used for many different applications, thus reducing the programming effort required to develop a comprehensive system.

4. There should be an easy interface mechanism for new program integration. Since the system must evolve and expand to meet the changing needs of a research establishment, it was considered to be essential that it be easy to expand the system, given limited manpower and money.

5. The system had to be small and easy to maintain. Since the software was to run on small real-time machines, size was critical. Limited manpower required that the system be easy to support. These requirements, plus some of those listed previously, indicated that the system should be modular.

At that time, there were no commercial packages available which met the above conditions and could be used in a data acquisition environment. On the basis of that evaluation, it was decided to proceed with in-house development of a Real-time Lab System (RLS) to meet user requirements. The software package, developed by external consultant Lorne Bechtel, is the property of the Alberta Research Council.

1.2 RLS OVERVIEW

The RLS system provides an English-like language to control a library of laboratory oriented programs. To use the existing RLS modules, the user interacts with the RLS command line interpreter, using an English-like command. RLS parses this command, searching a command catalog for the command. Once found it reads a table of defaults before initiating an appropriate application program. When RLS has determined which default values are to be sent to the application process, the information is passed to the executing program through a temporary file maintained by RLS. The executing program interacts with the user's input/output files and devices to perform the required operations. RLS waits for the executing program to terminate before returning to the user for more command instructions.

The RLS core, which handles the command parsing, keyword value control, and application processing control, also has a number of commands which allow the user to interface programs, and check and modify defaults without knowing a great deal about the operating system on the computer in question.

In addition to the central RLS modules, a number of application processes have been written. Some examples are real-time device control and data acquisition, chromatography, and x-y plotting.

1.3 RLS STATUS

The RLS system, as it now stands, is a very good prototype. The overall design is good, it is extremely flexible, and when used by knowledgable operators, it is a very powerful laboratory tool. With over 80 program modules, it is a system that is comparable in size and complexity to a computer operating system such as the DEC RSX operating system itself. The system is currently in use in a number of departments within the Alberta Research Council. Once the problems discovered during the test phase have been resolved, RLS should be ready for general distribution. RLS system limitations and problem areas are discussed in detail in Section 6.5.

1.4 RLS DOCUMENTATION

Two volumes of RLS documentation have been prepared to provide both user and system support. Volume I (1) is intended for the laboratory user, using existing RLS commands. This volume, Volume II, is designed for programmers, both at the user level and the system level. As well as describing how to interface high-level language programs to the RLS system, information is also provided for system maintenance and support.

For the casual user who wishes to incorporate a private program into the RLS system, Chapter 2 discusses program interfacing, using examples where possible. For a detailed description of the RLS core system and file interaction, read Chapter 3. This chapter is recommended reading for people involved in system upgrading and system maintenanc RLS libraries are documented in Chapter 4. Chapter 5 gives a list of all the RLS commands (including applications,) as well as the directories where the sources reside. RLS libraries are documented in Chapter 5. System generation and maintenance is discussed in Chapter 6.

1.5 REFERENCES

1. Bechtel, Lorne. "Real-time Laboratory System (RLS). Volume I: User Guide." Alberta Research Council Open File Report, 1984.

CHAPTER 2

INTERFACING AN APPLICATION PROCESS WITH RLS

2.1 INTERFACING NEW APPLICATION PROCESSES

Users can write new application programs, in languages such as FORTRAN or C, and merge them with the existing RLS system. The steps involved are straightforward, and can be done at two levels.

At the most trivial level, a command can be inserted in the command catalog, and RLS will install and run the process without any additional interaction. All processing instructions must be provided by the user.

The more advanced level of interfacing will give the user access to all the RLS processing features. This method involves:

1. Inserting a command in the command catalog (CCF)
2. Creating a command definition file (CDF)
3. Writing the application process
4. Building an executable task

The remainder of this chapter steps through RLS program interfacing. The program used for illustration purposes is a simple integration application, written in 'C'. Any user written program can be added, using the same procedure.

2.2 ENTERING A COMMAND IN THE COMMAND CATALOG

The command catalog is a list of all the valid RLS commands, along with the corresponding programs and command definition files. Adding a new command requires that the user specify the context (user determined), the identifier or command, the program or executable, and the keyword or command definition file. The user must also specify whether or not the program is an executable task or an RLS command procedure.

To insert a new command to the RLS command catalog, use the RLS 'ADD COMMAND' command. The following example shows how to include a new command:

```
==============================================
RLS<cr>
RLS>add command ?<cr>
CONTEXT          GENERAL :  example<cr>
IDENTIFIER               :  test integral<cr>
PROGRAM-FILE             :  integral<cr>
KEYWORD-FILE             :  integral<cr>
PROGRAM          NO      :  yes<cr>
==============================================
```

The resulting command would be 'TEST INTEGRAL'. It is not a requirement that the task name and the command definition file have the same name. However, system maintenance is much easier if this convention is followed. By entering 'yes' to the 'program' prompt, the user is telling RLS that the 'program-file' is an executable task rather than a command procedure. If the user does not make an entry for keyword file, RLS would assume that there isn't a CDF for that command.

If a mistake is made entering the command, it can be deleted using the RLS 'DELETE COMMAND' command.

2.3 CREATING A COMMAND DEFINITION

The command definition file provides a description and default values for all keywords associated with a given command. RLS reads this file, extracts default values and prompts users for missing values. The user can over-ride the values in this file by entering the keyword and an appropriate value after the keyword as part of the command line. The integration program used for illustration purposes expects to have the input filename, the output filename and the stepsize for equidistant files as keyword parameters.

The RLS command language allows the creation of new command definition files, using the command 'CREATE DEFINITION'. An example is given below:

```
=======================================================
RLS<cr>
RLS>create definition<cr>
ENTER FILE NAME:  integral.rls<cr>
ENTER KEYWORD (or return to finish):  input-file<cr>
ENTER DATA TYPE:   file<cr>
ENTER INPUT/OUTPUT STREAM:   input1<cr>
ENTER A HALF-LINE DESCRIPTION:  input data set name<cr>
ENTER DEFAULT VALUE:  ?<cr>
ENTER KEYWORD (or return to finish):  output-file<cr>
ENTER DATA TYPE:  file<cr>
ENTER INPUT/OUTPUT STREAM:  output1<cr>
ENTER A HALF-LINE DESCRIPTION:  output data set name<cr>
ENTER DEFAULT VALUE:  ?<cr>
ENTER KEYWORD (or return to finish):  stepsize<cr>
ENTER DATA TYPE:  float<cr>
ENTER A HALF-LINE DESCRIPTION:  stepsize for file<cr>
ENTER DEFAULT VALUE:  0.0<cr>
ENTER A KEYWORD (or return to finish):  <cr>
=======================================================
```

RLS creates a file called 'integral.rls', with the information supplied, and writes the resulting file in the user's disk area. The actual file created by the above sequence of commands is listed below:

```
================================================
keyword input-file -
        type file -
        stream input1 -
        help "input data set name" -
        default ?
keyword output-file -
        type file -
        stream output1
        help "output data set name" -
        default ?
keyword stepsize -
        type float -
        help "step size for equidistant data" -
        default 0.
================================================
```

2.4 CREATING SOURCE PROGRAMS

The general format of most programs within the RLS system is as follows:

1) Title page
2) Declarations
3) Reading user parameters
4) Application processing
5) Return code

2.4.1 Title Page

The title page describes the use of the command, and is maintained as a comment at the front of the program. Usually, the following information would be included:

1. The general command

2. Application function

3. Keyword list and description corresponding to CDF

4. Command example

This section ends with a formfeed, and can be extracted using the 'makhlp' program. It will be placed in 'filename.hlp' where 'filename' is the name of the application program. When the user requests RLS help, the system will display the information in the corresponding .hlp file. The next example illustrates the type of information to be included on the title page.

```
===================================================================
/*
COMMAND

        TEST INTEGRAL

ACTION

        This command generates the integral of x-y data
        contained in an INPUT-FILE into an OUTPUT-FILE.

KEYWORDS

        INPUT-FILE       CHAR    input file name
        OUTPUT-FILE      CHAR    output file name
        STEPSIZE         REAL    sample point interval

EXAMPLE

        RLS>TEST INTEGRAL INPUT-FILE DATA.DAT OUTPUT-FILE INT.DAT
*/
===================================================================
```

2.4.2 Declarations

The declarations would include any data type declarations required by the program. An example of the typical declarations of a C program are given below:

```
        ==============================================
        FIO infio, outfio,*fopen(),*fcreate();
        char filnam[40], outnam[40],upfnam[40];
```

```
extern double getfloat();
double stpsiz, x, y, sum, delta, xl;
int equi;
=============================================
```

2.4.3 Reading User Parameters

In this section of code, the user retrieves the file name of the command modifier file, and using the functions available, reads the file contents. The CMF is an intermediate file, maintained by RLS, which contains the keyword values required by the application program. RLS creates this file after reading the command definition file and the user command line. The following sequence of commands illustrates how to retrieve the CMF file name, open the file, and read the keyword values before closing the file.

```
===============================================
cmfnam(upfnam,sizeof(upfnam));
fopen(&infio,upfnam,READ);
getstring (&infio, filnam, sizeof(filnam));
getstring (&infio, outnam, sizeof(outnam));
stpsiz = getfloat (&infio);
fclose (&infio);
===============================================
```

For each parameter, one read is required. To read the ascii free format files, the following support utilities have been developed:

type	routine
integer	getint()
floating point	getfloat()
logical	getlogical()
string	getstring()

FORTRAN users wishing to use the C I/O routines to read files should refer to the chapter dealing with RLSCLIB for further instructions.

2.4.4 Application Processing

There is no restriction on the type of processing the user wishes to perform. If the files created by the user are to be used by the RLS core or other RLS application programs, they must adhere to the following conventions:

1. The files must be ascii free format

2. A file header must appear at the start. The minimum file header would be a data string (ie DATA) without a trailing dash. The file header for the preceeding CDF example could be the following:

   ```
   ==========================
   title Data from EPR -
   stepsize 1.0 -
   data
   0.0
     .
   ==========================
   ```

   In the above example, the first word would be a keyword, and the value following it would be the value assigned to the keyword. The trailing dash indicates that the header is continued on the next line. If the user wishes to ignore the file header, there are routines (xfrhdr(),skiphdr()) which either transfer the header to the output file, or skip the header completely. Additional keywords in the header will be transferred in the case of a header transfer, or ignored whenreading the header.

3. FORTRAN programs should use the carriage control convention 'logical record' to read/write files that are used with other commands implemented in the C language.

2.4.5 Return Code

The application program must return a suitable completion code, indicating successful program termination. The table below indicates the appropriate calls for the different completion codes.

EXIT STATUS	FORTRAN	C
Successful completion	call exit(1) (suppresses STOP message) stop (types STOP message on terminal)	exit(1);
Returns 'TRUE' value	call exst(1)	exit(1);
Returns 'FALSE' value	call exst(0)	exit(0);

2.5 BUILDING AN EXECUTABLE TASK IMAGE

To build an executable version of the program (C language), the following sequence of instructions is issued:

```
==========================================================
$ cc integral
$ link /notrace integral,[rls.library.vms]rlsclib/libr,-
chdr,clib/libr
==========================================================
```

An executable FORTRAN version of the 'integral' program, assuming that the 'C' library routines were used, would be built with the following command sequence:

```
=================================================================
$ for integral

$ link /notrace integral,[rls.library.vms]rlsclib/libr,-
sys$library:chdr,[rls.library.vms]clib/libr,sys$library:clib/libr
=================================================================
```

For people wishing to build an RSX version, the general format for the compile and task build would be:

```
=======================================================================
                C                          FORTRAN
=======================================================================
COMPILE    $ ccll integral         $ mcr for integral = integral.for


TASK BUILD $ mcr tkb
           TKB>integral/fp/cp=integral,[rls.library.rsx]rlsclib/lb,-
           TKB>sys$library:chdr,clib/lb
           TKB>/
           ENTER OPTIONS:
           TKB>reslib=sys$library:fcsres/ro
           TKB>units=10
           TKB>stack=3000
           TKB>asg=sy:1:2:3:4:6:7:8
           //
=======================================================================
```

2.6 PROGRAM EXAMPLE

Earlier sections described how to set up the command and associated files for a program called INTEGRAL. The 'integral' program is now part of the RLS system, and can be executed as just one of the many RLS modules. To execute the program, either of the following commands can be used:

```
===========================================================
        RLS>test integrate input data.dat output int.dat
   OR
        RLS>test integrate<cr>
        input-file      ?: data.dat
        output file     ?: int.dat
===========================================================
```

In the second example, the user is prompted for the names of the input/output files since the default value is a question mark. The stepsize, in both examples, is taken from the default table.

For user's reference, the program listing ,as well as the input and output files are listed in the following pages.

```
/*
COMMAND

        TEST INTEGRAL

ACTION

        This command generates the integral of x-y data
        contained in an INPUT-FILE into an OUTPUT-FILE.

KEYWORDS

        INPUT-FILE      CHAR    input file name
        OUTPUT-FILE     CHAR    output file name
        STEPSIZE        REAL    sample point interval

EXAMPLE

        RLS>TEST INTEGRAL INPUT-FILE DATA.DAT OUTPUT-FILE INT.DAT
*/
<FF>
#include <std.h>
#define FALSE  0
#define TRUE   1

main ()
        {
        FIO infio, outfio,*fopen(),*fcreate();
        char filnam[40], outnam[40],upfnam[40];
        extern double getfloat();
        double stpsiz, x, y, sum, delta, x1;
        int equi;

/*   read user parameters                                   */

        cmfnam(upfnam,sizeof(upfnam));
        fopen(&infio,upfnam,READ);
        getstring (&infio, filnam, sizeof(filnam));
        getstring (&infio, outnam, sizeof(outnam));
        stpsiz = getfloat (&infio);
        fclose (&infio);

        equi = (stpsiz > 0.) ? TRUE : FALSE;

/*   open files for input/output                       */

        if (!fopen (&infio, filnam, READ))
                {
                putfmt ("I am unable to open the file %p\n", filnam);
                exit(1);
                }
        if (!fcreate (&outfio, outnam, WRITE))
                {
                putfmt ("I am unable to create the file %p\n", outnam);
                exit (1);
                }
/*    transfer file header from input to output file        */

        xfrhdr (&infio, &outfio);

/*    integrate                                                 */

        sum = 0.;
        x = 0.;
        xl = 0.;
        while (!eofon (&infio))
                {
                if (!equi)
                        x = getfloat (&infio);
                else
                        x = x + stpsiz;
                y = getfloat (&infio);
                if (eofon (&infio))
                        break;

                if (equi)
                        delta = stpsiz;
                else
                        delta = x - xl;

                sum = sum + ( y * delta);

                if (!equi)
                        putf (&outfio, "%12.5d\n", x);
                putf (&outfio, "%12.5d\n", sum);

                xl = x;
                }

        fclose (&infio);
        fclose (&outfio);

        exit(1);
        }
```

The following file, DATA.DAT, was used as input.

```
title Data from EPR -
stepsize 1.0 -
data
0
1
2
3
4
5
6
7
8
9
10
```

The output, as a result of the RLS command, would be stored in a data file called 'int.dat'. The contents of the file are listed below.

```
title Data from EPR -
stepsize 1.0 -
data
 0.00000e+00
 1.00000e+00
 3.00000e+00
 6.00000e+00
 1.00000e+01
 1.50000e+01
 2.10000e+01
 2.80000e+01
 3.60000e+01
 4.50000e+01
 5.50000e+01
```

CHAPTER 3

RLS CORE SYSTEM

3.1 OVERVIEW

The RLS command line interpreter is the central program of the system that interacts with the user. This program reads command lines from the user, parses the line for commands and keyword/value pairs, and then initiates the appropriate program.

Commands are entered as a string of words and/or numbers. The blank character (space bar) is used as a separator between all words and symbols. Commands are of the general form:

```
        RLS>verb object keyword value keyword value ...
```

The 'verb object' implies that an action is to be performed on a general object such as a device or data file.

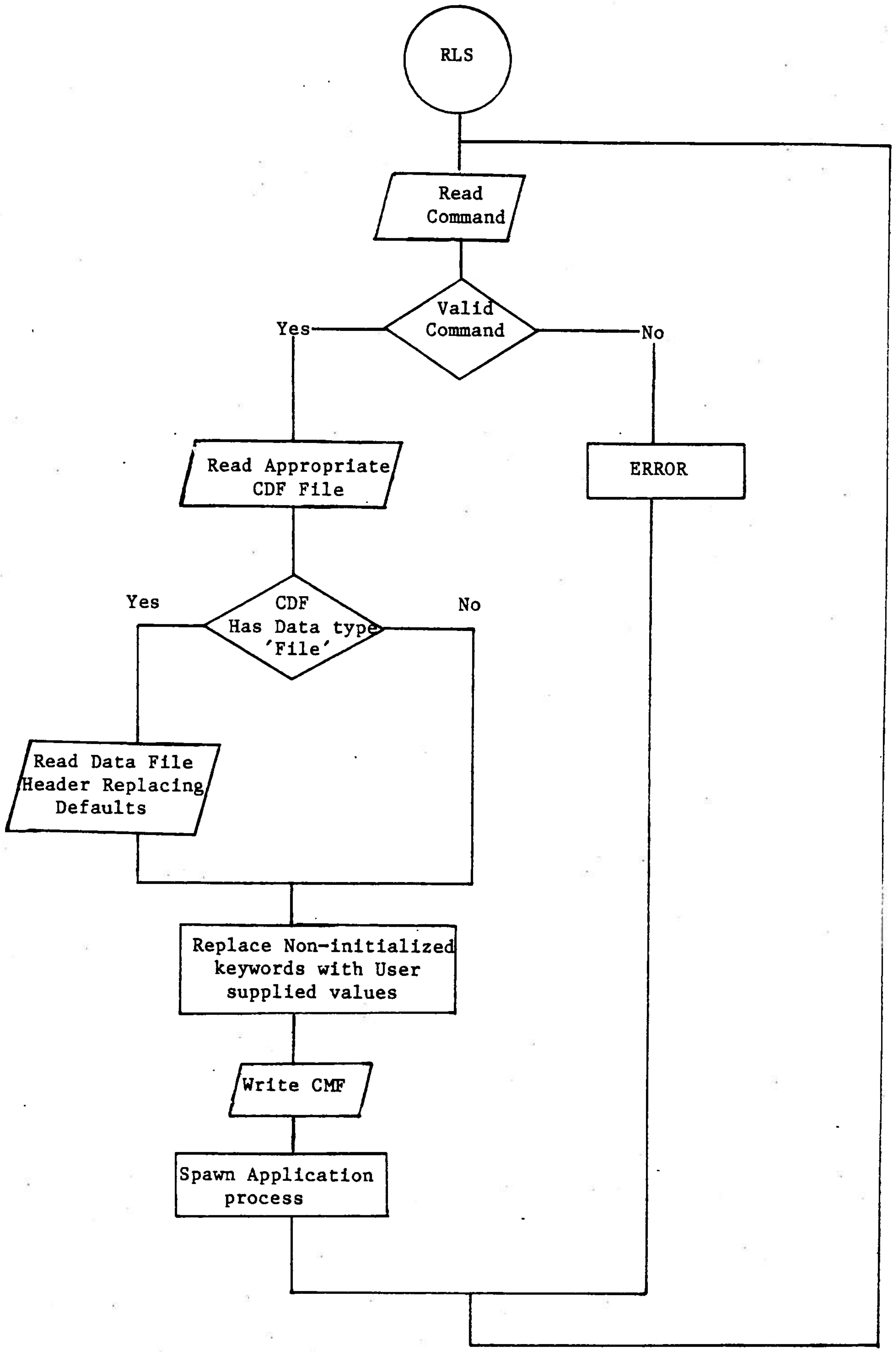
```
 eg.    start analog
        print report
```

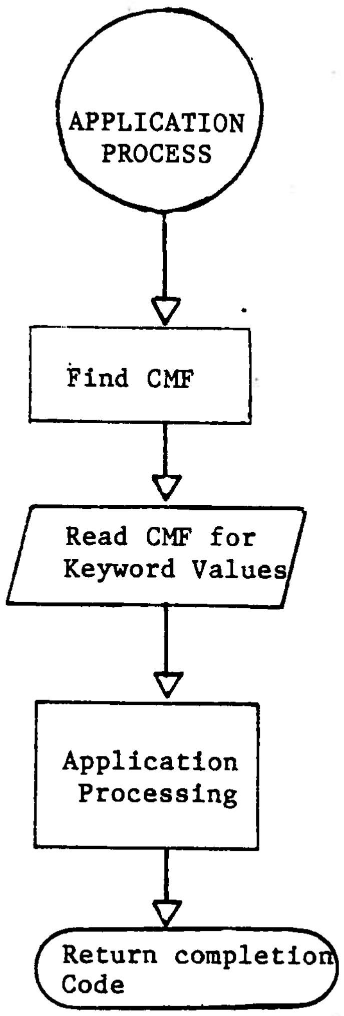
The 'keyword' indentifies the name of the parameter to be changed and the 'value' specifies the new value to be used. The set of 'keyword value' pairs do not have to be in any specific order. Some values, such as titles, are represented as an arbitrary string of characters with embedded blanks. The user encloses such sequences of characters within double quote marks in order to keep the words together as one 'value'. If keyword/value pairs are not specified, the value assigned is taken from a default table.

Much of the RLS command parsing and information control is handled through a system of files created and maintained by the RLS core system. These files are invoked at different levels, to interpret user commands, select default tables for specific commands and to pass information on to the spawned task. The main files that the RLS command line interpreter works with are:

1. The command catalog file (CCF)
2. The command definition file (CDF)
3. The command modifier file (CMF)

The interaction between the above files and the RLS command line interpreter is indicated by the following diagrams:


RLS
Read Command
Valid Command
Yes
No
Read Appropriate CDF File
ERROR
CDF Has Data type 'File'
Yes
No
Read Data File Header Replacing Defaults
Replace Non-initialized keywords with User supplied values
Write CMF
Spawn Application process


APPLICATION PROCESS
Find CMF
Read CMF for Keyword Values
Application Processing
Return completion Code

3.1.1 The Command Catalog

The command catalog contains all valid RLS commands. Two levels of the catalog are maintained - one global and one local. The global file resides in a system area and is available on a read only basis to all users. The local catalog is located in the user's area and can only be changed or accessed by the owner. When executing a command, RLS will scan the local catalog before searching the global catalog.

Whenever a command is added to the catalog, the following information is required:

1. Context in which the application program is used. Examples are chromatography, math routines, etc. The user can, of course, specify any context.
2. Identifiers or command strings
3. Command definition file name
4. Application program file name
5. Indicate whether the program is an executable image or a command file

For each valid RLS command, the following line will appear in the catalog file:

```
  context       "command string"  'cdf.file'    'task.name'

  where:
        "command string" - RLS command
        'cdf.file'       - Command definition file
                x            - no cdf file required
                *            - cdf file required, named 'cdf.file'
        'task.name'      - process task to be executed
                @            - process task a command program
                $            - process task an executable image
```

The 'x','*','@' and '$' are determined by the RLS program, on the basis of the user response to the 'ADD COMMAND' prompts. RLS inserts these symbols, immediately preceding the 'cdf.file' or the 'task.name', for later use during task spawning.

Regardless of the catalog level being modified (ie. global or local), the 'ADD COMMAND' feature of RLS is used. However, only a person having access to the RLS area can modify the global catalog file. When adding commands to the general catalog, 'USERS:[RLS]' must be specified as part of the file name. On the VAX system, this is the complete file specification. On the RSX system, RLS substitutes 'ei:[277,54]', which is the RLS disk and UIC. This general file specification is required for RLS to search the RLS area. If it was not specified, RLS would search only the user's UIC and disk area for the file.

3.1.2 The Command Definition File

The command definition provides information to the RLS control program regarding the input required by the application program. By convention, the CDF information is stored in a data file with the extension 'rls', in the RLS area. If, however, the user chooses to change defaults for a given command using the RLS 'SET command keyword value', then a modified copy of the command definition file will be placed in the user's area. The search order for the CDF's is local, then global. For each keyword to be processed by RLS, the command definition file contains:

1. the keyword
2. the data type of the keyword(integer,floating point,string)
3. the stream ID for input/output files

4. the description of the ke ord function

5. the keyword default value

An example of a command definition file is given below:

```
==============================================
keyword raw-file -
        type file -
        stream input1 -
        help "simdis file with additive" -
        default simdis.raw
keyword rsimdis-file -
        type file -
        stream output1 -
        help "additive area removed" -
        default simdis.dat
keyword start -
        type float -
        help "start of additive peak" -
        default 0.0
keyword stop -
        type float -
        help "end of additive peak" -
        default 0.0
==============================================
```

RLS interacts with the user in a very general style to obtain the keyword value pairs. It determines the defaults sent to the application process using the following algorithm:

1. Reads the appropriate command definition file

2. If there is an input file, it searches the file header of that file for the keywords, substituting values where appropriate.

3. Replaces non-initialized keywords with keyword values supplied by the user, either in the initial command line, or through prompting.

Once all the keyword values requested by the user have been determined, the values only, are written to the command modifier file.

3.1.3 The Command Modifier File

The command modifier file (CMF), written by the RLS control program, defines the keyword values to be read by the application process. The values appear in the order they were defined in the command definition file. The CMF file will be written in the user's disk area, and will be deleted once the process has successfully terminated. The actual file name is system/process dependant and application processes should use the function/subroutine CMFNAM() to retrieve the correct name.

The file contents are written in ascii free format, and can be read using the following library subroutines and functions:

type	routine
integer	getint()
floating point	getfloat()
logical	getlogical()
string	getstring()

Each variable to be read in is requested by an individual call to one of these routines. Further documentation of these routines is given in the Chapter discussing RLS libraries. As well, an example of an RLS integrated program is given in chapter 2.

3.2 APPLICATION PROCESS REQUIREMENTS

The application process to be integrated with the RLS system must follow certain internal conventions. Files to be used by other RLS modules should adhere to the RLS standard. Completion codes must be specified by the application process. Also, Fortran programs wishing to use C language subroutines should refer to the Library section dealing with RLSCLIB for detailed discussion on their use.

3.2.1 File Handling

The primary inter-program interfaces in the RLS system are the input/output files of the programs. The files are used to pass information among the programs, and should adhere to the following conventions:

1. Files are text/character (ASCII) for
2. Data appears in free format, rather than assuming fixed column positions.
3. A file header must appear at the start of the file.

Detailed file format descriptions are given in Appendix A.

3.2.2 Program Exit

The exit status of an application program is returned to RLS by the computer operating system. This exit status must be set by the application program. If the program completes successfully, it should return with an 'exit' code of 1. RLS command procedures using the IF/THEN/ELSE feature require that the program return a TRUE/FALSE value. Refer to the table below for the subroutine/function calls to return suitable exit codes.

EXIT STATUS	FORTRAN	C
Successful completion	call exit(1) (suppresses STOP message) stop (types STOP message on terminal)	exit(1);
Returns 'TRUE' value	call exst(1)	exit(1);
Returns 'FALSE' value	call exst(0)	exit(0);

3.3 COMMAND PROCEDURES

In addition to the general command line parsing capability, RLS handles rudimentary indirect command files and simple programming features within the command procedures.

The indirect command processing features allows the user to initiate a series of commands with the use of one command. Once the process has been started, it can usually be left to complete without additional user intervention.

To build a command procedure, use any text editor on the computer. The commands are entered in to the data file exactly as they would appear if they were entered on the terminal directly. Any valid RLS command can be used in the command procedure. On RSX systems, system commands are passed on to MCR for proper processing. Therefore, any valid MCR command can also be used. An example of a general command procedure would be:

```
===========================================
START ANALOG CHANNEL 1 DURATION 5
```

```
WAIT ANALOG CHANNEL 1
FIND PEAKS
ZERO BASELINE
PREPARE REPORT
===========================================
```

When RLS executes this procedure, it samples channel 1 for a duration of 5 minutes, before doing all the peak processing. The commands will be executed sequentially, with one command completing before the next one is initiated.

Once the command procedure has been created, an appropriate command must be added to the command catalog, using the 'ADD COMMAND' feature. For example, to include the above procedure, using the command line 'DO CALIBRATION', the following should be done:

```
=========================================
RLS>add command ?
context          general : command
identifier               : do calibration
program-ident            : calib
identifier-ident         : <cr>
program          no      : <cr>
=========================================
```

Since the command will be invoking an indirect command procedure, rather than an executable task, a CDF will not be necessary for this command. Defaults will be selected when the commands within the procedure are initiated. The user tells RLS that it is a command procedure rather than an executable image by stating that the 'program-ident' is not a program. The command procedure can now be executed in exactly the same manner as any of the other commands.

Where specified in the command procedure, keyword values will be substituted. Otherwise, all keyword values will be selected from the correct command definition files. It is possible to pass values down into the command procedure. For example:

```
        RLS>do calibration frequency 5
```

'Frequency 5' is passed down to any command which has the keyword 'frequency'. Some care should be taken when using this format for the command. If one of the processes modifies 'frequency', the value will be over-ridden by the user, possibly causing erroneous results.

For handling more complex command sequences, two control features have been implemented, an IF/THEN/ELSE statement and a GOTO.

3.3.1 IF/THEN/ELSE

These statements allow certain sets of commands to be executed conditionally, based on the results of some calculations. Each of these words must appear as the FIRST WORD in front of the command. For example, assume that a command called 'NO STANDARD' has been written and installed in the chromatography applications section. It checks that the standard peak was found in the analysis and returns TRUE if it is and FALSE if it is not. Then the following sequence would conditionally inject the next compound:

```
===================================
IF NO STANDARD
        THEN INJECT STANDARD
        ELSE INJECT NEXT
===================================
```

3.3.2 GOTO

The GOTO statement enables a non-sequential execution of program statements. For example, assume that a program command called 'COLUMN OK' checks whether the results from the analyses indicate that the column is still operative. Then the following sequence would enable repeated injections unless the column was determined to be inoperative:

```
==================================
REPEAT: INJECT NEXT
        .
        .
        do analysis
        .
        .
        IF COLUMN OK
                THEN GOTO REPEAT
==================================
```

Note that the target lab of the GOTO command is terminated by a colon character and must have a space between it and the following command.

Programs still have to be written which will return TRUE/FALSE values to the IF/THEN/ELSE. It is assumed that these will be written on demand to satisfy user groups and their specific applications.

CHAPTER 4

FUNCTION AND SUBROUTINE LIBRARIES

4.1 RLS REQUIRED LIBRARIES

A number of libraries are required for the RLS system generation. Some of the libraries were developed to support the RLS command line interpreter and the application processes. The remainder are either standard RSX or VMS libraries required to build the processes for the different operating systems. The required libraries are listed below:

Library	File Name RSX	File Name VMS	Description
CLIB	CLIB11	CLIB	C compiler run-time library
EXELIB	EXELIB	N/A	RSX executive library
EXEMC	EXEMC	N/A	RSX - required for device drivers and privileged tasks
MATH	MATH11	MATH	Math support library
PLOTDEV	PLOTDEV11	PLOTDEV	Graphics support library
PLOTLIB	PLOTLIB11	PLOTLIB	CALCOMP compatible library
RLSCLIB	RLSCLIB11	RLSCLIB	RLS C language support library
TARGETLIB	TARGETLIB	N/A	DEVICE I/O for data collection. One for each node: [RLS.LIBRARY.RSX.ALSARC] [RLS.LIBRARY.RSX.CHMARC]

CLIB, EXELIB, and EXEMC are system libraries. Refer to the 'C' runtime library for more information on CLIB. EXELIB and EXEMC are required for the RSX system builds. EXEMC.MLB provides code to allow software to refer to offsets within the Executive and system definitions of the Executive data structures. This library is required for building privileged tasks and for incorporating specially written device drivers. EXELIB.OLB is the Executive library and contains the definitions of the Executive symbols. Refer to the RSX documentation for more information regarding these libraries. The two target libraries are the actual data acquisition software. For Clover Bar, part of this library, LPA.OBJ, comes with the LPA distribution kit, and is incorporated in TARGETLIB.

The RLSCLIB library is an RLS C language support library. A number of these modules have been written to remove inconsistencies between the VMS and compatibility mode compiler. Others have been written to support a number of features available in the RLS command line interpreter. It should be noted, that although the VMS and RSX libraries are similar, they are not identical.

The remainder of the libraries are application support libraries. PLOTLIB and PLOTDEV are support for the RLS graphics. PLOTLIB contains the CALCOMP compatible routines, and PLOTDEV the device specific modules. MATH is the support for the mathematics section. Both the VMS and RSX systems are supported by the above libraries.

The libraries are described in more detail in the following sections.

4.2 PLOTLIB

Plotlib is a set of subroutines which are compatible with the typical CALCOMP plot library. These routines can be called by FORTRAN and the following documentation assumes FORTRAN call conventions. C language routines can call the routines by prepending a 'c' character to each name. For example:

```
FORTRAN          CALL PLOT(FX,FY,IC)
C                CPLOT(FX,FY,IC);
```

Although the routines are meant to be as CALCOMP compatible as possible, there are some major differences. The major differences are:

1. Pen movement commands are in centimeters rather than inches.
2. The SYMBOL() character set is somewhat different from the usual CALCOMP set.

Please note that the unit number supplied as a parameter to the PLOTS() subroutine is not the same as the RSX logical unit number. Use the CRELOG() subroutine of the RLSCLIB library in order to assign a file name to this unit number. For example, if you call PLOTS() with unit number 8 as in (C language version):

`CPLOTS (dummy1, dummy2, 8);` then create a name for the file with the following call:

```
CALL CRELOG("FOR008",filename):
```

4.2.1 Directory Listing Of Plotlib Library

Since the documentation for the fortran version is similar to that of the C versions, no attempt has been made to include both versions. All the help information in the following section is for the C version.

PLOTLIB	PLOTLIB(cont'd)
AXIS	LINE
CAXIS	MINMAX
CFACTOR	NEWPEN
CLINE	NUMBER
CNEWPEN	PLOT
CNUMBER	PLOTS
CODE	PLOTSYMB
CPLOTS	SCALE
CSCALE	SYMBOL
CWHERE	S_FACTOR
FACTOR	WHERE

```
/*
NAME

        AXIS - draw an axis with tick marks and annotation

SYNOPSIS

        CALL AXIS (FX, FY, TITLE, NTITLE, AXLEN, ANGLE, AXMIN, AXDELT)
        REAL FX
        REAL FY
        LOGICAL*1 TITLE(NTITLE)
        INTEGER NTITLE
        REAL AXLEN
        REAL ANGLE
        REAL AXMIN
        REAL AXDELT

FUNCTION

        This routine draws an axis with tick marks and annotation.
        The axis is drawn at any specified angle. Tick marks are
        drawn at intervals along the axis and annotated with
        the numeric value that they represent. A title may be
        plotted along the axis. The tick marks and title may
        be placed on either side of the axis. The numeric
        values may be scaled by some power of ten. If they are
        then this will be noted by placing the notation
        '*10^n' on the title line.

INPUT

        FX                REAL            X coordinate of start of axis
        FY                REAL            Y coordinate of start of axis
        TITLE(NTITLE)     LOGICAL*1       axis title
        NTITLE            INTEGER         number of characters in title
                                          if NTITLE<0 then ticks and title
                                          will be on the clockwise side of
                                          the axis
        AXLEN             REAL            length of axis
        ANGLE             REAL            angle of axis in degrees
                                          counter-clockwise from the
                                          positive X direction
        AXMIN             REAL            value for first tick mark of axis
        AXDELT            REAL            units per tick mark

OUTPUT

        none

RETURNS

        none
*/
```

```
/*
NAME

        FACTOR - scale the plot

SYNOPSIS

        CALL FACTOR (FACT)
        REAL FACT

FUNCTION

        This routine scales further plotting by an
        arbitrary amount. Initially FACT=1.0 so that
        the plotting units are in centimeters.

INPUT

        FACT    REAL    scaling factor

OUTPUT

        none

RETURNS

        none
*/
```

```
/*
NAME

        LINE - draw (scaled) line and/or symbols

SYNOPSIS

        CALL LINE (FX, FY, N, IREP, ICHREP, ICHAR)
        REAL X(IREP,N)
        REAL Y(IREP,N)
        INTEGER N
        INTEGER IREP
        INTEGER ICHREP
        INTEGER ICHAR

FUNCTION

        This routine draws a line using the two REAL arrays
        X and Y. These arrays must be scaled before calling
        LINE() by the SCALE() subroutine. The scaling
        parameters MIN and SCALE are assumed to be contained
        as two members of each array at locations X(N+1),X(N+2),
        and Y(N+1),Y(N+2). Under control of the parameter
        ICHREP the lines only, lines and symbols, or symbols only
        may be plotted at each of the locations.

INPUT

        X(IREP,N)       REAL    new X coordinate
        Y(IREP,N)       REAL    new Y coordinate
        N               INTEGER number of data points in X and Y
        IREP            INTEGER repetition count, if X and Y
                        are dimensioned X(IREP,N) and Y(IREP,N)
                        then the I'th data elements will be at
                        X(1,I) and Y(1,I).
        ICHREP          INTEGER control parameter such that:
                        = 0,    only straight lines are plotted
                        > 0,    line is plotted and then symbol given by
                                ICHAR will be plotted
                        < 0,    only symbol given by ICHAR will be plotted
        ICHAR           INTEGER symbol number

OUTPUT

        none

RETURNS

        none
*/
```

```
/*
NAME

        NEWPEN - select new pen

SYNOPSIS

        CALL NEWPEN (INP)
        INTEGER INP

FUNCTION

        This routine selects a new pen for further plotting
        on plotters supporting automatic pen changes.
        The number is device dependent and usually in
        the range from 0-4.

INPUT

        INP     INTEGER         new pen number

OUTPUT

        none

RETURNS

        none
*/
```

/*
NAME

NUMBER - plot floating point number

SYNOPSIS

```
CALL NUMBER (FX, FY, HEIGHT, FNUM, THETA, NN)
REAL FX
REAL FY
REAL HEIGHT
REAL FNUM
REAL THETA
INTEGER NN
```

FUNCTION

This routine plots a floating point number at the location specified. The precision of the number is controlled by the parameter NN.

INPUT

FX	REAL	new X coordinate
FY	REAL	new Y coordinate
HEIGHT	REAL	height of number
FNUM	REAL	number to be plotted
THETA	REAL	angle of inclination in degrees
NN	INTEGER	precision of number:
	> 0,	NN digits will be plotted after the decimal point.
	= 0,	only integer portion and decimal are plotted.
	=-1,	only integer portion is plotted
	<-1,	ABS(NN)-1 digits are truncated from the integer portion

OUTPUT

none

RETURNS

none
*/

```
/*
NAME

        PLOT - draw a line (and other things)

SYNOPSIS

        CALL PLOT (FX, FY, IC)
        REAL FX
        REAL FY
        INTEGER IC

FUNCTION

        This routine moves or draws a line from the
        current position to a new position depending
        on the value of the control parameter IC.
        The origin of the plot (0.,0.) may also be changed
        to the new position if IC is given as a negative
        number. Note that a call with IC=999 terminates
        the plot and should be used at the end of
        every plotting program.

INPUT

        FX      REAL    new X coordinate in centimeters
        FY      REAL    new Y coordinate in centimeters
        IC      INTEGER control argument
                =  2,   draw a line to new position
                =  3,   move to new position
                = -2,   draw a line to new position
                        and move origin to new position
                = -3,   move to new position and
                        move origin to new position
                = 999   terminate the plot

OUTPUT

        none

RETURNS

        none

*/
```

```
/*
NAME

        PLOTS - initialize plotting

SYNOPSIS

        CALL PLOTS (DUMMY1, DUMMY2, UNIT)
        INTEGER DUMMY1
        INTEGER DUMMY2
        INTEGER UNIT

FUNCTION

        This routine initializes the plotting and must
        be made before any call to PLOT. The plot description
        will be written to a file with the name FORx.DAT
        where UNIT=x. The unit number is not the logical
        unit number as used in most of RSX, but is
        the equivalence name as used in VMS. If the
        file name FORx.DAT is assigned to a physical file
        name then the new physical name will be used.
        The RSX version of this library always uses logical
        unit number 1 for the I/O.

INPUT

        DUMMY1  INTEGER
        DUMMY2  INTEGER
        UNIT    INTEGER         unit number

RETURNS

        none
*/
```

```
/*
NAME

        SCALE - produce scaling parameter for vectors

SYNOPSIS

        CALL SCALE (VA, AXLEN, N, IREP)
        REAL VA(IREP,N)
        REAL AXLEN
        INTEGER N
        INTEGER IREP

FUNCTION

        This routine produces the scaling parameters
        MIN and SCALE at the end of the vector provided.
        The scaling parameters are calculated so as to
        produce 'nice' values for the tick marks.

INPUT

        VA(IREP,N)        REAL    vector of data points to be scaled
        AXLEN             REAL    maximum length to scale data
        N                 INTEGER number of data points
        IREP              INTEGER repetition count such that if VA
                                  is declared VA(IREP,N) then the I'th data
                                  point is at VA(1,I).

OUTPUT

        none

RETURNS

        none
*/
```

/*
NAME

SYMBOL - plot text or symbols

SYNOPSIS

```
CALL SYMBOL (FX, FY, HEIGHT, ICHAR, THETA, NCHARS)
REAL FX
REAL FY
REAL HEIGHT
INTEGER ICHAR
REAL THETA
INTEGER NCHARS
```

FUNCTION

This routine plots either a line of text or a special symbol starting at a specified location.

In order to plot a line of text ICHAR should be the usual name of the character array and NCHARS should contain the number of characters in the line of text. The text will be plotted with the coordinates (FX,FY) determining the bottom, left position of the first character in the string.

In order to plot a single symbol from the set of special characters, ICHAR should contain the symbol number (see following figure) and NCHARS should be -1. In this case if NCHARS=-2 then a line will also be drawn from the previous pen position to the new position.

Note that the first 14 symbols are centered symbols for which the coordinates will define the center of the symbol plot. The rest of the symbols will be drawn with the coordinates defining the bottom, left of the symbol.

INPUT

FX	REAL	new X coordinate
FY	REAL	new Y coordinate
HEIGHT	REAL	height of characters in centimeters
ICHAR	INTEGER	symbol number or text address
THETA	REAL	angle of inclination of characters
NCHARS	INTEGER	number of characters or
	= -1,	draw a single symbol
	= -2,	draw a line to new position and then draw symbol

OUTPUT

none

```
RETURNS

        none
*/
```

```
/*
NAME

        WHERE - find current pen position

SYNOPSIS

        CALL WHERE (FX, FY, FACT)
        REAL FX
        REAL FY
        REAL FACT

FUNCTION

        This routine returns the current X and Y coordinates
        of the pen position and the current scaling factor
        as set by the last call to FACTOR().

INPUT

        none

OUTPUT

        FX      REAL    receives current X coordinate
        FY      REAL    receives current Y coordinate
        FACT    REAL    receives current scale factor

RETURNS

        none
*/
```

4.3 RLSCLIB - RLS C FUNCTION LIBRARY

RLSCLIB provides 'C' language support functions not available in the 'C' runtime library, but required by the RLS system. Two versions of the library are available, one the RSX version in [RLS.LIBRARY.RSX], and the VMS version in [RLS.LIBRARY.VMS]. Wherever possible, the two versions have the same functionality. In some cases, this has not been possible, given the different systems. Users should check the libraries before using. The libraries can be used from FORTRAN programs, using the instructions in the following section. The directory listing for the RLSCLIB library given, is the VMS version.

4.3.1 Fortran Use Of C Language Subroutines

Most of the subroutines supporting RLS have been written in the C language. Those application processes written in Fortran, wishing to call these routines must use the following conventions:

1. declare all C subroutines to be used as EXTERNAL

2. declare the C interface subroutines:

```
REAL*8 CDOUBLE
INTEGER CINT
LOGICAL CLOGICAL
INTEGER*4 CLONG
```

3. to call the C routine, call the appropriate interface subroutine with the first argument being the name of the C routine, and subsequent arguments as documented in the C library manual. Use %REF() and %VAL() for each argument to 'call-by-reference' or 'call-by-value' as necessary. Note that the use of the % symbol in the PDP-11 FORTRAN will cause many warning diagnostics in the output listing, however, the linkage will execute correctly.

```
eg,    CALL CDTOE(%REF(buf),value,%VAL(1),%VAL(3))
```

Note that the double or long argument is not invoked with either %VAL() or %REF().

4. The C FIO data structure should be declared in a FORTRAN program as:

```
INTEGER*2 fio(262)
```

where the longest FIO type in use is the 262 word structure for the VMS C compiler. For example, to call the C fopen() function for a data file:

```
EXTERNAL FOPEN
INTEGER CINT
INTEGER*2 INFIO(262)
CALL CINT(FOPEN,%REF(INFIO),%REF('TEST.DAT'),%VAL(0))
```

4.3.2 Directory Listing Of RLSCLIB Modules

The RLSCLIB library modules are in the following table. All the module names with '*' indicate modules for which there is no documentation. For those files for which there is help information available, the help information follows the directory listing.

RLSCLIB	RLSCLIB(cont'd)
ADDEXT	GETTUPLE
ALOG*	GTTY
ASSIGN	HOST
CDTOE*	INSTUPLE
CDTOF*	IORAL*
CHAIN	IOWAL*
CHK_TYPE	LGE*
CLNFLNAM*	OPENR
CLOSER	POLAR*
CLSEEK*	POLY_LSQ*
CMFNAM*	POLY_NORM*
CMPSTR*	PREFXD*
CNVSTR	PROMPT
CREATR	PUTTUPLE
CRELOG	QUEUE
DASSGN*	READHDR
DELTUPLE	RENAME
DEQUEUE*	RLSUIC*
DRUM*	RLS_PROMPT*
EOFON	RPLTUPLE
FILEXPND	SKIPHDR
FILL*	STTY
FTOC*	SUBPRC*
GETCPD	TRANSLATE
GETFLOAT	TRNLOG
GETINT	WAIT*
GETLOGICAL	WAIT_ABS*
GETPID	WAIT_REL*
GETSTRING	XFRHDR

```
/*
NAME

        addext - add extension

FUNCTION

        This routine adds the given extension to a file name.
*/
```

```
/*
assign                                                      assign

NAME

        assign - assign a device

FUNCTION

        This function assigns the named device to a channel.

SYNOPSIS

        assign (name)
                char *name;

RETURNS

        The channel number assigned.

*/
```

```
/*
chain                                                        chain

NAME

        chain - chain to an external program

SYNOPSIS

        int chain(task)
                char task[];

FUNCTON

        This routine starts up the external program, task,
        and waits for the program to complete execution.

INPUT

        task    the name of the target program

RETURNS

        exit status code of the target program

*/
```

```
/*
chk_type                                                    chk_type

NAME

        chk_type - check type

FUNCTION

        This routine checks a character string for valid type.

SYNOPSIS

        int chk_type (type, value)
                char type[];
                char value[];

INPUT

        type            string specifying type check to be performed
        value           string specifying value to be checked

OUTPUT

        none

RETURNS

        TRUE            if value is of correct type
        FALSE           if value is not correct
*/
```

```
/*
NAME

        closer - close relation

FUNCTION

        This routine closes the specified relation.
*/
```

/*
NAME

CNVSTR - convert string

SYNOPSIS

```
int cnvstr(output, format)
        char output[], format[];
```

FUNCTION

cnvstr() converts a FORMAT string to an OUTPUT string. This is primarily a copy of the FORMAT string to the OUTPUT but also handles non-printing characters. A non-printing character can be output by entering them enclosed in angle brackets eg.<>. Several non-printing characters can be handled by separating them with commas.

The character set is:

```
"NUL",   0,
"SOH",   1,
"STX",   2,
"ETX",   3,
"EOT",   4,
"ENQ",   5,
"ACK",   6,
"BEL",   7,
"BS",  010,
"HT",  011,
"LF",  012,
"VT",  013,
"FF",  014,
"CR",  015,
"SO",  016,
"SI",  017,
"DLE", 020,
"DC1", 021,
"DC2", 022,
"DC3", 023,
"DC4", 024,
"NAK", 025,
"SYN", 026,
"ETB", 027,
"CAN", 030,
"EM",  031,
"SUB", 032,
"ESC", 033,
"FS",  034,
"GS",  035,
"RS",  036,
"US",  037,
"DEL", 377,
                    "^A",     1,
                    "^B",     2,
                    "^C",     3,
                    "^D",     4,
                    "^E",     5,
                    "^F",     6,
                    "^G",     7,
                    "^H",   010,
                    "^I",   011,
                    "^J",   012,
                    "^K",   013,
                    "^L",   014,
                    "^M",   015,
                    "^N",   016,
                    "^O",   017,
                    "^P",   020,
                    "^Q",   021,
                    "^R",   022,
                    "^S",   023,
                    "^T",   024,
                    "^U",   025,
                    "^V",   026,
                    "^W",   027,
                    "^X",   030,
                    "^Y",   031,
                    "^Z",   032,

*/
```

```
/*
NAME

        creatr - create relation

FUNCTION

        This function creates a new relation for writing.
*/
```

```
/*
NAME

        CRELOG - create logical name

SYNOPSIS

        crelog (logical, physical)
                char *logical;
                char *physical;

FUNCTION

        This routine creates an assignment of the
        physical name in the NULL terminated string
        at *physical to the logical name given
        by the NULL terminated string at *logical.

INPUT

        logical         char *  logical name
        physical        char *  physical name

OUTPUT

        none

RETURNS

        none
*/
```

```
/*
NAME

        deltuple - delete tuple

FUNCTION

        This function deletes a tuple from a relation. All tuples
        matching the specified key are deleted.

*/
```

```
/*
eofon                                                    eofon

NAME

        eofon - check for EOF on file

SYNOPSIS

        int eofon(filfio)
                FIO *filfio;

FUNCTION

        This routine checks to see if there is an EOF condition
        on the file.

INPUT

        filfio  pointer to the file fio

OUTPUT

        none

RETURNS

        TRUE if file is at EOF
        FALSE otherwise

*/
```

```
/*
NAME

        FILEXPND - file name expand

SYNOPSIS

        LISTELT *filexpnd(filename)
                char *filename;

FUNCTION

        To expand a file name with wild card characters into a list
        of file names. A pointer to the list is returned.
        A list element is defined as two longwords. The first is the
        pointer to the next element. The second points
        to a character string.

INPUT

        filename        character string (null terminated) containing the
                        filename to expand.

OUTPUT

        none

RETURNS

        Success         pointer to a list of file names.
        Failure         a null pointer is returned (0).

NOTE

        The list elements and the strings they point to are dynamically
        allocated by the C storage allocator.

*/
```

```
/*
getcpd                                                  getcpd

NAME

        getcpd - get command definition sequence

FUNCTION

        This routine reads in the command parameter definitions.

*/
```

```
/*
NAME

        GETFLOAT - get floating point number (double)

SYNOPSIS

        double getfloat (filfio)
                FIO *filfio;

FUNCTION

        This routine scans the input stream of the specified
        file for the next valid floating point number.

INPUT

        filfio  a pointer to the FIO of the file

OUTPUT

        none

RETURNS

        the double floating point value of the
        next number in the stream

*/
```

```
/*
NAME

        GETINT - get integer number

SYNOPSIS

        int getint(filfio)
                FIO *filfio;

FUNCTION

        This routine scans the input stream from the specified
        file for the next valid integer number.

INPUT

        filfio  a pointer to the FIO of the file

OUTPUT

        none

RETURNS

        the integer value of the
        next number in the stream
*/
```

```
/*
NAME

        GETLOGICAL - get logical variable

SYNOPSIS

        int getlogical(filfio)
                FIO *filfio;

FUNCTION

        This routine scans the input stream from the specified
        file for the next valid logical variable.

INPUT

        filfio  a pointer to the FIO of the file

OUTPUT

        none

RETURNS

        the logical value of the
        next variable in the stream
*/
```

/*
getnewline

NAME

getnewline - get new line

SYNOPSIS

```
int getnewline(filfio)
        FIO *filfio;
```

FUNCTION

This function scans the input stream for the file controlled by filfio for the next newline character.

INPUT

filfio	a pointer to the FIO of the file

OUTPUT

none

RETURNS

*/

```
/*
getoctal                                                    getoctal

NAME

        getoctal - get octal number

SYNOPSIS

        int getoctal(filfio)
                FIO *filfio;

FUNCTION

        This routine scans the input stream from the specified
        file for the next valid octal number.

INPUT

        filfio  a pointer to the FIO of the file

OUTPUT

        none

RETURNS

        the octal value of the
        next number in the stream
*/
```

```
/*
NAME

        GETPID - get process identification number

FUNCTION

        Gets the processes id number via the getjpi system service.

INPUT

        none.

OUTPUT

        none.

RETURNS

        Returns longword containing the process id. number.

*/
```

```
/*
NAME

        GETSTRING - get string of characters

SYNOPSIS

        int getstring(filfio, str, nchars)
                FIO *filfio;
                char *str;
                int nchars;

FUNCTION

        This routine reads in a text string from the specified input file.
        If the string is delimited by quote marks, '"', then whitespace
        inside the quotes is also read in.
        Otherwise only the next symbol on the file is read in.

INPUT

        filfio  the FIO pointer for the file
        str     pointer to where the string is to be stored
        nchars  number of characters in the str buffer

OUTPUT

        str     this string receives the character string read in

RETURNS

        the length of the string

*/
```

```
/*
NAME

        gettuple - get a tuple

FUNCTION

        This function returns the next tuple from a relation
        that matches a specified key.

*/
```

```
/*
NAME

        GTTY - get terminal status

SYNOPSIS

        gtty (fd, buf)
                FILE fd;
                TTY_STATUS *buf;

FUNCTION

        GTTY() returns information on the current status
        of the terminal associated with file pointer FD
        into the TTY_STATUS record pointed at by BUF.
        A record of type TTY_STATUS is defined as:

        typedef {
                char _raw;
                char _wrap;
                char _escape;
                } TTY_STATUS;

        where:

                _raw    = TRUE for raw char I/O
                _wrap   = TRUE for computer generated wrap around
                _escape = TRUE for escape sequence generation
RETURNS

        none

*/
```

```
/*
NAME

        HOST - execute a host system command

SYNOPSIS

        int host(cmdlin)
                char cmdlin[];

FUNCTON

        This routine submits a command line to the host system,
        and waits for execution to complete.

INPUT

        cmdlin          command for the host system

RETURNS

        exit status code of the target program

*/
```

```
/*
NAME

        instuple - insert a tuple

FUNCTION

        This function inserts a new tuple into a relation.
        The new record is sorted into the relation.
*/
```

```
/*
NAME

        openr - open relation

FUNCTION

        This function opens an existing relation for reading.
*/
```

```
/*
NAME

        PROMPT - prompt user

SYNOPSIS

        int prompt (keyword, def, resp)
        char *keyword;
        char *def;
        char resp[];

FUNCTION

        This routine prompts the user to enter the value
        for a keyword. If the user replies with a <RETURN>
        (ie. no value) then this routine returns the
        default string specified by the calling routine.

INPUT

        keyword          keyword identifier
        def              default string

OUTPUT

        resp             response string to be returned

RETURNS

        TRUE    if user entered a value
        FALSE   otherwise

*/
```

```
/*
NAME

        puttuple - put tuple

FUNCTION

        This function writes a new tuple to a relation. The write
        is performed sequentially to the next record.
*/
```

```
/*
NAME

        QUEUE - queue a file to a spooler

SYNOPSIS

        queue(filnam, quename)
                char * filnam;
                char *quename;

FUNCTION

        To queue the file opened and connected on the rab pointed to by
        file.  The associated fab must have a nam block.  This is all set up
        by the RMS C interface.
        The file is queued to queue <quename>.  The delete modifier is used
        so that the file is deleted when the symbiont has dealt with it.
        The reply of the symbiont manager is returned, so that errors
        may be checked for.

        The file is closed before it is queued, so that no symbiont errors
        will occur.

RETURNS

*/
```

```
/*
readhdr

NAME

        readhdr - read RLS header

SYNOPSIS

        int readhdr (usrfio, inpbuf, maxchrs)
                FIO *usrfio;
                char inpbuf[];
                int maxchrs;

FUNCTION

        This routine reads an RLS header into a buffer.
        Lines of user input are accepted until a line without
        a terminating '-' character is found.
        A NULL is stored after the last character.

INPUT

        usrfio  the FIO for the input device

OUTPUT

        inpbuf  the character buffer filled with user input
        maxchrs the size of the input buffer

RETURNS

        int     the number of characters put into the buffer

*/
```

```
/*
NAME

        rename - rename a file.

SYNOPSIS

        BOOL rename (fname, tname)
                TEXT *fname, *tname;

FUNCTION

        rename() renames the file from fname to tname.

RETURNS

        TRUE    if ok
        FALSE   if not ok

EXAMPLE

        if (!rename ("templ.c", "temp2.c"))
                errfmt ("can't rename templ.c\n");
*/
```

```
/*
NAME

        rpltuple - replace tuple

FUNCTION

        This function replaces tuples in a relation with new
        tuples. All tuples matching the specified key are
        replaced. Only specific fields from the tuple are
        modified. All NULL fields are left alone.
*/
```

```
/*
skiphdr                                                          skiphdr

NAME

        skiphdr - skip header

SYNOPSIS

        skiphdr(filfio)
                FIO *filfio;

FUNCTION

        This routine skips over the 'RLS header' at the front of
        a data file.

INPUT

        filfio  pointer to the FIO of the file

OUTPUT

        none

RETURNS

        none

*/
```

```
/*
NAME

        STTY - set terminal status

SYNOPSIS

        stty (fd, buf)
                FILE fd;
                TTY_STATUS *buf;

FUNCTION

        STTY() sets the status of the terminal associated
        with file pointer FD to the values specified by
        TTY_STATUS record pointed at by BUF. A TTY_STATUS
        record is defined as:

        typedef {
                char _raw;
                char _wrap;
                char _escape;
                } TTY_STATUS;

        where:

                _raw    = TRUE raw char I/O
                _wrap   = TRUE computer generate wrap around
                _escape = TRUE recognize escape sequences

RETURNS

        none

*/
```

```
/*
NAME

        TRANSLATE - translate logical name

SYNOPSIS

        translate (name, physname, size)
                char *name;
                char *physname;
                int size;

FUNCTION

        This routine searches for the logical name specified
        by NAME and returns the physical device name actually
        assigned in the string PHYSNAME.

RETURNS

        number of chars placed into *physname

*/
```

```
/*
NAME

        TRNLOG - translate logical name

SYNOPSIS

        trnlog (name, physname, size)
                char *name;
                char *physname;
                int size;

FUNCTION

        This routine searches for and returns the physical
        name assigned to a logical name.

INPUT

        name            char *  logical name
        size            int     max number of chars to place
                                into *physname

OUTPUT

        physname        char *  address of string to receive
                                physical name

RETURNS

        number of chars placed into *physname

*/
```

```
/*
NAME

        XFRHDR - transfer rls header

SYNOPSIS

        xfrhdr(infio, outfio)
                FIO *infio;
                FIO *outfio;

FUNCTION

        This function transfers an RLS header from one file to another.

INPUT

        infio   pointer to the FIO of the input file
        outfio  pointer to the FIO of the output file

OUTPUT

        none

RETURNS

        none

*/
```

4.4 MATH

The Math library consists of the library support routines for the statistical analysis and signal processing sections of RLS. It is built as part of the math section, without user intervention. During the build of the signal processing, some modules are added. It is important, though, that the math subdirectory be built before the signal processing whenever a new RLS generation is being done, to insure that the task builds are being done with the most up-to-date modules.

4.5 PLOTDEV

The PLOTDEV library is the device support library for the graphics devices in the RLS system. Some of the devices supported are the tektronics 4014, tek4662, vt100, vt125, lxy, and benson plotters. As this library is generated during the course of the xyplot generation, it is always the current library being used for the generation. This is necessary, since some of the software is site specific, and therefore must be replaced for each generation.

4.6 TARGETLIB

TARGETLIB is the library required for the device support for the data acquisition for the lpa and lps. Again, modules are added to the library at the time of the build for the RLS device generation. The Chemistry system does not rely on any external libraries, so during the build of the [.device] subdirectory, the necessary modules are automatically replaced.

For the Clover Bar system, this is not the case. Whenever there has been a new sysgen, it is necessary to update the target library to include the most current lpa object modules. Lpa.obj can be copied from the Clover Bar system, after the sysgen, and can be used to replace the existing version.

CHAPTER 5

RLS COMMAND LIBRARY

5.1 RLS ORGANIZATION

In the User's Manual, the RLS 'commands' have been described under the different application sections, rather than in alphabetical order. This organization is strictly for documentation purposes, as most experiments will utilize modules from different sections. An example of this would be the GC analysis. For a typical run, programs from the device control, the math, the GC and the graphics sections could all be used. All of the modules are completely compatible, in that they can read and write files that can be used by all other applications within the RLS system. In the remainder of this chapter, the commands available, the programs which execute the commands, and any support modules, are identified.

5.1.1 RLS Command Language

RLS has in its command language a number of commands used for system maintenance and modification. When used in the RLS area, they can modify or tailor the system to suit the site requirements. When used in the user's area, they can further modify the RLS system to suit an individual's needs. The general commands are:

1. ADD COMMAND - adds a command to the command catalog.
2. DELETE COMMAND - deletes a command from the command catalog.
3. CREATE DEFINITION - creates a command definition file.
4. SET 'command keyword value' - changes defaults in the command definition file.

The remaining RLS commands are used either for device control or for different data analysis. The different applications are documented in the RLS User's Manual.

VERB	SET	SHOW	SHOW
OBJECT	(IDENTIFIER) - (VERB OBJECT KEYWORD)		COMMANDS

VERB	ADD	DELETE	CREATE
OBJECT	COMMANDS		DEFINITION
		- (VERB OBJECT KEYWORD)	
KEYWORDS	CONTEXT IDENTIFIER PROGRAM-FILE KEYWORD-FILE PROGRAM	CONTEXT IDENTIFIER	

COMMAND/DESCRIPTION	PROGRAM
ADD COMMAND	ADDCOMM.*
CREATE DEFINITION	CRECDF.*
DELETE COMMAND	DELCOMM.*
HELP 'command'	RLSHLP.*
RLS command language interpreter	RLSCLI.*
SET 'command'	RLSSET.*
SHOW 'identifier'	RLSSHO.*
SHOW COMMANDS	SHOCOMM.*
ADD COUNTER	CTRADD.*
DECREMENT COUNTER	CTRDECR.*
DELETE COUNTER	CTRDELETE.*
SHOW COUNTERS	CTRSHOW.*

NOTE: The following modules are required during the operation of the RLS command line interpreter:

MODULE	FUNCTION
ASK.*	return TRUE/FALSE value from a user
GETCPD.*	read command definition
OPNCDF.*	open command definition file
PRCNAM.*	determine if a program is running as an RLS subprocess
PROMPT.*	perform RLS prompting operations
READHDR.*	read an RLS command line

5.1.2 RLS Devices

VERB	PULSE	READ	WAIT	WRITE	MONITOR	SHOW	START	STOP	WAIT	WAIT	READ	RING	WAIT	WRITE
OBJECT	DIGITAL				ANALOG					CLOCK	TERMINAL			
KEYWORDS	POINT DURATION	POINT WIDTH	POINT	POINT WIDTH VALUE	CHANNEL THROUGH FREQUENCY DURATION POINT GAIN SIGNAL-FILE DESCRIPTION	CHANNEL	CHANNEL THROUGH FREQUENCY DURATION POINT GAIN SIGNAL-FILE DESCRIPTION	CHANNEL	CHANNEL	DURATION TIME	PROMPT REPLY DEFAULT	(NONE)	(NONE)	MESSAGE

COMMAND	PROGRAM
READ TERMINAL	TRMREAD.*
RING TERMINAL	TRMRING.*
WAIT TERMINAL	TRMWAIT.*
WRITE TERMINAL	TRMWRITE.*
PULSE DIGITAL	DIGPULSE.*
READ DIGITAL	DIGREAD.*
WAIT DIGITAL	DIGWAIT.*
WRITE DIGITAL	DIGWRITE.*
MONITOR ANALOG	ANAMONITR.*
SHOW ANALOG	ANASHOW.*
START ANALOG	ANASTART.*
STOP ANALOG	ANASTOP.*
WAIT ANALOG	ANAWAIT.*
WAIT CLOCK	CLKWAIT.*

NOTE: The modules listed below are required for the execution of the above RLS commands:

MODULE	FUNCTION
BDDRV.*	DR-11K (BD:) device driver for the ALSARC PDP-11/34
CDIGIO.*	C digital input/output subroutines.
LPACM.*	the analog channel start sampling program
LPAEMU.*	subroutines to interface the LPARDA program to either the LPA-11 device or to the LPS-11 device.
LPARDA.*	Data acquisition program which interfaces to the drivers.
LSDRV.*	modified LPS-11 (LS:) driver used on CHMARC PDP-11/24
WAIT.*	subroutine to put a program into a sleep state for a specified duration.

5.1.3 GC Applications

VERB	Zero	FIND	ANALYZE	FIND	PREPARE
OBJECT	BASELINE	CHN-PEAKS	PEAKS		REPORT
KEYWORDS	PEAK FILE	SIGNAL-FILE	PEAK-FILE	SIGNAL-FILE	PEAK-FILE
		PEAK-FILE	CALIB.NAME	PEAK-FILE	OUTPUT-FILE
		STEPSIZE	SMP-AMT	GATE	FORMAT
			STD-AMT	SLOPE	METHOD
			REF-RTW	DENSITY	TITLE
			ID-LVL	WIDTH	SUBTITLE
			% RTW	FREQUENCY	UNITS
			RF-UNK		SUP-UNK
			DVT		TIME
			DIL-FTR		STD-AMT
					SMP-AMT
					MISC

ADD	CHANGE	DELETE	ENTER	EXTRACT	SHOW	UPDATE	ADD
STANDARD							TICS
CALIB.NAME	CALIB.NAME	CALIB.NAME	CALIB.NAME	X-Y NAME	OUTPUT-FILE	CALIB.NAME	PEAK-FILE
PEAK	PEAK	PEAK	PEAK-FILE	CALIB.NAME	CALIB.NAME	PEAK-FILE	GRAPH-NAME
AMOUNT	AMOUNT	AMOUNT		PEAK		% WINDOW	
						RT-FTR	
						SIZE-FTR	

COMMAND	PROGRAM
ADD STANDARD	CALIBINS.*
ADD TICS	TICKS.*
ANALYZE PEAKS	POSTRUN.*
CHANGE STANDARD	CALIBCHG.*
CONVERT LPS-FILE	DABFILE.*
DELETE STANDARD	CALIBDEL.*
ENTER STANDARD	CALIBENT.* CALIBRATE.*(obsolete)
EXTRACT STANDARD	CALIBEXTR.*
FIND CHN-PEAKS	CONCRTCHN.*
FIND PEAKS	LPSPKP.*
GENERATE PEAKS	GENDAT.*
PREPARE REPORT	GCREPORT.*
SHOW STANDARD	CALIBSHOW.*
UPDATE STANDARD	CALIBUPD.*
ZERO BASELINE	BASCOR.*

NOTE: The following routines are required by the programs listed above:

MODULE	FUNCTION
GETTUPLE.*	data file access routine
NEWPROMPT.*	RLS standard prompt subroutine
REL.*	data file access routine
RPLTUPLE.*	data file access routine

VERB				
PREPARE	SEND	DELETE	SHOW	PREPARE
OBJECT				
PLOT				~~PLOT~~ TEXT
KEYWORDS				
X-Y.NAME	GRAPH.NAME	GRAPH-NAME	GRAPH.NAME	INPUT-FILE
GRAPH.NAME	DEVICE		OUTPUT-FILE	GRAPH.NAME
AXIS	FACTOR			HEIGHT
TITLE				PEN
DESCRIPTION				
X LABEL				
Y LABEL				
UP				
AUTO				
X MIN				
X MAX				
Y MIN				
Y MAX				
SYMBOL				
PEN				
FACTOR				
STEPSIZE				
POINTS				
X-TRANSFORM				
Y-TRANSFORM				

COMMAND	PROGRAM
DELETE PLOT	GRAPHDEL.*
PREPARE PLOT	PLOT.*
PREPARE TEXT	TEXT.*
SEND PLOT	PLOTTR.*
SHOW PLOTS	GRAPHSHOW.*

NOTE: The following routines are required during the execution of the above commands:

MODULE	FUNCTION
CLIP.*	performs clipping to graph boundaries
SYMSET.*	produces figure of current symbol set

5.1.5 Math Routines

The math routines require that the dataset named be in the xydata directory. At the present time, there is no way to enter an existing dataset into the directory except through the use of an editor. To get a description of how to add the data file name to the directory, refer to Appendix A, to the section dealing with the 'xydataset directory'.

VERB	ADD	DELETE	SHOW	CONVERT	GENERATE	FIT	GENERATE	SHOW
OBJECT	FUNCTION			UNITS	CURVE		RESIDUALS	~~FITS~~ STATISTICS
KEYWORDS	COEFFICIENT.NAME	COEFFICIENT.NAME FORM	COEFFICIENT.NAME FORM OUTPUT-FILE	SIGNAL.FILE NEWX NEWY XUNITS YUNITS STEPSIZE	COEFFICIENT.NAME FORM X-Y.NAME XMIN YMIN npoints	COEFFICIENT.NAME FORM X-Y.NAME	COEFFICIENT.NAME FORM X-Y.NAME RESIDUAL.NAME VERSUS	COEFFICIENT.NAME FORM X-Y.NAME OUTPUT-FILE FULL

VERB				BUNCH
OBJECT	AMPLITUDE SPECTRUM	AUTO CORRELATION	BACKWARD FFT	DATA
KEYWORDS	input-file output-file nyquist npoints stepsize	input-file output-file nyquist npoints stepsize	input-file output-file npoints stepsize	signal-file average stepsize

VERB	EXTRACT	EXTRACT		GENERATE
OBJECT	DATA	RESIDUALS	FORWARD FFT	RESIDUALS
KEYWORDS	input-file output-file xmin xmax stepsize	coefficient-name form x-y.name residual.name versus	input-file output-file normalize npoints stepsize	coefficient-name form x-y.name residual.name versus

VERB	GENERATE	GENERATE	MAKE		
OBJECT	INTEGRAL	NYQUIST	EQUIDISTANT	PHASE SPECTRUM	POWER SPECTRUM
KEYWORDS	input-file output-file stepsize	input-file output-file npoints stepsize	x-y.name	input-file output-file continuous nyquist npoints stepsize	input-file output-file nyquist npoints stepsize

VERB	SHOW	REMOVE	REMOVE
OBJECT	NPOINTS	MEAN	TREND
KEYWORDS	input-file	input-file stepsize	input-file output-file window width npoints

COMMAND	PROGRAM
ADD FUNCTION	FNADD.*
AMPLITUDE SPECTRUM	AMPLITUDE.*
AUTO CORRELATION	AUTOCORR.*
BACKWARD FFT	FFTB.*
BUNCH DATA	BUNCH.*
CONVERT EPR-FILE	CNVEPR.*
CONVERT UNITS	CONVERT.*
DELETE FUNCTION	FNDEL.*
EXTRACT DATA	EXTRACT.*
EXTRACT RESIDUALS	RESIDUAL.*
FIT CURVE	RLSLSQFIT.*
FORWARD FFT	FFTF.*
GENERATE CURVE	RLSEVAL.*
GENERATE DERIVATIVE	DERIVATIV.*
GENERATE INTEGRAL	INTEGRAL.*
GENERATE NYQUIST	NYQUIST.*
GENERATE RESIDUALS	RESIDUAL.*
MAKE EQUIDISTANT	EQUI.*
PHASE SPECTRUM	PHASE.*
POWER SPECTRUM	POWER.*
SHOW NPOINTS	NPOINTS.*
SHOW STATISTICS	RLSANOVA.*
REMOVE MEAN	RMVMEAN.*
REMOVE TREND	RMVTREND.*

NOTE: The following support routines are inserted in the math library when the [.math] and [.signal] directories are built, and are support routines for the above commands.

MODULE	FUNCTION
BASIS.*	generates basis vectors
CFFTB.FOR	
CFFTF.FOR	
CFFTI.FOR	
DOTPR.*	vector dot product
DRUM.C	
EVALUATE.*	evaluate function
EXP.*	exponential function
FFTB.MAC	
FFTF.MAC	
FFTI.MAC	
FNCTYPE.*	encode function type
LOG.*	log base 10 function
LSQFIT.*	least squares curve fit
MATINV.*	matrix inverse
ML1DECOMP.*	matrix LU decomposition
ML1NORM.*	generate normal equations
ML1SOLVE.*	solve linear system
MMUL.*	matrix multiply
MTRANS.*	matrix transpose
NORM.*	generate normal equations
POLAR.C	
PWR.*	exponential operator
SVEMG.*	sum of vector element magnitudes
UXFCOEFFS.*	untransform function coefficients
VPOLY.*	evaluate polynomial
VSMUL.*	multiply vector elements by scalar
VSWAP.*	swap to vectors
XFCOEFFS.*	transform function coeffiecients
XFDATA.*	transform data vectors

AUXILIARY FILES:

MATHLIB.H	C symbol definition file

VERB

ADD	CHANGE	DELETE	SHOW

OBJECT

DATASET

KEYWORDS

X-Y.NAME	X-Y.NAME	X-Y.NAME	X-Y.NAME OUTPUT-FILE

COMMAND	PROGRAM
ADD DATASET	XYADD.*
CHANGE DATASET	XYCHANGE.*
DELETE DATASET	XYDEL.*
SHOW DATASET	XYSHOW.*

5.1.7 RLS Simulated Distillation

VERB	$\boxed{\text{CONVERT}}$	$\boxed{\text{CORRECT}}$	$\boxed{\text{PREPARE}}$	$\boxed{\text{REMOVE}}$	$\boxed{\text{REPORT}}$	$\boxed{\text{SLICE}}$	$\boxed{\text{WRITE}}$
OBJECT	$\boxed{\text{TIME-TO-TEMP}}$	$\boxed{\text{BASELINE}}$	$\boxed{\text{CALIBRATION}}$	$\boxed{\text{ADDITIVE}}$	$\boxed{\text{D2887}}$	$\boxed{\text{PEAK-AREA}}$	$\boxed{\text{RLS-FILE}}$
KEYWORDS	pcarea-file temp-file calib-file	rsimdis-file blank-file area-file	xyfile calib-file title units	raw-file rsimdis-file start stop	temp-file output-file lreport title subtitle	area-file pcarea-file stepsize title subtitle rtime	gc-file raw-file subtitle

COMMAND	PROGRAM
CONVERT TIME-TO-TEMP	SCALE.*
CORRECT BASELINE	SIMBAS.*
PREPARE CALIBRATION	CALIBSIM.*
REMOVE ADDITIVE	REMOVE.*
REPORT D2887	REPORT.*
SLICE PEAK-AREA	PCAREA.*
WRITE RLS-FILE	WRTFILE.*

CHAPTER 6

SYSTEM NOTES

RLS Program development has been done on the VAX/VMS system, and as a result, all archival and system documentation is on the VAX. With a system as large as the current RLS system, it has become necessary to catalog the files/programs in different logical areas. This has been done through the use of VAX/VMS subdirectories. The first part of the chapter describes the RLS directory organization. Documentation standards and generation are discussed briefly, with the assumption that the user is able to use DEC RUNOFFF. A complete description of the task generation is given, although attempts have been made to simplify the process through the use of indirect command procedures. System peculiarities which have had a minor impact on RLS development are mentioned. The very last section describes how to get a user started on either the VMS or RSX systems.

6.1 PROGRAM DEVELOPMENT DIRECTORIES

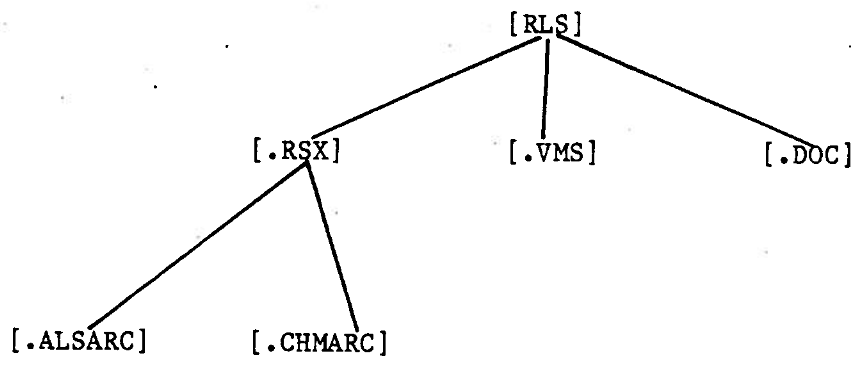
Logically, all the files in the RLS directories fall into one of three catagories - the run areas, the source development areas or the system support or library areas. A number of subdirectories have been created in the RLS directory to accommodate this cataloguing of files.

The three different catagories are described in detail in the following sections.

6.1.1 Run Areas

The run areas have all the executable tasks. All general tasks are located in the [.VMS] or [.RSX] areas. The only exceptions to this are the 'help' and 'command definition files', all of which are stored in the main [RLS] directory. Any site or system specific modules will be located in a directory designated with the site or system name.

RUN AREAS

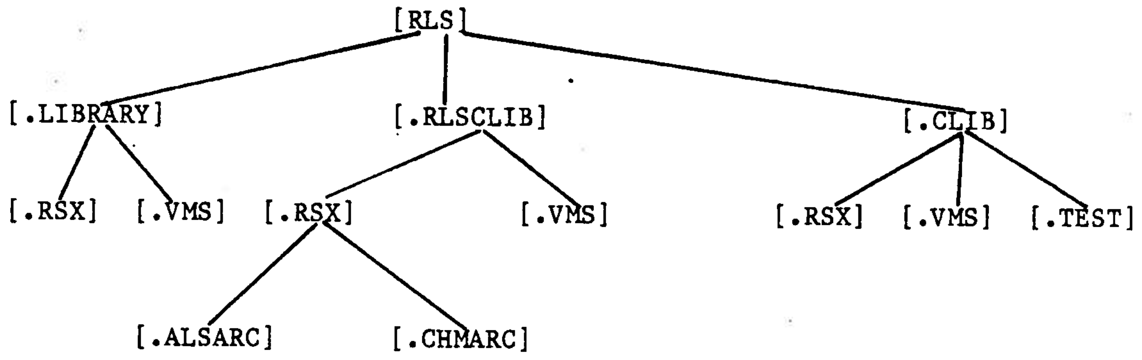


DIRECTORY	TARGET SYSTEM	MODULES
[RLS]	- ALL SYSTEMS (SYSTEM INDEPENDENT)	*.COM, *.RLS *.HLP
[.RSX]	-ALL RSX SYSTEMS	*.EXE
[.VMS]	- VAX SYSTEM DEPENDENT	*.EXE
[.DOC]	- ALL SYSTEMS	*.RNO, *.MEM
[.ALSARC]	- CLOVER BAR SITE DEPENDENT	*.EXE, *.TSK
[.CHMARC]	- CHEMISTRY SITE DEPENDENT	*.EXE, *.TSK

6.1.2 Library Areas

The libraries required for RLS support are in three subdirectories - [.LIBRARY], [.RLSCLIB], and [.CLIB]. Again, site specific tasks are found in the site specific subdirectories within these directories.

LIBRARY AREAS

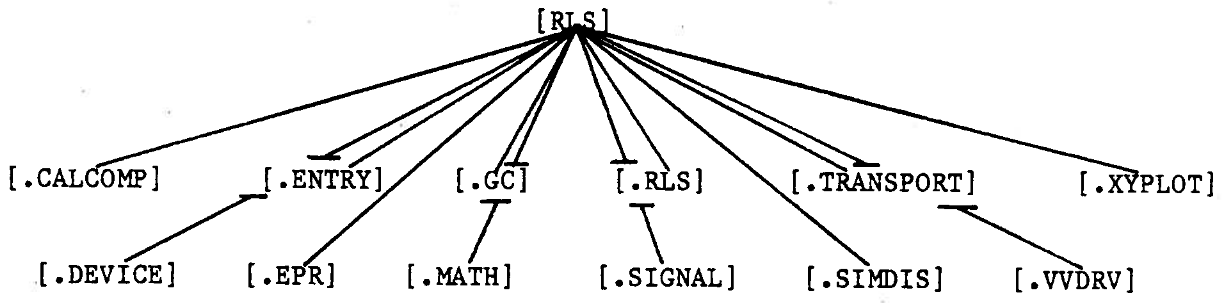


DIRECTORY	TARGET SYSTEM	MODULES
[.LIBRARY]	-ALL SYSTEMS (COMPILE TIME LIBRARY)	*.H, STD.H SYS.H(RSX,VMS)
[.RLSCLIB]	-ALL SYSTEMS (SOURCE)	*.C,*.FOR,*.MAC
[.CLIB]	-ALL SYSTEMS (C LIBRARY)	*.C,*.FOR,*.MAC

6.1.3 Source Development Areas

All of the applications within the RLS system are located in a subdirectory with the application name. Site or system specific modules are located in appropriately marked subdirectories, within the application directory.

SOURCE DEVELOPMENT AREAS



DIRECTORY	APPLICATION	MODULES
[.CALCOMP]	-CALCOMP PLOTTING	*.FOR,*.C
[.DEVICE]	-DEVICE CONTROL	*.FOR,*.C,*.MAC
[.ENTRY]	-DATA ENTRY	*.FOR,*.C
[.EPR]*	-EPR SPECTROMETER	*.FOR,*.CMD
[.GC]	-GAS CHROMATOGRAPHY	*.FOR,*.C
[.MATH]	-MATH ROUTINES	*.C
[.RLS]	-RLS CONTROL MODULES	*.C
[.SIGNAL]*	-SIGNAL PROCESSING	*.C,*.FOR,*.MAC
[.SIMDIS]	-D2887 SIMDIS	*.C
[.TRANSPORT]*	-TRANSPORTATION	*.C
[.VVDRV]*	-VIRTUAL DISK	*.COM,*.MAC
[.XYPLOT]	-PLOTTING	*.C

* - undocumented development work.

6.2 DOCUMENTATION

The RLS documentation has been divided into two volumes. Both documents are located on-line, in the [RLS.DOC] area. The two files are written using DEC standard runoff, and all or parts of the documents can be regenerated as required.

Volume I, The User's Manual, describes the end user applications. Each chapter or major application of this document has a separate runoff file, describing that application. The runoff files are named after their application subdirectories. The different chapters are included in the main RLS document using the '.REQUIRE' feature of runoff.

The format for each of these chapters is similar. The initial section is text, discussing the theory, followed by the program documentation. Program documentation is maintained in the source code at the front of the program, to facilitate documentation updates when there are program modifications. A <FF> at the end of the flag page allows a program, called MAKHLP to strip off the documentation page and put it into a '.hlp' file. These 'help' files are included in the application chapters with the '.REQUIRE' command.

The second volume, the Programmer's Manual is intended for those users who wish to maintain or extend the capabilities of the RLS system. Unlike the first volume, the text of the Programmer's Manual is completely contained within the one runoff file. Only the appendices are to be included during the runoff procedure.

The '.mem' files should be created on the VAX and printed on the qume printer. Before printing, the print wheel should be changed to the 'prestige elite' wheel, to obtain the correct character set.

6.3 BUILDING RLS FROM SOURCES

Building RLS from sources is done on the VAX/VMS system. As there are a number of different target systems, all requiring specific compilers, libraries, and task builders, a number of logical assignments must be made to insure that the correct task building software is being used during RLS generation. Once the proper assignments have been made, the support libraries should be rebuilt, after which the applications can be generated.

Because the VAX system does not require that executables are contiguous, it may be necessary to re-copy the files once they are on the RSX system, to force this. Also, creating RSX libraries is often a problem on the VAX, as there is often not enough contiguous disk space. If this happens, an open error will occur on the library file. To get around this problem, library files should be created with '/size=blocks:1.' It is not necessary for the library to be contiguous, for the task build to be successful.

Some of the RSX libraries which must be moved to the VAX system, after a system generation are: FCSRES, SYSLIB, and the lpa object modules. The executive libraries for the RSX systems are no longer being used, since the drivers for the lps and lpa are now being built as part of the sysgen.

To simplify the build procedure, a number of indirect command files have been developed. A control command procedure accepts the target name before setting up the appropriate assignments. RLS 'legal' target names are:

TARGET NAME	SITE/SYSTEM
RSX	SYSTEM NAME
VMS	SYSTEM NAME
ALSARC	SITE NAME
CMPARC	SITE NAME
CHMARC	SITE NAME

System names are those which build software which can run on any of the systems with that operating systems. Site names are used when software is being built which is intended to run on only that machine. Once the general control procedure has completed, the build file located in the application area directory can be used to build the application software within that subdirectory, and any of it's subdirectories.

Due to some qirks in the RLS development, some of the directories must be built before other directories, as library modules required for some of the applications are inserted or replaced in the course of another application build. The correct build sequence is:

1. Library areas:

    1. [rls.rlsclib]

    2. [rls.clib]

    3. [rls.relation]

    4. [rls.calcomp]

2. Applications dependent on other directory builds require that the following build order be observed:

    1. [rls.math]

    2. [rls.rls]

    3. [rls.device]

    4. [rls.xyplot]

    5. [rls.signal]

3. The remaining applications only require that the RLSCLIB library has been regenerated:

    1. [rls.gc]

    2. [rls.entry]

    3. [rls.epr]

    4. [rls.simdis]

The general build procedure is as follows:

1. Set up target assignments

2. Build appropriate libraries

3. Generate executable files

4. Generate help files

At the completion of the system build, the only files which should be left in the source area are the source programs (one version) and the build command files. All other files should either be moved to other areas, or deleted.

6.3.1 Setting Up Target System Assignments

The logical assigments for correct libraries, compilers and task builders are made in the main RLS area, using the indirect command file DEVELOP.COM. At the start of this command file, a number of commands are defined, allowing commands within the build files to invoke either the VAX or RSX version of the C or FORTRAN compilers. Logical RLS required disk assignments are made before the procedure skips through to the correct target name, at which point it makes a number of logical assignments to specified libraries and task builders. Also, during this part of the procedure, the target definitions for conditional compiles are made. For the RSX task build, a foreign command is set up, as defined by the file RSXLINK.COM. This was necessary to select correct task builder switches for the RSX system. For the listings of the indirect command discussed here, refer to Appendix B.

When DEVELOP.COM is executed, a number of messages regarding previous logical assignments will be written to the terminal, as indicated below:

```
==============================================
$ @develop 'targetname'<cr>
Previous logical assignment replaced
        .
        .
$
==============================================
```

Once DEVELOP.COM completes, the RLS system generation can proceed.

6.3.2 Building The RLS Libraries

RLS libraries are rebuilt at the start of the generation, replacing all the site specific modules and including all the current versions of the library modules.

Building RLSCLIB requires that the user access three different build files, in the following order:

[RLS.RLSLCIB]BUILD.COM	-creates and inserts general modules
[RLS.RELATION]BUILD.COM	-inserts all the pseudo database modules
[RLS.CLIB]BUILD.COM	-inserts some CLIB I/O modules which replace those in the Whitesmith C library

The CALCOMP library, PLOTLIB, is built from the [RLS.CALCOMP] directory. All the routines in this library are general routines. Site specific modules are replaced as part of the graphics generation.

The remaining libraries, PLOTDEV, MATH, and TARGETLIB are all built during the appropriate applications generation, since they have either site specific or system specific modules which must be replaced with each generation. As this is done automatically by the build files, there are no additional instructions required on the part of the user.

6.3.3 Generating Executable Tasks

Generation of common executable tasks is to be done from the appropriate directory, using as a target system either 'VMS' or 'RSX'. The executable tasks must be transferred to the appropriate run area by the use of the VAX 'rename' command. Site specific software must be built with a target name of 'ALSARC', 'CHMARC', or 'CMPARC', then moved to the appropriate site specific subdirectories, to avoid confusion regarding the target system.

The majority of tasks should be built using the general system names. The following applications/libraries should be generated using the general target names of either RSX or VMS:

VAX	RSX
RLSCLIB	RLSCLIB11
PLOTLIB	PLOTLIB11
MATH	MATH
GC	GC
RLS	RLS
ENTRY	ENTRY
SIGNAL	SIGNAL
SIMDIS	SIMDIS

Applications which should be built specifying one of the legal site specific names are in the [.device] and [.xyplot] subdirectories.

During the task build a number of diagnostic/error messages will be generated, as indicated in the example. The messages regarding the 'multiply defines' of 'onexit' and 'exit' can be ignored. They are caused by the RLSCLIB version of the 'C' exit routine.

```
==============================================
$ @develop rsx
Previous logical name assignment replaced
        .
        .

$ set def [.gc]
$ @build
... FHDR  multiply defines ONEXIT
... FHDR  multiply defines EXIT
        .
        .
        .
$ rename [rls.gc]*.exe [rls.rsx]*
==============================================
```

The complete build consists of building all the libraries/applications indicated earlier in this chapter, in the order specified. All the build files will 'clean' the directories once they are finished the task build. This includes purging any old program versions and deleting all object modules. The user has to go into the library areas to purge all old libraries, as this is not done automatically.

6.3.4 Generating Help Files

After the initial generation of the help files, they need only be recreated when there are program changes affecting the user and thus the documentation. The files are generated using the 'makhlp' program, which extracts the title page from the program file, and creates a file called 'prgname.hlp'. An example of the 'makhlp' execution is:

```
=======================================
$ run [rls.doc]makhlp
enter filename ?
integrate.c
enter filename ?
cntrl z
$
=======================================
```

Once generated, these files should be moved to the general [RLS] area.

6.3.5 System Peculiarities

RSX system peculiarities have made it necessary to specifically include the name of the TEK4662 being used as the main plotter in the source code for the program CMPARC::[USERS:[RLS.XYPLOT]PLOTTR.C. The C function [RLS.XYPLOT.SPOOLER]TEK4662.C has some 'open()' and 'create()' routines that must be conditionally compiled for the different systems. The main program PLOTTR is then rebuilt with this function and the executable image is maintained in site specific directories USERS:[RLS.RSX.*]

6.4 STARTING NEW RLS USERS

6.4.1 Installing RLS On The PDP-11

To install the RLS software on the PDP-11/RSX system, do the following:

1. Create an account RLS/password on the PDP-11 with the UIC [277,54]. The account may be on any disk.

2. Copy all of the files in the disk area CMPARC::USERS:[RLS] to the PDP-11.

3. Copy all of the files in the disk area CMPARC::[USERS:[RLS.RSX] to the PDP-11.

```
NFT>=cmparc/account/password::users:[rls.rsx]*.*
```

4. In the RSX STARTUP.CMD command file include the following statements:

```
        ASN Dxx:=EI/GBL
        @EI:[277,54]STARTRLS
                where Dxx is the physical disk containing
                the RLS UIC [277,54]
```

5. Make up a command file EI:[277,54]STARTRLS.CMD that includes:

```
ASN TTyy:=PL0:/GBL
.IFINS PLTTER ins ei:[277,54]plotter.exe/task=pltter/slv=yes/ckp=yes/pri=55.
.IFINS ...rls ins ei:[277,54]rlscli.exe/task=...rls/pri=75./ckp=yes
.IFNACT PLTTER RUN PLTTER
.IFNACT LPARDA RUN LPARDA

                where Tyy is the physical terminal for the
                TEK4662 plotter.
```

6. After running the command files users can invoke RLS by entering 'RLS' at the MCR prompt.

6.4.2 Adding A New VAX User

For new VAX users, their login command procedure should be edited to include the following symbol definition:

```
RLS:==@USERS:[RLS]RLS
```

6.5 RLS BUGS AND LIMITATIONS

Although currently being used in a number of departments, there are problems with the RLS system which can cause difficulties during it's use. The list of problems have been broken into two list, those which must be fixed, and those which should be fixed. Of immediate concern are those problems which must be fixed or better documented to allow trouble free operation. The changes discussed in this catagory are bugs which can be fixed within individual program modules and therefore in a reasonable time frame. Those which should be fixed tend to deal with the limitations of the system and require that additional development work be done, or that major changes be made to the system. This may require a modification in the original design philosophy. However, once these limitations have been resolved, the result should be a commercial equivalent system.

For adequate trouble free operation, the following changes are recommended to correct the minor problems:

1. Improve the quality of the graphics:

    1. The axis lettering touches the axis.

    2. Capitals are forced to lowercase when titles are entered through prompt mode.

    3. The 0's at the origin run into each other.

    4. It should be possible to enclose the plot in a box.

    5. The user should be able to set the size of the plot label.

    6. There should be increased documentation on the use of the 'auto' keyword.

    7. The overlay feature does not work if the graph descriptor is too long. There is no warning message in this case.

8. The factor parameter increases the size of the graph, but also increases the space at the bottom. This results in the description at the top being forced off of the page and a large blank space at the bottom.

9. The labels and titles are not centered on the axis.

10. When doing a SEND PLOT to the TEK terminals the device is slaved and it is not possible to abort the operation in the middle of a long plot.

2. 'ZERO BASELINE' aborts if there are too many peaks. It should be rewritten to correct this, and to convert it from FORTRAN to 'C'.

3. Fix the 'FIND PEAKS':

   1. The program does not print the last peak value if the run terminated immediately after the peak finished. It should be modified to print the peak area, with a special flag.

   2. The data filtering logic should be checked.

   3. This module should be converted to 'C'.

4. 'ANALYZE PEAKS' should have a provision for amount of sample injected. (CHN analyzer requires that calculations of mg. of each of C, H and N be made)

5. There are a number of inconsistencies within the RLS file system, which are an annoyance to frequent users and a source of considerable confusion to naive users. Although all the difficulties cannot be easily fixed, improved documentation may help.

   1. File extension requirements vary from program to program, and this can cause problems when the output from one program is being used in some way by another program. For example, in the graphics section, the output file from the 'PREPARE PLOT' command is also the output file for the 'ADD TICS' command. The 'PREPARE PLOT' command will force an extension of '.plt'. The 'ADD TICS' command not only does not force the extension, it does not print an error message when it cannot find the plot file. Either all the programs should force extension, or none of the programs should.

   2. File handling is not consistent from process to process. At the present time, some of the files are being handled by a mini data handling system within RLS, while others are standard RSX/VMS files. This duality would be expensive to change, so improved documentation should be written.

   3. The contents of some of the files are destroyed when certain commands are executed, others are not. There should be consistent logic for this, as well as improved documentation to indicate which is the case for the

different commands.

4. The 'ADD DATASET' command will destroy an existing file in the directory if the user tries to create another file with the same name. It should give the user another chance to specify a new name or at least create a new version. As well, the user should be able to add an existing file to the database directory without adding all the information again.

6. Handling the 'C' subroutines through FORTRAN calls is confusing, messy, and given resource limitations, too time consuming to support. The remaining RLS FORTRAN modules should be rewritten in 'C', and the support dropped.

7. The RLS system generation is slow and tedious. It could be cleaned up so that it would be somewhat easier. To have a good generating method (such as a computer system generation) would require substantial effort.

8. Poor document generation and poor quality of the document appearance. For in-house documentation, another revision of both the User's manual and the Programmer's Manual is probably sufficient. Both manuals should be re-organized in such a manner that only the sections which have been changed need to be regenerated. This would reduce the effort required on the part of support staff.

It would be desirable to make more extensive changes to the RLS system, to include features commonly available in commercial systems. Before proceeding, another evaluation on the long term direction of the RLS development would have to be made. Some of the obvious changes which would have to be considered are the following:

1. Make the corrections specified in the first three points above.

2. Drop the 'C' support through FORTRAN calls, as in point 6 above.

3. Improve the documentation as specified above. In addition, send the completed documents to an editing group for modification and then to a professional printing group for high quality documents.

4. Automate the RLS system generation to simplify the procedure, and to allow the easy generation of only selected sections for different sites.

5. Increased data management capability. To compete with commercial systems in terms of capability and ease of use, additional data management should be included. This would fix the current problems with file handling that RLS now has. Improved data management should make it easier to handle more complex data manipulations.

6. Increased use of error messages. Currently, when a process aborts, the user does not get diagnostic messages, to help resolve the problem. This, with the data management, would make RLS a much more 'user friendly system'.

7. Improve RLS response time. Because of the method RLS uses for the parameter passing, the initial response time for task start up is poor - approximately 3 seconds to set a digital point. Other methods should be checked into in an attempt to improve the start-up time.

8. Improve RLS command language syntax and command handling to remove ambiguity. By making the language more structured, fewer problems should arise due to mis-use of the system.

9. There should be a STOP or ABORT command, to allow the user to interrupt a command sequence, and start over or restart the interrupted sequence.

10. There should be improved editing capabilities, or the ability to use system·editors on the data files.

11. Improve the quality control and testing of new or modified modules.

APPENDIX A

RLS FILE FORMATS

A.1 RLS FILE DIRECTORY

The RLS files being created and maintained are listed below. Detailed file formats are given in following sections.

1. Command Catalog (catalog.rls)
2. Command definition file (CDF)
3. Command modifier file (CMF)
4. Analog Signal File
5. Peak Data File
6. Calibration files
    1. Calibr1.rls
    2. Calibr2.rls
    3. Calibr3.rls
7. Graph Directory File
8. Plot Descriptor file
9. Math Function Files
    1. Fnnames.rls
    2. Fncoeffs.rls
10. X-Y Data set directory (xydata.rls)
11. X-Y Data file values

A.2 COMMAND CATALOG FILE

The catalog file contains all the valid RLS commands. Written in ASCII free format, each record has the RLS command line, the command definition file name and the program name. Immediately preceding the program name and the command definition file name are control characters, indicating the type of files. The records within the file are of the following form:

Field #	Variable	Type	Size	Description
1	context	char	<=40	RLS keyword
2	identifier	char	<=80	command line
3	control.cdf	char	1	control command definition file
4	cdf.name	char	<=128	cdf file name
5	control.prg	char	1	control program type
6	program.name	char	<=128	program name

where:

control.cdf - can be either:

x - no keyword definition file, field #4 will be blank

* - keyword defintion file specified, field #4 will contain file name. There should not intervening blanks between fields 3 & 4.

control.prg - can be either:

$ - executable program, field #6 will contain task.exe (no blanks between #5 & #6)

@ - command procedure, field #6 will contain task.dat. (no blanks between #5 & #6)

Field #4 and #6 should have the form:

device:[UIC]filename

A.3 COMMAND DEFINTION FILE (CDF)

The command defintion files, used to define the keyword values to be used as input, are ASCII free format. They can be on one line, or continued on subsequent lines if they have the dash '-', indicating that the record is to be continued on the next line. There are two general formats for the records in these files:

a) Record #1

All variables written in capitals are RLS keywords.

Field #	Variable	Type	Size	Description
1	KEYWORD	char	7	RLS keyword
2	user-keyword	char	vl	user specified keyword
3	TYPE	char	4	RLS keyword
4	user-type	char	vl	user specified type (int,float,string,octal logical)
5	HELP	char	4	RLS keyword
6	user-descr	char	vl	user description
7	DEFAULT	char	7	RLS keyword
8	user-default	char	vl	user specified default corresponding to 'type'

b) Record #2

Field #	Variable	Type	Size	Description
1	KEYWORD	char	7	RLS keyword
2	user-keyword	char	vl	user specified keyword
3	TYPE	char	4	RLS keyword
4*	FILE	char	4	Keyword triggering record type #2
5	STREAM	char	6	RLS keyword
6	user-stream	char	vl	user specified input/output
7	HELP	char	4	RLS keyword
8	user-help	char	vl	user keyword description
9	DEFAULT	char	7	RLS keyword
10	user-default	char	vl	user default (inputn/outputn)

A.4 COMMAND MODIFIER FILE (CMF)

The CMF files contain the default values required by the application program. The data is in ascii free format, in the order specified by the command definition file. There will be one value for each keyword defined in the command definition file.

Field #	Variable	Type	Size	Description
1	Value1	char	vl	as defined in CDF
2	Value2	char	vl	as defined in CDF
.				
.				
n	Valuen	char	vl	as defined in CDF

NOTE: The values in the CMF can be any of the data types specified in the command definition file (ie int,float,string,octal,logical). The size will be dependent on the data type.

A.5 ANALOG SIGNAL FILES

The analog signal files are produced by the 'monitor analog' plus switches or the 'start analog' command. The files consist of two record types, the RLS header, followed by a stream of floating point values representing voltages read at the computer a/d converter.

a) RLS header

Field #	Variable	Type	Size	Description
1	SUBTITLE	char	8	RLS keyword
2	user-title	char	<=30	user-title
3	CHANNEL	char	7	RLS keyword
4	user-channel	int	4	channel number
5	FREQUENCY	char	9	RLS keyword
6	user-frequency	float	4	user requested frequency
7	TIME	char	4	RLS keyword
8	sample-start	char	vl	time at sample start
9	STEPSIZE	char	8	RLS keyword
10	user-stepsize	float	4	user requested stepsize
11	XUNITS	char	6	RLS keyword
12	x-units	char	vl	user specified units for x-axis
13	YUNITS	char	6	RLS keyword
14	y-units	char	vl	user specified units for y-axis
15	DATA	char	4	RLS end of header flag

b) Data Record

Fields #	Variables	Type	Size	Description
1	voltage	float	4	voltage reading from a/d converter
•				
•				
n	voltage	float	4	voltage reading

A.6 PEAK DATA FILES

Peak data files contain information on the peaks that have been detected in the signal file. The two record types are the RLS header followed by the peak data records. The files are in ascii free format.

a) RLS header

Field #	Variable	Type	Size	Description
1	GATE	char	4	RLS keyword
2	gate-value	int	4	user specified persistent change threshold
3	SLOPE	char	5	RLS keyword
4	slope-value	float	4	slope sensitivity
5	DENSITY	char	7	RLS keyword
6	density-value	float	4	smoothing parameter
7	WIDTH	char	5	RLS keyword
8	peak-width	int	4	used to determine peak stop
9	SUBTITLE	char	8	RLS keyword
10	user-title	char	vl	user specified subtitle
11	CHANNEL	char	7	RLS keyword
12	channel-no	int	4	user specified channel number
13	FREQUENCY	char	9	RLS keyword
14	user-freq	float	4	user specified sampling frequency
15	TIME	char	4	RLS keyword
16	sample-time	char	vl	sample start time
17	STEPSIZE	char	8	RLS keyword
18	stepsize	float	4	sampling time interval
19	XUNITS	char	6	RLS keyword
20	x-units	char	vl	x-axis units

21	YUNITS	char	6	RLS keyword
22	y-units	char	vl	y-axis units
23	DATA	char	4	RLS flag - end of header

b) Peak Data Records (one for each peak)

Field #	Variable	Type	Size	Description
1	area	float	4	peak area
2	height	float	4	peak height
3	rt-time	float	4	peak retention time
4	st-height	float	4	standard height
5	st-time	float	4	standard retention time
6	width	float	4	peak width
7	end-height	float	4	?
8	end-time	float	4	peak finish
9	sep-code	char	vl	separation code
10	conc	float	4	concentration
11	type	char	vl	peak type
12	name	char	vl	peak name
13	id-time	float	4	peak identification time
14	resp-factor	float	4	response factor

A.7 CALIBRATION FILES

There are three calibration files, created and maintained by the calibration programs.

a) Calibrl.rls

This file describes the peak method.

Field #	Variable	Type	Size	Description
1	name	char	<=20	Calibration name
2	method	char	4	calibration method (APCT,ESTD, ISTD,NORM)
3	using	char	<=7	using areas or heights
4	units	char	<=10	units for report concentrations
5	description	char	<=40	user description of calibration

b) Calibr2.rls

This file describes the peaks in the various methods. There is one record in the file for each peak in all the calibration files.

Field #	Variable	Type	Size	Description
1	calib-name	char	<=20	calibration name
2	rt-time	float	4	retention time
3	peak-name	char	<=20	peak name
4	type	char	vl	peak type
5	poly-order	int	4	poly order
6	x**3	float	4	1st poly order (high order)
7	x**2	float	4	2nd poly order
8	x**3	float	4	3rd poly order
9	x**4	float	4	4th ploy order

c) Calibr3.rls

This file defines the points in the calibration curves for each calibration peak defined. There is a record in this relation for each calibration point.

Field #	Variable	Type	Size	Description
1	Calib-name	char	<=20	Calibration name
2	Peak-name	char	<=20	peak name
3	amount	float	4	peak amount
4	size	float	4	peak size (area or height)

A.8 GRAPH DIRECTORY FILE (GRAPH.RLS)

The graph directory file contains information on the current set of plots. This includes the minimum and maximum x and y values that the graph has been scaled to, and the legend number for the next graph overlay. The file is written in ascii free format.

Field #	Variable	Type	Size	Description
1	graph-name	char	<=20	graph file name
2	x-min	float	4	minimum x-axis value
3	x-max	float	4	maximum x-axis value
4	y-min	float	4	minimum y-axis value
5	y-max	float	4	maximum y-axis value
6	legend-number	int	2	graph legend number
7	graph-descr	char	<=40	graph description

A.9 PLOT DESCRIPTOR FILE

For each graph there is a plot descriptor file (PDF) with the extension '.plt'. The plot descriptor files are a set of (x,y) pen moves.

Field #	Variable	Type	Size	Description
1	code	int	4	pen codes (valid codes listed below)
2	x-cor	float	4	x-coordinate im cms or new pen number
3	y-cor	float	4	y-coordinate in cms

Valid Pen codes:

1	Draw
2	Move
3	New Pen
-13	End Plot
-14	Init Plot

A.10 MATH ROUTINES

The math routines access three RLS created files. The XYDATA.RLS (directory file for x-ydata files) is described in the following section. It is created and maintained by RLS as part of the data entry portion. The files created and modified by the RLS math routines are FNNAMES.RLS and FNCOEFFS.RLS.

a) Fnnames.rls

Description of the function data.

Field #	Variable	Type	Size	Description
1	fnc-name	char	<=20	function name
2	fnc-form	char	<=20	function form
3	fnc-descr	char	<=40	function description (user's)

b) Fncoeffs.rls

Description of the function form and coefficients.

Field #	Variable	Type	Size	Description
1	fnc-name	char	<=20	function name (user specified)
2	fnc-form	char	<=20	function form
3	coef-name	int	4	coefficient number
4	fnc-coef	float	4	function coefficient

A.11 X-Y DATASET DIRECTORY (XYDATA.RLS)

The data entry directory maintains file information for the files created or modified using the data entry routines. Both the data entry commands and the math routines use this directory for data file information. Files can be entered into this directory only through the data entry and the math routines.

Field #	Variable	Type	Size	Description
1	dataset-name	char	<=20	dataset or file name
2	user-descr	char	<=40	user description for dataset
3	x-units	char	<=10	x-axis units
4	x-label	char	<=80	x-axis label
5	y-units	char	<=10	y-axis units
6	y-label	char	<=80	y-axis label
7	n-rows	int	4	number of rows
8	n-columns	int	4	number of columns

A.12 X-Y DATA FILE RECORDS

The x-y data files are of the format of the other RLS files, with the RLS header, followed by the stream of x-y data points.

APPENDIX B

RLS SYSTEM GENERATION FILES

B.1 INTRODUCTION

The RLS system generation is done through a series of command files. The main RLS area contains all the indirect command files for setting logical assignments, and assigning compilers and task builders. Each subdirectory contains a build file, which will build the contents of that directory, as well as initiating the build of any appropriate subdirectories.

As not all of the subdirectories are part of the RLS production system, the description of the build files has been divided into two different sections. The RLS production build files should be error free. Those in the second section are those used for development work, and may require further corrections.

B.2 CONTROL FILES

The RLS control files, located in the main directory are those files required to set logical assignments, and to assign compilers and task builders for the different systems. They are an essential part of any task build, whether production or development.

DEVELOP.COM accepts one argument, the name of the site or system for which the software is being generated. Using this argument, the correct library assignments, 'C' header files, and conditional compile assignments are made.

FOR.COM and RSXLINK.COM are foreign commands which replace the VMS commands. This allows the build files to issue one command, and depending on the target system, invoke the appropriate utilities. In the case of the linking, it also allows the correct switches to be set for the RSX system, since the link/rsx switch does not do this.

B.2.1 DEVELOP.COM

```
$ ! Modifications:
$ ! 3 March 1983 Alan Covington
$ !     Changed sys$library:fcsres to sys$sysroot:[sysmgr.newsyslib]fcsres
$ !     so that V4.0 development will work.
$ ! 9 March 1983 Karin Tremaine
$ !     Changed FCSRES assignment to read sys$library:fcsresnew fcsres
$
$ ! 14 FEB 1984 Karin Tremaine
$ !     Changed 'C' compiler command, so that it no longer references
$ !     CC.COM file
$ ! 22 JUN 1984 Karin Tremaine
$ !     Cleaned up the command file to reflect the changes in C compiler
$ !     and changes in the library requirements
$
$ develop:==@users:[rls]develop
$ for:==@users:[rls]for
$ assign sys$disk sy
$ assign users ei
$ if p1 .eqs. "" then inquire p1 target node name

$ ! RSX with FPP and FCSRES

$ if p1 .nes. "RSX" then goto s9
$ target:=="RSX"
$ cc:== $cc /def=(RSX,'target')/incl=users:[rls.library]/member
$ macro:==macro /rsx
$ libr:== libr /rsx
$ link:== @users:[rls]rsxlink           ! link /rsx/tkb=users:[rls]tkb
$ assign sys$sysroot:[sysmgr.wpascal.rsx11m]fhdr.obj chdr
$ assign sys$sysroot:[sysmgr.wpascal.rsx11m]pflib.olb clib
$ assign users:[rls.library.rsx]exemc.mlb exemc
$ assign users:[rls.library.rsx]exelib exelib
$ assign users:[rls.library.rsx]math11 math
$ assign users:[rls.library.rsx]plotdev11 plotdev
$ assign users:[rls.library.rsx]plotlib11 plotlib
$ assign users:[rls.library.rsx]rlsclib11 rlsclib
$ assign sys$library:f77ots f77ots
$ assign sys$library:fcsresnew fcsres
$ assign sys$library:syslib syslib
$ s9:

$ ! VMS

$ if p1 .nes. "VMS" then goto s10
$ target:=="VMS"
$ cc:== $cc /def=(VMS,'target')/incl=users:[rls.library]/member
$ macro:==macro
$ libr:== libr
$ link:==link
$ assign users:[rls.library.vms] lb:
$ assign sys$sysroot:[sysmgr.wpascal.vaxvms]chdr.obj chdr
$ assign sys$sysroot:[sysmgr.wpascal.vaxvms]plib.olb clib
$ assign users:[rls.library.vms]math math
$ assign users:[rls.library.vms]plotdev plotdev
$ assign users:[rls.library.vms]plotlib plotlib
$ assign users:[rls.library.vms]rlsclib rlsclib
$ s10:

$ ! ALSARC

$ if p1 .nes. "ALSARC" then goto s1
$ target:=="ALSARC"
$ cc:== $cc /def=(RSX,'target')/incl=users:[rls.library]/member
$ macro:==macro /rsx
$ libr:== libr /rsx
$ link:== @users:[rls]rsxlink           ! link /rsx/tkb=users:[rls]tkb
$ assign users:[rls.library.rsx.alsarc] lb:
$ assign sys$sysroot:[sysmgr.wpascal.rsx11m]fhdr.obj chdr
$ assign sys$sysroot:[sysmgr.wpascal.rsx11m]pflib.olb clib
$ assign users:[rls.library.rsx]exemc.mlb exemc
$ assign users:[rls.library.rsx.alsarc]rsxmc.mac rsxmc
$ assign users:[rls.library.rsx.alsarc]rsx11m.stb rsx11m
$ assign users:[rls.library.rsx]exelib exelib
$ assign users:[rls.library.rsx]math11 math
$ assign users:[rls.library.rsx]plotdev11 plotdev
$ assign users:[rls.library.rsx]plotlib11 plotlib
$ assign sys$library:f77ots f77ots
$ assign users:[rls.library.rsx]rlsclib11 rlsclib
$ assign users:[rls.library.rsx.alsarc]targetlib targetlib
$ assign sys$library:fcsresnew fcsres
$ assign dba0:[1,1]syslib syslib
$ s1:

$ ! CMPARC

$ if p1 .nes. "CMPARC" then goto s2
$ target:=="CMPARC"
$ cc:== $cc /def=(RSX,'target')/incl=users:[rls.library]/member
$ macro:==macro
$ libr:== libr
$ link:==link
$ assign users:[rls.library.vms] lb:
$ ! assign users:[rls.library.vms]chdr chdr
$ ! assign users:[rls.library.vms]clib clib
$ assign sys$sysroot:[sysmgr.wpascal.vaxvms]chdr.obj chdr
$ assign sys$sysroot:[sysmgr.wpascal.vaxvms]plib.olb clib
$ assign users:[rls.library.vms]math math
$ assign users:[rls.library.vms]plotdev plotdev
$ assign users:[rls.library.vms]plotlib plotlib
$ assign users:[rls.library.vms]rlsclib rlsclib
$ assign users:[rls.library.rsx.cmparc]targetlib targetlib
$ s2:

$ ! CHMARC

$ if p1 .nes. "CHMARC" then goto s3
$ target:=="CHMARC"
$ cc:== $cc /def=(RSX,'target')/incl=users:[rls.library]/member
$ macro:==macro /rsx
$ libr:== libr /rsx
$ link:== @users:[rls]rsxlink           !link /rsx/tkb=users:[rls]tkb
$ assign users:[rls.library.rsx.chmarc] lb:
$ assign sys$sysroot:[sysmgr.wpascal.rsx11m]fhdr.obj chdr
$ assign sys$sysroot:[sysmgr.wpascal.rsx11m]pflib.olb clib
$ assign users:[rls.library.rsx.chmarc]rsxmc.mac rsxmc
$ assign users:[rls.library.rsx.chmarc]rsx11m.stb rsx11m
$ assign users:[rls.library.rsx]exemc.mlb exemc
$ assign users:[rls.library.rsx]exelib exelib
$ assign users:[rls.library.rsx]math11 math
$ assign users:[rls.library.rsx]plotdev11 plotdev
$ assign sys$library:f77ots f77ots
$ assign users:[rls.library.rsx]plotlib11 plotlib
$ assign users:[rls.library.rsx]rlsclib11 rlsclib
$ assign users:[rls.library.rsx.chmarc]targetlib targetlib
$ assign sys$library:fcsresnew fcsres
$ assign dba0:[1,1]syslib syslib
$ s3:
```

B.2.2 FOR.COM

```
$ if target .nes. "ALSARC" then goto s1
$ mcr f77 'p1'='p1'.for
$ goto exit
$ s1:

$ if target .nes. "CHMARC" then goto s2
$ mcr f77 'p1'='p1'.for
$ goto exit
$ s2:

$ if target .nes. "CMPARC" then goto s3
$ fortran 'p1'
$ goto exit
$ s3:

$ if target .nes. "VMS" then goto s4
$ fortran 'p1'
$ goto exit
$ s4:

$ if target .nes. "RSX" then goto s5
$ mcr f77 'p1'='p1'.for
$ goto exit
$ s5:

$ exit:
```

B.2.3 RSXLINK.COM

```
$
$ ! This file does a RSX task build from a VMS linker command
$ ! line (excluding switches).  This is necessary so that the
$ ! appropriate TKB switches (/fp/cp) can be set.
$ ! This file accesses the new FCSRES (sys$library:newfcsres).
$ ! Thus the tasks produced will only run on V4.0 of RSX.
$ ! When the Version 4.0 TKB and FCSRES become standard, change
$ ! sys$library:newfcsres in develop.com to sys$library:fcsres
$ !
$ ! K. Tremaine  29-Feb-84  Removed the reference to FCSRES
$ !                         as there are problems with CLIB/FCSRES
$ !                         compatability
$
$ if pl .eqs. "" then goto ERROR
$
$ comma = 'f$locate(",", pl)
$ first_file := "''f$extract(0, comma, pl)'"
$
$ ! Switch all the /LIBR switches for /LB switches
$
$ newpl := ""
$NEXTLBSW:
$ lbsw = 'f$locate("/LIBR", pl)
$ if(lbsw .eq. 'f$length(pl)) then goto NOLBSW
$       newpl := 'newpl''f$extract(0, lbsw, pl)'/LB
$       pl := 'f$extract(lbsw + 5, f$length(pl), pl)
$       goto NEXTLBSW
$NOLBSW:
$ newpl := 'newpl''pl'
$
$ ! Now create the TKB command file and put in the appropriate
$ ! TKB commands and options
$
$ open tkbcmd tkbtmp.cmd /write /error=CREATE_ERROR
$
$ write tkbcmd "''first_file'/fp/cp=''newpl'"
$ write tkbcmd "/"
$ write tkbcmd "units=10"
$ write tkbcmd "actfil=10"
$ write tkbcmd "stack=3000"
$ write tkbcmd "asg=sy:1:2:3:4:6:7:8"
$ ! write tkbcmd "reslib=fcsres/ro"
$ write tkbcmd "//"
$ close tkbcmd
$
$ ! Now do task build
$
$ mcr tkb @tkbtmp.cmd
$
$ ! clean up
$
```

```
$ delete tkbtmp.cmd;*
$
$ exit
$
$ERROR:
$ write sys$output "RSX task builder requires a file to build"
$ exit
$
$CREATE_ERROR:
$ write sys$output "RSX task builder.  Cannot create temporary command file."
$ exit
```

B.3 RLS PRODUCTION SYSTEM BUILD

The following files listed here are all the build files required to build the RLS production system. The files are listed in alphabetical order, based on directory names. The actual order in which the build must occur is quite different. To insure that all the programs execute properly, it is important that the following sequence for the actual build is followed:

1. Library areas:

    1. [rls.rlsclib]

    2. [rls.clib]

    3. [rls.relation]

    4. [rls.calcomp]

2. Applications which depend on modules being inserted in the course of another directory build require that the following directories are built in the order specified below:

    1. [rls.math]

    2. [rls.rls]

    3. [rls.device]

    4. [rls.xyplot]

    5. [rls.signal]

3. The remaining areas can be built any time after the initial libraries are built.

    1. [rls.entry]

    2. [rls.epr]

    3. [rls.gc]

    4. [rls.simdis]

B.3.1 [RLS.CALCOMP]BUILD.COM

```
$ savedef:='f$directory()'
$ set def [rls.calcomp]
$ libr /create=blocks:1 plotlib
$ !
$ ! subdirectories
$ !
$ if target .eqs. "VMS" then @[.vms]build
$ if target .eqs. "RSX" then @[.rsx]build
$ !
$ ! non-CALCOMP functions
$ !
$ cc clip
$ libr /rep plotlib clip
$ !
$ ! CALCOMP functions
$ !
$ cc axis
$ cc calglb
$ cc factor
$ cc line
$ cc newpen
$ cc number
$ cc plot
$ cc plots
$ cc scale
$ cc symbol
$ cc where
$ libr /rep plotlib axis,calglb,factor,line,newpen
$ libr /rep plotlib number,plot,plots,scale,symbol,where
$ !
$ ! cleanup
$ !
$ del *.obj.*
$ purge
$ set def 'savedef'
```

B.3.1.1 [RLS.CALCOMP.RSX]BUILD.COM -

```
$ savedef:='f$directory()'
$ set def [rls.calcomp.rsx]
$ !
$ ! system dependent functions
$ !
$ macro faxis
$ macro fline
$ macro ffactor
$ macro fnewpen
$ macro fnumber
$ macro fplot
$ macro fplots
$ macro fscale
$ macro fsymbol
$ macro fwhere
$ libr /rep plotlib faxis,fline,ffactor,fnewpen
$ libr /rep plotlib fnumber,fplot,fplots,fscale
$ libr /rep plotlib fsymbol,fwhere
$ !
$ ! cleanup
$ !
$ del *.obj.*
$ purge
$ set def 'savedef'
```

B.3.1.2 [RLS.CALCOMP.VMS]BUILD.COM -

```
$ savedef:='f$directory()'
$ set def [rls.calcomp.vms]
$ !
$ ! system dependent functions
$ !
$ macro faxis
$ macro ffactor
$ macro fline
$ macro fnewpen
$ macro fnumber
$ macro fplot
$ macro fplots
$ macro fscale
$ macro fsymbol
$ macro fwhere
$ libr /rep plotlib faxis,ffactor,fline,fnewpen
$ libr /rep plotlib fnumber,fplot,fplots,fscale,fsymbol,fwhere
$ !
$ ! cleanup
$ !
$ del *.obj.*
$ purge
$ set def 'savedef'
```

B.3.2 [RLS.CLIB]BUILD.COM

```
$ savedef:='f$directory()'
$ set def [rls.clib]
$ !
$ !
$ ! subroutines/functions
$ !
$ cc exit
$ cc main
$ libr /rep rlsclib exit, main
$ !
$ ! cleanup
$ !
$ del *.obj.*
$ purge
$ set def 'savedef'
```

B.3.3 [RLS.DEVICE]BUILD.COM

```
$ savedef:='f$directory()'
$ set def [rls.device]
$ set noon
$ !
$ ! subdirectories
$ ! 22-jun-84 K. Tremaine
$ !     This subdirectories are no longer part of the RLS generation
$ !     This function has been included as part of the SYSGEN
$ !
$ ! if target .eqs. "ALSARC" then @[.rsx.alsarc]build
$ ! if target .eqs. "CHMARC" then @[.rsx.chmarc]build
$ !
$ ! functions
$ !
$ if target .nes. "ALSARC" .and. target .nes. "CHMARC" then goto s20
$ cc cdigio
$ cc lpaemu
$ cc voltage
$ libr /rep targetlib cdigio,lpaemu,voltage
$ s20:
$ cc waitabs
$ cc waitrel
$ libr /rep rlsclib waitrel,waitabs
$ !
$ ! programs
$ !
$ cc anamonitr
$ link anamonitr,rlsclib/libr,chdr,clib/libr
$ cc anashow
$ link anashow,rlsclib/libr,chdr,clib/libr
$ cc anastart
$ link anastart,rlsclib/libr,chdr,clib/libr
$ cc anastop
$ link anastop,rlsclib/libr,chdr,clib/libr
$ cc anawait
$ link anawait,rlsclib/libr,chdr,clib/libr
$ cc clkwait
$ link clkwait,rlsclib/libr,chdr,clib/libr
$ cc digpulse
$ link digpulse,targetlib/libr,rlsclib/libr,chdr,clib/libr
$ cc digread
$ link digread,targetlib/libr,rlsclib/libr,chdr,clib/libr
$ cc digwait
$ link digwait,targetlib/libr,rlsclib/libr,chdr,clib/libr
$ cc digwrite
$ link digwrite,targetlib/libr,rlsclib/libr,chdr,clib/libr
$ cc lparda
$ if target .nes. "ALSARC" then goto next
$ mcr tkb
lparda/fp/cp,lparda/-sp=lparda
targetlib/lb,rlsclib/lb
chdr,clib/lb
```

```
/
stack=1500
units=32
actfil=8
pri=100
//
$ next:
$ if target .nes. "CHMARC" then goto exit
$ mcr tkb
lparda/fp/cp,lparda/-sp=lparda
targetlib/lb
rlsclib/lb
chdr,clib/lb
/
stack=3000
units=32
actfil=32
pri=100
//
$ exit:
$ cc trmread
$ link trmread,rlsclib/libr,chdr,clib/libr
$ cc trmring
$ link trmring,rlsclib/libr,chdr,clib/libr
$ cc trmwait
$ link trmwait,rlsclib/libr,chdr,clib/libr
$ cc trmwrite
$ link trmwrite,rlsclib/libr,chdr,clib/libr
$ !
$ ! cleanup
$ !
$ del *.obj.*
$ set on
$ purge
$ set def 'savedef'
```

B.3.4 [RLS.ENTRY]BUILD.COM

```
$ savedef:='f$directory()'
$ set def [rls.entry]
$ !
$ ! programs
$ !
$ cc xyadd
$ link xyadd,rlsclib/libr,chdr,clib/libr
$ cc xychange
$ link xychange,rlsclib/libr,chdr,clib/libr
$ cc xydel
$ link xydel,rlsclib/libr,chdr,clib/libr
$ cc xyshow
$ link xyshow,rlsclib/libr,chdr,clib/libr
$ !
$ ! cleanup
$ !
$ del *.obj.*
$ purge
$ set def 'savedef'
```

B.3.5 [RLS.GC]BUILD.COM

```
$ savedef:='f$directory()'
$ set def [rls.gc]
$ set noon
$ for bascor
$ link bascor,rlsclib/libr,chdr,clib/libr,f77ots/libr
$ cc calibchg
$ link calibchg,rlsclib/libr,chdr,clib/libr
$ cc calibdel
$ link calibdel,rlsclib/libr,chdr,clib/libr
$ cc calibent
$ link calibent,math/libr,rlsclib/libr,chdr,clib/libr
$ cc calibextr
$ link calibextr,rlsclib/libr,chdr,clib/libr
$ cc calibins
$ link calibins,rlsclib/libr,chdr,clib/libr
$ cc calibshow
$ link calibshow,rlsclib/libr,chdr,clib/libr
$ cc calibupd
$ link calibupd,math/libr,rlsclib/libr,chdr,clib/libr
$ cc convrtchn
$ link convrtchn,rlsclib/libr,chdr,clib/libr
$ cc gcreport
$ link gcreport,rlsclib/libr,chdr,clib/libr
$ for lpspkp
$ link lpspkp,rlsclib/libr,chdr,clib/libr
$ cc postrun
$ link postrun,math/libr,rlsclib/libr,chdr,clib/libr
$ cc ticks
$ link ticks,plotlib/libr,rlsclib/libr,chdr,clib/libr
$ !
$ ! clean up
$ !
$ delete *.obj.*
$ purge
$ set on
$
$ set def 'savedef'
```

B.3.6 [RLS.MATH]BUILD.COM

```
$ savedef:='f$directory()'
$ set def [rls.math]
$ set noon
$ !
$ ! subroutines
$ !
$ cc basis
$ cc dotpr
$ cc evaluate
$ cc exp
$ cc fnctype
$ cc log
$ cc lsqfit
$ cc matinv
$ cc mllfndlsq
$ cc mlldecomp
$ cc mllnorm
$ cc mllsolve
$ cc mmul
$ cc mtrans
$ cc norm
$ cc pwr
$ cc sin
$ cc svemg
$ cc uxfcoeffs
$ cc vpoly
$ cc vsmul
$ cc vswap
$ cc xfcoeffs
$ cc xfdata
$ libr /create=blocks:1 math
$ libr /rep math basis,dotpr,evaluate,exp,fnctype,log,lsqfit
$ libr /rep math matinv,mllfndlsq,mlldecomp,mllnorm
$ libr /rep math mllsolve,mmul,mtrans
$ libr /rep math norm,pwr,sin,svemg,uxfcoeffs,vpoly,vswap
$ libr /rep math vsmul,xfcoeffs,xfdata
$ !
$ ! programs
$ !
$ cc bunch
$ link bunch,rlsclib/libr,chdr,clib/libr
$ cc convert
$ link convert,rlsclib/libr,chdr,clib/libr
$ cc fnadd
$ link fnadd,math/libr,rlsclib/libr,chdr,clib/libr
$ cc fndel
$ link fndel,math/libr,rlsclib/libr,chdr,clib/libr
$ cc fnshow
$ link fnshow,math/libr,rlsclib/libr,chdr,clib/libr
$ cc residual
$ link residual,math/libr,rlsclib/libr,chdr,clib/libr
$ cc rlsanova
$ link rlsanova,math/libr,rlsclib/libr,chdr,clib/libr
$ cc rlseval
$ link rlseval,math/libr,rlsclib/libr,chdr,clib/libr
$ cc rlslsqfit
$ link rlslsqfit,math/libr,rlsclib/libr,chdr,clib/libr
$ !
$ ! clean up
$ !
$ delete *.obj.*
$ purge
$ set on
$
$ set def 'savedef'
```

B.3.7 [RLS.RELATION]BUILD.COM

```
$ savedef:='f$directory()'
$ set def [rls.relation]
$ !
$ ! subroutines/functions
$ !
$ cc addext
$ cc closer
$ cc creatr
$ cc deltuple
$ cc gettuple
$ cc instuple
$ cc openr
$ cc puttuple
$ cc rel
$ cc rpltuple
$ libr /rep rlsclib addext,closer,creatr,deltuple,gettuple,instuple
$ libr /rep rlsclib openr,puttuple,rel,rpltuple
$ !
$ ! cleanup
$ !
$ del *.obj.*
$ purge
$ set def 'savedef'
```

B.3.8 [RLS.RLS]BUILD.COM

```
$ savedef:='f$directory()'
$ set noon
$ set def [rls.rls]
$ !
$ ! subroutines
$ !
$ cc getcpd
$ cc opncdf
$ cc prcnam
$ cc prompt
$ cc readhdr
$ libr /rep rlsclib getcpd,opncdf,prcnam,prompt,readhdr
$ !
$ ! programs
$ !
$ cc addcomm
$ link addcomm,rlsclib/libr,chdr,clib/libr
$ cc crecdf
$ link crecdf,rlsclib/libr,chdr,clib/libr
$ cc ctradd
$ link ctradd,rlsclib/libr,chdr,clib/libr
$ cc ctrdecr
$ link ctrdecr,rlsclib/libr,chdr,clib/libr
$ cc ctrdelete
$ link ctrdelete,rlsclib/libr,chdr,clib/libr
$ cc ctrshow
$ link ctrshow,rlsclib/libr,chdr,clib/libr
$ cc delcomm
$ link delcomm,rlsclib/libr,chdr,clib/libr
$ cc rlscli
$ link rlscli,rlsclib/libr,chdr,clib/libr,f77ots/libr
$ cc rlshlp
$ link rlshlp,rlsclib/libr,chdr,clib/libr
$ cc rlsset
$ link rlsset,rlsclib/libr,chdr,clib/libr
$ cc rlssho
$ link rlssho,rlsclib/libr,chdr,clib/libr
$ cc shocomm
$ link shocomm,rlsclib/libr,chdr,clib/libr
$ !
$ ! clean up
$ !
$ delete *.obj.*
$ purge
$
$ set def 'savedef'
```

B.3.9 [RLS.RLSCLIB]BUILD.COM

```
$ savedef:='f$directory()'
$ set def [rls.rlsclib]
$ !
$ ! create the object library
$ !
$ libr /create=blocks:1 rlsclib
$ !
$ ! subdirectories
$ !
$ if target .eqs. "VMS" then @[.vms]build
$ if target .eqs. "RSX" then @[.rsx]build
$ !
$ ! subroutines/functions
$ !
$ cc assign,chain,chktype,cmfnam,crelog
$ cc cnvstr,dassgn,date,dequeue
$ cc eofon,filexpnd,filnam,getoctal,gtty
$ cc getpid,getfloat,getint,getlog,getstring,host
$ cc ioral,iowal,newprompt,queue,rename,skiphdr,symbol
$ cc stty,time,translate,trnlog,xfrhdr
$ libr /rep rlsclib assign,chain,chktype,cmfnam,crelog
$ libr /rep rlsclib cnvstr,dassgn,date,dequeue
$ libr /rep rlsclib eofon,filexpnd,filnam,getoctal,gtty
$ libr /rep rlsclib getpid,getfloat,getint,getlog,getstring,host
$ libr /rep rlsclib ioral,iowal,newprompt,queue,rename,skiphdr,symbol
$ libr /rep rlsclib stty,time,translate,trnlog,xfrhdr
$ !
$ ! cleanup
$ !
$ del *.obj.*
$ purge
$ set def 'savedef'
```

B.3.9.1 [RLS.RLSCLIB.RSX]BUILD.COM -

```
$ savedef:='f$directory()'
$ set def [rls.rlsclib.rsx]
$ !
$ ! assemble functions
$ !
$ macro c5ta
$ macro cdtoe
$ macro cdtof
$ macro clseek
$ macro csi
$ macro files11
$ !macro frenam
$ macro ftoc
$ macro rcall
$ macro rdfui
$ !macro renam
$ macro wfown
$ !
$ ! put into object library
$ !
$ libr /rep rlsclib c5ta,cdtoe,cdtof,clseek,csi
$ libr /rep rlsclib files11,ftoc
$ libr /rep rlsclib rcall,rdfui,wfown
$ !
$ ! cleanup
$ !
$ del *.obj.*
$ purge
$ set def 'savedef'
```

B.3.9.2 [RLS.RLSCLIB.VMS]BUILD.COM -

```
$ savedef:='f$directory()'
$ set def [rls.rlsclib.vms]
$ !
$ ! assemble functions
$ !
$ macro cdtoe
$ macro cdtof
$ macro clseek
$ macro ftoc
$ !
$ ! put into object library
$ !
$ libr /rep rlsclib cdtoe,cdtof,clseek,ftoc
$ !
$ ! cleanup
$ !
$ del *.obj.*
$ purge
$ set def 'savedef'
```

B.3.10 [RLS.SIGNAL]BUILD.COM

```
$ savedef:='f$directory()'
$ set def [rls.signal]
$ set noon
$ !
$ ! subroutines
$ !
$ for cfftb
$ for cfftf
$ for cffti
$ cc drum
$ macro fftb
$ macro fftf
$ macro ffti
$ cc polar
$ libr /rep math drum,cfftb,cfftf,cffti,fftb,fftf,ffti,polar
$ !
$ ! programs
$ !
$ cc amplitude
$ link amplitude,math/libr,rlsclib/libr,chdr,clib/libr
$ cc autocorr
$ link autocorr,math/libr,rlsclib/libr,chdr,clib/libr,f77ots/libr
$ cc derivativ
$ link derivativ,math/libr,rlsclib/libr,chdr,clib/libr
$ cc equi
$ link equi,math/libr,rlsclib/libr,chdr,clib/libr
$ cc extract
$ link extract,math/libr,rlsclib/libr,chdr,clib/libr
$ cc fftb
$ link fftb,math/libr,rlsclib/libr,chdr,clib/libr,f77ots/libr
$ cc fftf
$ link fftf,math/libr,rlsclib/libr,chdr,clib/libr,f77ots/libr
$ cc integral
$ link integral,math/libr,rlsclib/libr,chdr,clib/libr
$ cc npoints
$ link npoints,math/libr,rlsclib/libr,chdr,clib/libr
$ cc nyquist
$ link nyquist,math/libr,rlsclib/libr,chdr,clib/libr
$ cc phase
$ link phase,math/libr,rlsclib/libr,chdr,clib/libr
$ cc power
$ link power,math/libr,rlsclib/libr,chdr,clib/libr
$ cc rmvmean
$ link rmvmean,math/libr,rlsclib/libr,chdr,clib/libr
$ cc rmvtrend
$ link rmvtrend.math/libr,rlsclib/libr,chdr,clib/libr
$ !
$ ! clean up
$ !
$ delete *.obj.*
$ purge
$ set on
$ set def 'savedef'
```

B.3.11 [RLS.SIMDIS]BUILD.COM

```
$ savedef:='f$directory()'
$ set def [rls.simdis]
$ set noon
$ cc wrtfile
$ link wrtfile,rlsclib/libr,chdr,clib/libr
$ cc remove
$ link remove,rlsclib/libr,chdr,clib/libr
$ cc simbas
$ link simbas,rlsclib/libr,chdr,clib/libr
$ cc pcarea
$ link pcarea,rlsclib/libr,chdr,clib/libr
$ cc scale
$ link scale,rlsclib/libr,chdr,clib/libr
$ cc report
$ link report,rlsclib/libr,chdr,clib/libr
$ cc calibsim
$ link calibsim,rlsclib/libr,chdr,clib/libr
$
$
$!
$!  clean up
$!
$
$ delete *.obj;*
$ purge
$ set on
$
$ set def 'savedef'
```

B.3.12 [RLS.XYPLOT]BUILD.COM

```
$ savedef:='f$directory()'
$ set noon
$ set def [rls.xyplot]
$ !
$ ! subdirectories
$ !
$ @[.spooler]build
$ !
$ ! programs
$ !
$ cc graphdel
$ link graphdel,rlsclib/libr,chdr,clib/libr
$ cc graphshow
$ link graphshow,rlsclib/libr,chdr,clib/libr
$ cc plot
$ link plot,plotlib/libr,math/libr,rlsclib/libr,chdr,clib/libr
$ !cc pdmp
$ !link pdmp,chdr,clib/libr
$ cc plottr
$ link plottr,plotdev/libr,rlsclib/libr,chdr,clib/libr,f77ots/libr
$ cc text
$ link text,plotlib/libr,math/libr,rlsclib/libr,chdr,clib/libr
$ !
$ ! cleanup
$ !
$ del *.obj.*
$ set on
$ purge
$ set def 'savedef'
```

B.3.12.1 [RLS.XYPLOT.SPOOLER]BUILD.COM -

```
$ savedef:='f$directory()'
$ set noon
$ set def [rls.xyplot.spooler]
$ libr /create=blocks:1 plotdev
$ if target .eqs. "VMS" then @[.vms]build
$ if target .eqs. "VMS" then @[.rsx]build
$ !
$ ! functions
$ !
$ cc bensoni
$ cc calcomp
$ cc eprplot
$ cc lxy
$ cc ploti
$ cc tek4006
$ cc tek4014
$ cc tek4662
$ cc vt100
$ cc vt125
$ libr /rep plotdev bensoni,calcomp,eprplot,lxy
$ libr /rep plotdev ploti,tek4006,tek4014,tek4662
$ libr /rep plotdev vt100,vt125
$ !
$ ! cleanup
$ !
$ del *.obj.*
$ purge
$ set def 'savedef'
```

B.3.12.1.1 [RLS.XYPLOT.SPOOLER.RSX]BUILD.COM -

```
$ savedef:='f$directory()'
$ set def [rls.xyplot.spooler.rsx]
$ !
$ ! functions
$ !
$ for copyfll
$ libr /rep plotdev copyfll
$ !
$ ! cleanup
$ !
$ del *.obj.*
$ purge
$ set def 'savedef'
```

B.3.12.1.2 [RLS.XYPLOT.SPOOLER.VMS]BUILD.COM -

```
$ savedef:='f$directory()'
$ set def [rls.xyplot.spooler.vms]
$ !
$ for benson
$ for copyfile
$ macro nargs
$ libr /rep plotdev benson,copyfile,nargs
$ !
$ ! cleanup
$ !
$ del *.obj.*
$ purge
$ set def 'savedef'
```

B.4 DEVELOPMENT AREAS

The remaining areas are those which are still under development or which have become obsolete due to changes in operation, or changes in software requirements.

B.4.1 CC.COM (obsolete)

```
$ ver = 'f$verify(0)
$ exitstatus = 1
$ ! This is a test version.  To make it present rather than new all
$ ! references to sys$new: should be replaced with the appropriate
$ ! directory.
$ ! To be executed as a command procedure the following command line
$ ! should be executed:
$ ! cc :== @sys$system:cc c
$ ! Modification history:
$ ! 1982-Aug-13 ARC
$ ! Made into ordinary c compiler.  references to sys$new: changed and
$ ! appropriate files copied over.
$
$ if target .nes. "CMPARC" .and. target .nes. "VMS" then goto RSX
$ cpp := $sys$system:cpp
$ cp1 := $sys$system:cp1
$ cp2 := $sys$system:cp2
$ goto L1
$ RSX:
$ cpp := $sys$system:cpp11 -x
$ cp1 := $sys$system:cp111 -o
$ cp2 := $sys$system:cp211 -o
$ L1:
$
$ ! compile C programs.
$ !
$       on CONTROL_Y then goto CLEANUP
$       on SEVERE_ERROR then goto CLEANUP
$       libnam := [rls.library]
$       cppflags := ""
$       cp1flags := ""
$       syslibflag := 0
$       paramindex = 0
$       paramlimit = 9  ! [sic]
$ LOOP:
$       paramindex = paramindex + 1
$       if paramindex .eq. paramlimit then goto COMPILE
$       if p'paramindex .eqs. "" then goto COMPILE
$       pn := "'p''paramindex''"
$       pn := 'pn
$       if paramindex .ne. 1 then goto NOFUDGE
$ !     This is because this proc. allows has one parameter of length 1.
$       len = 'f$length(pn)
$       if len .eq. 1 then goto LOOP
$       pn := 'f$extract(1, len, pn)
$ NOFUDGE:
$
$       if 'f$locate("/", pn) .ne. 'f$length(pn) then goto FLAGS
$       files := 'pn
$       goto LOOP
$
$ FLAGS:
$       len = 'f$length(pn)
$       if len .eq. 0 then goto LOOP
$
$ ! extract the flag letter and any filenames and dispatch to appropriate place
$
$       slash = 'f$locate("/", pn)
$       if slash .ne. 0 then files := 'f$extract(0, slash, pn)
$       slash = slash + 1
$       pn := 'f$extract(slash, len, pn)
$       slash = 'f$locate("/", pn)
$       flag := 'f$extract(0, 1, pn)
$       if flag .eqs. "M" then goto CP1FLAGS
$       if flag .eqs. "D" .or. flag .eqs. "I" .or. flag .eqs. "P" .or. flag-
         .eqs. "S" then goto CPPFLAGS
$       write sys$error "Unknown switch at \", pn, "\"
$       exitstatus = 4
$       goto LEAVE
$
$ CP1FLAGS:
$       cp1flags := "-m"
$       goto ENDFLAG
$
$ CPPFLAGS:
$       flagstr := 'f$extract(0, slash, pn)
$       flagstr := 'f$extract(1, slash, flagstr)
$       if flag .eqs. "I" then goto CPPIFLAG
$ CPPLOOP:
$       if 'f$length(flagstr) .eq. 0 then goto ENDFLAG
$       comma = 'f$locate(",", flagstr)
$       cppflags := 'cppflags'" -"'flag''f$extract(0, comma, flagstr)'
$       comma = comma + 1
$       flagstr := 'f$extract(comma, len, flagstr)
$       goto CPPLOOP
$
$ CPPIFLAG:
$       if syslibflag .eq 0 then cppflags := 'cppflags'" -I"'libnam'
$       syslibflag := 1
$ CPPILOOP:
$       if 'f$length(flagstr) .eq. 0 then goto ENDFLAG
$       comma = 'f$locate(",", flagstr)
$       cppflags := 'cppflags'"|"'f$extract(0, comma, flagstr)
$       comma = comma + 1
$       flagstr := 'f$extract(comma, len, flagstr)
$       goto CPPILOOP
$
$ ENDFLAG:
```

```
$       pn := 'f$extract(slash, len, pn)
$       goto FLAGS
$
$ COMPILE:
$       if syslibflag .eq. 0 then cppflags := 'cppflags'" -I"'libnam'
$ COMPILELOOP:
$       len = 'f$length(files)
$       if len .eq. 0 then goto CLEANUP
$       comma = 'f$locate(",", files)
$       pn := 'f$extract(0, comma, files)
$       comma = comma + 1
$       files := 'f$extract(comma, len, files)
$       dot = 'f$locate(".",pn)
$       basename := 'f$extract(0,dot,pn)
$ !
$ ! errorout' is expanded to ">>filename" or null (default)
$ !
$       cnam := 'basename'.c
$       if dot .eq. 'f$length(pn) then pn := 'cnam'
$       marnam := 'basename'.mar
$       write sys$output pn
$ DISPATCH:
$       on SEVERE_ERROR then goto NOTE_ERROR
$       if pn .eqs. cnam then goto DOTC
$       if pn .eqs. marnam then goto DOTMAR
$ DOTC:
$ if target .nes. "CMPARC" .and. target .nes. "VMS" then goto RSX2
$       cpp -x -dVMS -d'target' -dvax$vms 'cppflags' -o cpp.tmp 'pn' 'errorout'
$ ! use -n32 in cp1 for calling vms system functions
$       cp1 'cp1flags' -n31 -lb0 -o cp1.tmp cpp.tmp   'errorout'
$       cp2 -o cp2.tmp cp1.tmp   'errorout'
$ goto RSX3
$ RSX2:
$       cpp -x 'cppflags' -d RSX -d 'target' -o cpp.tmp 'pn' 'errorout'
$       cp1 'cp1flags' -n6 -o cp1.tmp cpp.tmp 'errorout'
$       cp2 -f -o cp2.tmp cp1.tmp 'errorout'
$ RSX3:
$ DOTMAR:
$       marnam := cp2.tmp
$       macro/object='basename' 'marnam'
$       on SEVERE_ERROR then goto CLEANUP
$       goto COMPILELOOP
$ NOTE_ERROR:
$       exitstatus = 4
$       goto COMPILELOOP
$
$ CLEANUP:
$       open/error=NOCPP tmpfi:  cpp.tmp
$       close tmpfi:
$       del cpp.tmp;*
$ NOCPP:
$       open/error=NOCP1 tmpfi: cp1.tmp
$       close tmpfi:
$       del cp1.tmp;*
$ NOCP1:
```

```
$       open/error=NOCP2 tmpfi: cp2.tmp
$       close tmpfi:
$       del cp2.tmp;*
$ NOCP2:
$ LEAVE:
```

B.4.2 [RLS.CLIB.TEST]BUILD.COM

```
$ !
$ ! data conversion
$ !
$ cc btod,btoi,btol,btos
$ clibr /repl [.'pl']clib btod,btoi,btol,btos
$ cc decode,dtento,dtoe,dtof
$ clibr /repl [.'pl']clib decode,dtento,dtoe,dtof
$ cc encode,errfmt
$ clibr /repl [.'pl']clib encode,errfmt
$ !
$ ! file i/o routines
$ !
$ cc fclose,fcreate,fill
$ cc finit,fopen,fread
$ cc getc,getch,getf,getfmt,getl,getlin
$ cc putc,putch,putf,putfmt,putl,putlin,putstr
$ clibr /repl [.'pl']clib fclose,fcreate,fill
$ clibr /repl [.'pl']clib finit,fopen,fread
$ clibr /repl [.'pl']clib getc,getch,getf,getfmt,getl,getlin
$ clibr /repl [.'pl']clib putc,putch,putf,putfmt,putl,putlin,putstr
$ !
$ ! string parsing
$ !
$ cc cmpbuf,cmpstr
$ cc inbuf,instr,itob,itols
$ cc notbuf,notstr,prefix
$ clibr /repl [.'pl']clib cmpbuf,cmpstr
$ clibr /repl [.'pl']clib inbuf,instr,itob,itols
$ clibr /repl [.'pl']clib notbuf,notstr,prefix
$ !
$ ! buffer manipulation
$ !
$ cc cpybuf,cpystr
$ cc lenstr,lower,lstoi,ltob
$ clibr /repl [.'pl']clib cpybuf,cpystr
$ clibr /repl [.'pl']clib lenstr,lower,lstoi,ltob
$ !
$ ! low level i/o
$ !
$ !
$ ! program control
$ !
$ cc main
$ clibr /repl [.'pl']clib main
$ cc scnbuf,scnstr,squeeze
$ cc stdin,stdout
$ cc stob,subbuf,substr
$ cc uname
!
! programs
!
$ cc clibtst
```

B.4.3 [RLS.DEVICE.RSX.ALSARC]BUILD.COM (obsolete)

```
$ savedef:='f$directory()'
$ set def [rls.device.rsx.alsarc]
$ !
$ ! BD: driver for the DR-11K digital i/o board
$ !
$ mcr mac bddrv=exemc/ml,rsxmc/pa:1,sy:bddrv
$ mcr tkb
bddrv.tsk/-hd/-mm,,bddrv=bddrv
rsx11m/ss
exelib/lb
/
stack=0
par=drvpar:120000:20000
//
$ !
$ ! cleanup
$ !
$ del *.obj.*
$ purge
$ set def 'savedef'
```

B.4.4 [RLS.DEVICE.RSX.CHMARC]BUILD.COM (obsolete)

```
$ savedef:='f$directory()'
$ set def [rls.device.rsx.chmarc]
$ !
$ ! LS: driver for LPS-11
$ !
$ mcr mac lsdrv=exemc/ml,rsxmc/pa:1,sy:lsdrv
$ mcr tkb
lsdrv.tsk/-hd/-mm,,lsdrv=lsdrv
rsx11m/ss
exelib/lb
/
stack=0
par=drvpar:120000:20000
//
$ !
$ ! cleanup
$ !
$ del *.obj.*
$ purge
$ set def 'savedef'
```

B.4.5 [RLS.RELATION.TEST]BUILD.COM

```
$ savedef:='f$directory()'
$ set def [rls.relation]
$ !
$ ! subroutines/functions
$ !
$ cc addext
$ cc closer
$ cc creatr
$ cc deltuple
$ cc gettuple
$ cc instuple
$ cc openr
$ cc puttuple
$ cc rel
$ cc rpltuple
$ libr /rep rlsclib addext,closer,creatr,deltuple,gettuple,instuple
$ libr /rep rlsclib openr,puttuple,rel,rpltuple
$ !
$ ! cleanup
$ !
$ del *.obj.*
$ purge
$ set def 'savedef'
```

B.4.6 [RLS.TRANSPORT]BUILD.COM

```
$ develop chmarc
$ cc binary
$ mcr tkb
binary/fp/cp=binary,voltage
rlsclib/lb
chdr,clib/lb
/
stack=3000
units=10
actfil=10
asg=sy:1:2:3:4:6:7:8
reslib=fcsres/ro
//
$ cc lparda
$ mcr tkb
lparda/fp/cp,lparda/-sp=lparda,lpaemu,cdigio
rlsclib/lb
chdr,clib/lb
/
stack=2000
units=32
actfil=32
reslib=fcsres/ro
pri=100
//
```