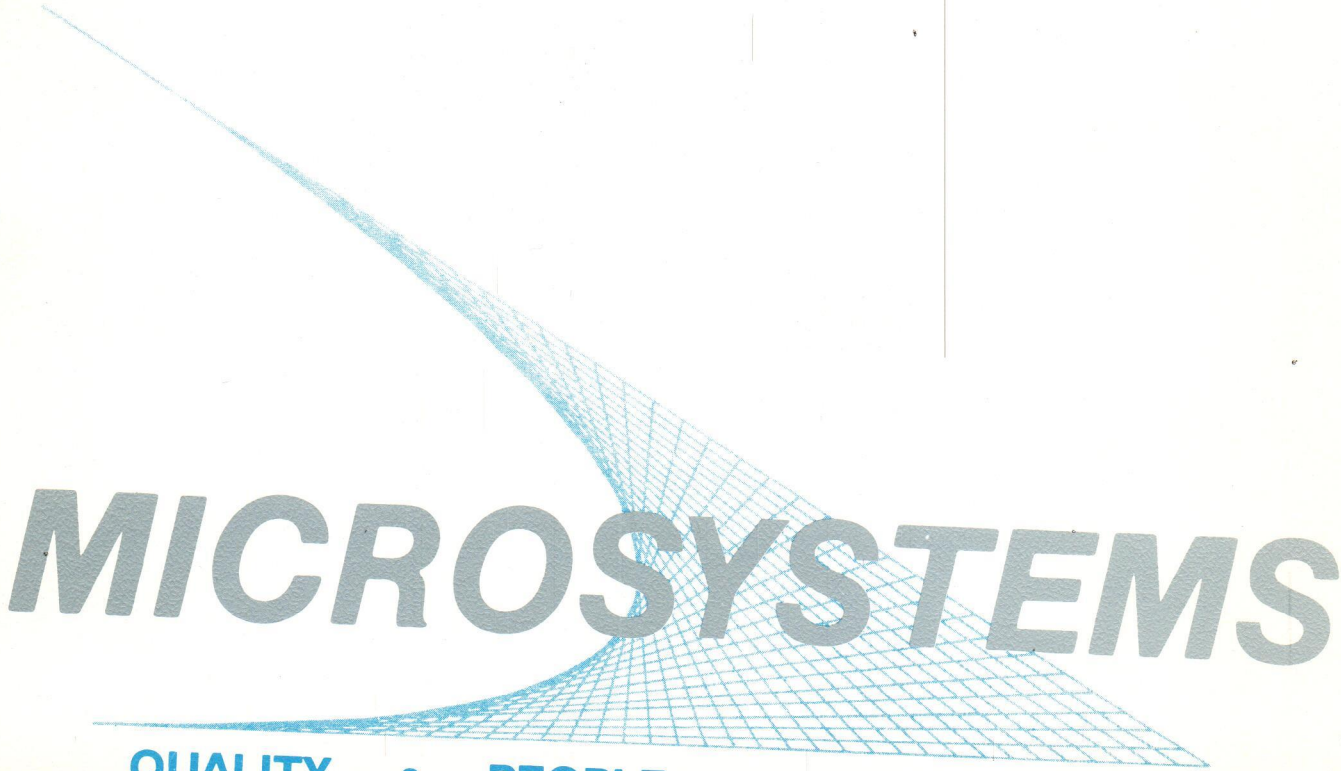
MOTOROLA

M68KVPRTL/D1

VERSAdos Pascal Resident Run-Time Library User's Manual



QUALITY • PEOPLE • PERFORMANCE

M68KVPRTL/D1

JULY 1985

VERSAdos PASCAL RESIDENT

RUN-TIME LIBRARY

USER'S MANUAL

The information in this document has been carefully checked and is believed to be entirely reliable. However, no responsibility is assumed for inaccuracies. Furthermore, Motorola reserves the right to make changes to any products herein to improve reliability, function, or design. Motorola does not assume any liability arising out of the application or use of any product or circuit described herein; neither does it convey any license under its patent rights or the rights of others.

RSM68K and VERSAdos are trademarks of Motorola Inc.

First Edition

Copyright 1985 by Motorola Inc.

CHAPTER 1

INTRODUCTION

1.1 GENERAL INFORMATION

The VERSAdos Resident Run-Time Library (RRTL) is a collection of routines that exists as an extension of the VERSAdos operating system. The routines provide an interface between programs written in PASCAL and the VERSAdos operating system. The Pascal language runs under VERSAdos and is targeted for VERSAdos.

1.2 CONVENTIONS

The following conventions are used in the command syntax, examples, and text in this manual:

boldface strings	A boldface string is a literal, such as a command or program name, and is to be typed just as it appears.
italic strings	An italic string is a "syntactic variable" and is to be replaced by one of a class of items it represents.
|	A vertical bar separating two or more items indicates that a choice is to be made; only one of the items separated by this symbol should be selected.
[ ]	Square brackets enclose an item that is optional. The item may appear zero or one time.
[ ] . . .	Square brackets followed by an ellipsis (three dots) enclose an item that is optional/repetitive. The item may appear zero or more times.
[ ]	Boldface brackets are required characters.

Operator inputs are to be followed by a carriage return. The carriage return is shown as (CR) only if it is the only input required.

1.3 RELATED DOCUMENTATION

The following publications may provide additional helpful information. If not shipped with this product, they may be obtained from Motorola's Literature Distribution Center, 616 West 24th Street, Tempe, AZ 85282; telephone (602) 994-6561.

DOCUMENT TITLE	MOTOROLA PUBLICATION NUMBER
M68000 Family Real-Time Multitasking Software User's Manual	M68KRMS68K
M68000 Family Resident Pascal *User's Manual*	M68KPASC
M68000 Family VERSAdos System Facilities Reference Manual	M68KVSF
VERSAdos Data Management Services and Program Loader User's Manual	RMS68KIO
Pascal Programming Structures for Motorola Microprocessors	TB304/D

The *FORTRAN 77 Reference Manual Revision 2.0* is available from the manufacturer, ABSOFT Corporation, Royal Oak, Michigan.

CHAPTER 2

FUNCTIONAL DESCRIPTION

2.1 SOFTWARE INTERFACES

Software interfaces are needed to provide the VERSAdos RRTL with access to the RMS68K and VERSAdos directives. A language interface for Pascal is also required.

2.1.1 RMS68K Interface

RMS68K consists of an inner kernel (or nucleus) that supports the priority-driven, multitasking environment, and eight resource managers. Each resource manager consists of data structures and from five to seven RMS68K directives, with each directive providing a specific service from the resource manager. The eight resource managers are:

a. Event Management Directives
b. Event Management Directives
c. Memory Management Directives
d. Task Management Directives
e. Time Management Directives
f. Semaphore Management Directives
g. Trap Server Management Directives
h. Exception Monitor Management Directives
i. Exception Management Directives

Entry to the RMS68K directives is through a TRAP #1 instruction. Directive calls require a numeric value (directive number) in data register D0. Most directive calls also require the address of a parameter block in address register A0. On return from the directive call, data register D0 contains status information. The status register is also set to reflect the contents of D0.

Appendix A contains a complete listing of the RMS68K directives that are supported by the VERSAdos Resident Run-Time Library. A full discussion of the RMS68K directives is provided in the *M68000 Family Real-Time Multitasking Software User's Manual*. The Channel Management Request (CMR) directive is documented in the *Guide to Writing Device Drivers for VERSAdos*.

2.1.2 VERSAdos Directives

The VERSAdos directives are divided into three logical groups:

a. Input/Output Service (IOS)
b. File Handling Services (FHS)
c. Loader Directives

Each group of directives requires a specific interface. In addition, a software interface for the Error Message Handler (EMH) program is also required.

2.1.2.1 Input/Output Services Interface. I/O operations within VERSAdos are essentially device-independent. Operations are based on logical properties, not device characteristics or file formats. Logical Unit Numbers (LUNs) are assigned to devices and files before I/O between programs and files/devices occurs.

In VERSAdos, all devices and files are treated as files. I/O is handled by two modules, IOS and FHS. IOS handles all data transfers, referring to task or user identification and LUN.

Entry to the IOS directives is through a TRAP #2 instruction. IOS directive calls require the address of a parameter block in address register A0. The IOS function code is specified in the first word of the parameter block. On return from the directive call, data register D0 reflects the contents of D0. The PROCEED call is an exception. The D0 does not reflect the status of the function since the function has not yet completed. The parameter block status byte will reflect the status of the function after the function has completed.

Refer to the *VERSAdos Data Management Services and Program Loader User's Manual* for further information.

2.1.2.2 File Handling Services Interface. As part of I/O, FHS is called into service when creating disk files and their attributes, associating a LUN with each device or file.

Entry to the FHS directives is through a TRAP #3 instruction. FHS directive calls require the address of a parameter block in address register A0. The FHS function code is specified in the first word of the parameter block. On return from the directive call, data register D0 contains the status returned from the function. The parameter block status byte contains the function status. The status register is also set to reflect the contents of D0. Some directive calls also return data in address register A0 or address registers A0 and A1.

Refer to the *VERSAdos Data Management Services and Program Loader User's Manual* for further information.

2.1.2.3 Loader Interface. The program loader's function is to:

a. Create a new task

b. Allocate memory segments for the task based on segment information found in the Loader Information Block (LIB) of a load file created by the linkage editor

c. Read the contents of each segment from the load file into the segments allocated

The task created by the loader is in a dormant state following a successful completion of that load function.

Entry to the Loader directives is through a TRAP #4 instruction. The loader call requires the address of a parameter block in address register A0. The directive number for the loader (1) is contained in data register D0. On return from the directive call, data register D0 contains the status returned from the loader.

The loader command is thoroughly described in the *VERSAdos Data Management Services and Program Loader User's Manual.*

2.1.2.4 Error Message Handler Interface. The EMH program is a system server task that provides standard error message displays in response to exception conditions, relieving the user task of maintaining its own error message list. The program retrieves the key value of the message to be displayed from the ERRORMSG.SY file, expanding any embedded sentinels.

Entry to the EMH is through a TRAP #4 instruction. The EMH call requires the address of a parameter block in address register A0. The directive number for EMH (2) is contained in data register D0. On return from the directive call, data register D0 contains the status returned from EMH.

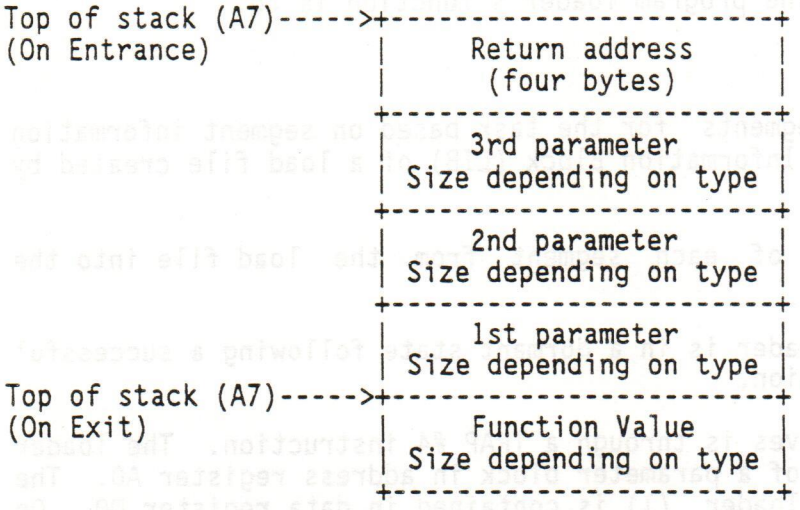
Refer to the *M68000 VERSAdos System Facilities Reference Manual* for a complete description of the EMH program.

2.1.3 Pascal Interface

The Pascal run-time library consists of routines written in assembly language that can be declared and called from the Pascal programs as FUNCTIONs. In every case, the function value returned is the trap status information, the contents of D0.

The format of a library routine call from a Pascal program is identical to a regular function or procedure call. Parameters are placed on the top of the stack beneath the return address, in the order in which they are declared. The size of the parameters depends on the parameter type. The return value for a function is below the first parameter on the stack.

Figure 2-1 illustrates the stack as it would appear on entry to a library routine called as a function with three parameters:



Register contents on entrance to a library routine are as follows:

A3 = base address of the libraries
A5 = pointer to base of local variable area
A6 = pointer to base of global variable area
A7 = stack pointer

FIGURE 2-1. Stack on Entry to a Library Routine.

The library routines are required by the compiler to preserve the value of registers A3, A5, and A6. They are also responsible for removing from the stack all parameters passed to it from the calling program.

Refer to the *M68000 Family Resident* Pascal *User's Manual* for a complete description of the stack entries created by the various types of parameters.

CHAPTER 3

ACCESS TO SYSTEM DIRECTIVES

3.1 GENERAL INFORMATION

The RRTL routines provide an interface between programs written in Pascal and the VERSAdos operating system. This chapter describes the means to access the RMS68K and VERSAdos directives.

Pascal RRTL routines are called from the applications program as functions. Like any Pascal function, two items must be defined for each RRTL directive used: the function's declaration and the function's calling sequence.

3.2 Pascal DECLARATIONS

Each VERSAdos directive used in the Pascal program requires a declaration statement. The general description of a Pascal declaration for a RRTL function is as follows:

FUNCTION *name* (*parameter_list*) : **INTEGER; FORWARD;**

where:

name = the name of the directive
parameter_list = the list of parameters required by that directive.

3.3 Pascal RRTL CALLING SEQUENCE

A function call for the RRTL functions can be used anywhere a Pascal function can be used. The RRTL functions return the directive status as a integer value. For the directives that do not return status, the returned value of the function is 0.

The general form of the RRTL calling sequence is as follows:

name(*p1*,*p2*,...,*pn*);

Descriptions of the Pascal declarations and calling sequences for the RRTL functions are provided in Appendix A.

3.4 CONSTRUCTING A Pascal TASK INCLUDING THE RRTL

The chain file that follows contains the steps necessary to compile and link a task that includes the RRTLs in a sharable segment with a major portion of the Pascal library.

Notice that although the routines RTLINIT, RFINIT, RTRAPS, and PLJSR are Pascal library routines, the routines are not shareable. Therefore, in the chain file, the routines have been separated from the remainder of the Pascal library to permit the library to be shared by other tasks.

```
=/* COMPILE MASTER PROGRAM
=Pascal \1,\2,\2.LX;Z=100
=Pascal2 \2,\2,\2.LS;LZ=100
=/* LINK TOGETHER
=LINK ,\2,\2;MSX
 SEG SEG1(R):0,9 $0000
 SEG SEG2:15
 SEG RRTL(GR):8
 IN 9998.RRTL.RTLINIT<INIT>
 IN \2
 IN 9998.RRTL.RRTLACCS
 IN 9998.RRTL.PRTL
 IN 9998.RRTL.PLJSR
 IN 9998.RRTL.RFINIT
 IN 9998.RRTL.RTRAPS
 IN 9998.RRTL.RPSCALIB
 ATTR P
 END
=END
```

where:

\1	= the name of the Pascal source file
\2	= the name of the resultant linked file
RTLINIT	= a replacement for the Pascal INIT routine for use with the RRTLs
RTTLACCS	= the RRTL access routines
PRTL	= the RRTL routines
PLJSR	= Pascal library routine that cannot be shared
RFINIT	= Pascal library routine that cannot be shared
RTRAPS	= Pascal library routine that cannot be shared
RPSCALIB	= Pascal library that is shareable

CHAPTER 4

RRTL SOURCE FILES

4.1 GENERAL INFORMATION

It is helpful to understand the construction of the RRTL's source files before attempting to create a VERSAdos run-time library. The RRTL consists of two major modules: one module which contains the library routines, (PRTL); and one module which contains the library access routines, (RTLACCS).

The library access routines are necessary to overcome the compiler restrictions and maintain a globally sharable run-time library. Any library routines that are added to or deleted from the RRTL require changes to both modules. Appendix B describes the structure of both modules.

The VERSAdos RRTL's design enables users to construct three types of libraries: a full standard library; a subset of the standard library; or a customized library in which new routines have been added or unnecessary routines deleted. A full standard library is one that contains all VERSAdos interface routines (grouped by directives: RMS68K, IOS, and FHS/LOADER). A subset would be a library that contains some combination of these groups. A customized library might contain only those directives a user required, plus library routines that the user created.

4.2 BUILDING A STANDARD RUN-TIME LIBRARY

To build a standard run-time library, the user specifies first, the language that the library supports, and second, the subsets of directives that are desired for the library. A full standard library contains all subsets of directives.

4.2.1 Specifying the Language

To specify Pascal as the language supported by the library, enter the following command while assembling the RRTL:

```
ASM 9998.RRTL.RRTLPASC/9998.RRTL.RRTLSRC,PRTL,PRTL;RMD
```

The files RRTLPASC and RRTLABSF contain the equate statements that designate Pascal as the support language.

4.2.2 Specifying a Standard Subset

The file RRTLIDS.EQ contains the library scope equates, one equate for each directive subset. To include a particular directive subset in the library, the equate is set to 1. To omit a particular subset from the library, the equate is set to 0:

0 = Omit the group of directives
1 = Include the group of directives

For example, a full standard library that includes all directive subsets would be represented in the RRTLIDS.EQ file as follows:

```
S$RMS    EQU    1      RMS68K directives
S$IOS    EQU    1      IOS directives
S$FHS    EQU    1      FHS directives
S$GEN    EQU    1      GENERIC Trap routines
```

4.3 BUILDING A CUSTOMIZED RUN-TIME LIBRARY

Users can customize the run-time library either by creating their own library routines, or deleting all routines that are not required.

4.3.1 Deleting Directive Routines from the RRTL

To customize a library so that it contains only those routines that are required, do the following:

STEP 1. Modify the file RRTLIDS.EQ to indicate the directive subsets that are to be omitted completely from the library. For example, if no FHS directives are required, set the equate S$FHS to 0. All references to the FHS directives are excluded from the library modules.

STEP 2. Exclude individual routines from the library by deleting references to the routine from the following modules:

Library Routine Source File (one of the following):

```
9998.RRTL.SRCIOS.SA   for IOS routines
9998.RRTL.SRCFHS.SA   for FHS routines
9998.RRTL.SRCRMS.SA   for RMS68K routines
9998.RRTL.SRCGEN.AI   for generic trap routines
```

Access Routine Source File (one of the following):

```
9998.RRTL.ACSIOS.AI   for IOS routines
9998.RRTL.ACSFHS.AI   for FHS routines
9998.RRTL.ACSRMS.AI   for RMS68K routines
9998.RRTL.ACSGEN.AI   for generic trap routines
```

Vector Table:

9998.RRTL.RRTLSRC.SA

Access Routine External Definition File:

9998.RRTL.ACSXDEF.AI

Access Code Definition File:

9998.RRTL.ACSCODES.AI

STEP 3. After the source files are edited, create the library by executing both of the following chain files:

9998.RRTL.RTL.CF (RRTL routines)
9998.RRTL.ACS.CF (Access routines)

Descriptions of both chain files are contained in Appendix C.

4.3.2 Adding Directive Routines to the Run-Time Library

To create library routines and include them with the standard RRTL routines, do the following:

STEP 1. Create the library routine.

Carefully study the information in Chapter 2 that describes the Pascal calling conventions. Pay particular attention to the registers that must be preserved for the languages and the parameter passing mechanisms.

STEP 2. Update the RRTL source file.

Source code for newly created library routine can be incorporated as an include file, merged into one of the existing include files, or merged into RRTLSRC.SA.

STEP 3. Add an entry for the routine to the external reference file, ACSXDEF.AI.

STEP 4. Add an entry for the routine to the vector table located in RRTLSRC.SA.

STEP 5. Add an entry for the routine to the access code module, ACSCODES.AI. This entry and the entry in the vector table must correspond.

STEP 6. After the source files are modified, create the library by executing the following chain files:

9998.RRTL.RTL.CF (RRTL routines)
9998.RRTL.ACS.CF (Access routines)

THIS PAGE INTENTIONALLY LEFT BLANK.

APPENDIX A

RESIDENT RUN-TIME LIBRARY DIRECTIVE DESCRIPTIONS

This appendix contains a description of each directive included in the RRTL. The descriptions are organized by directive group:

Paragraph A.1 - RMS68K directives
Paragraph A.2 - IOS directives
Paragraph A.3 - FHS directives
Paragraph A.4 - Generic trap routines

A.1 RMS68K DIRECTIVES

The RMS68K directive descriptions have been organized into eight sub-groups, following the format used in the *M68000 Family Real-Time Multitasking Software User's Manual*. The sub-groups are defined as follows:

Paragraph A.1.1 - Event Management Directives
Paragraph A.1.2 - Memory Management Directives
Paragraph A.1.3 - Task Management Directives
Paragraph A.1.4 - Time Management Directives
Paragraph A.1.5 - Semaphore Management Directives
Paragraph A.1.6 - Trap Server Management Directives
Paragraph A.1.7 - Exception Monitor Management Directives
Paragraph A.1.8 - Exception Management Directives

Within these groups, the directives are arranged in alphabetical order. The single driver directive is described in:

Paragraph A.1.9 - Driver Directive

A.1.1 Event Management Directives

a. **LDEASQ** - Deallocate Asynchronous Service Queue (ASQ)

Directive number = 32

The LDEASQ library routine provides access to the RMS68K directive DEASQ. The Deallocate-ASQ function frees the memory dedicated to the requestor's ASQ.

Call Line: RVL = LDEASQ()

Where: RVL = Returned Value (Always zero)

Declaration:

```
function ldeasq
                : integer; forward;
```

b. **LGTASQ** - Allocate ASQ

Directive number = 31

The LGTASQ library routine provides access to the RMS68K directive GTASQ. The Allocate-ASQ function allocates memory for the target task's ASQ.

Call Line: RVL = LGTASQ(TSK,SES,ASQS,MXL,QLN,VEC,BFR,PBA)

Where:
- RVL = Returned Value (Status Code)
- TSK = Task Id
- SES = Session number
- ASQS = ASQ Status
- MXL = Maximum Message Length
- QLN = Que Length
- VEC = ASR Service Vector
- BFR = Receiving Buffer
- PBA = Parameter Block Address

Declaration:

```
function  lgtasq(tsk  : integer;
                 ses  : integer;
                 asqs : byte;
                 mxl  : byte;
                 qln  : integer;
                 vec  : integer;
                 bfr  : integer;
                 : integer; forward;
```

c. **LGTEVNT** - Get event from its ASQ

Directive number = 38

The LGTEVNT library routine provides access to the RMS68K directive GTEVNT. The Get-Event function moves the oldest event sent to the task to the receive buffer. If no event exists in the task's ASQ, the task goes into a wait-for-event state.

Call Line: RVL = LGTEVNT(RCV)

Where: RVL = Returned Value (Status Code)
RCV = Address of receiving area

Declaration:

```
function  lgtevnt(rcv : integer)
                  : integer; forward;
```

d. **LQEVNT** - Queue event to task's ASQ

Directive number = 35

The LQEVNT library routine provides access to the RMS68K directive QEVNT. The Queue-Event function places the specified event in the ASQ of the target task, or moves the event directly into the target task's default buffer.

Call Line: RVL = LQEVNT(TSK,SES,OPT,EAD,ALV,PBA)

Where: RVL = Returned Value (Status Code)
TSK = Task Id
SES = Session number
OPT = Options
EAD = Event Address
ALV = Alternate Service Vector
PBA = Parameter Block Address

Declaration:

```
function  lqevnt(tsk  : integer;
                 ses  : integer;
                 opt  : word;       (Directive options)
                 ead  : integer;    (Event address)
                 alv  : integer;    (Alternate service vector)
                 pba  : integer)
                : integer; forward;
```

e. **LRDEVNT** - Task reads event from its ASQ

Directive number = 34

The LRDEVNT library routine provides access to the RMS68K directive RDEVNT. The Read-Event moves the oldest event sent to the task to the receive buffer. If no event exists, the first two bytes of the receive buffer are set to zero.

Call Line: RVL = LRDEVNT(RCV)

Where: RVL = Returned Value (Status Code)
RCV = Address of receiving area

Declaration:

```
function  lrdevnt(rcv : integer)
                  : integer; forward;
```

f. **LRTEVNT** - ASR returns after event servicing

Directive number = 37

The LRTEVNT library routine provides access to the RMS68K directive RTEVNT. The Return-from-Event function restores the environment and returns control to the point where the event interrupt occurred.

Call Line: RVL = LRTEVNT(ASRO)

Where: RVL = Returned Value (Always zero)
ASRO = ASR options

Declaration:

```
function lrtevnt(asrs : integer)
                 : integer; forward;
```

g. **LSETASQ** - Task changes its ASQS/ASR status

Directive number = 33

The LSETASQ library routine provides access to the RMS68K directive SETASQ. The Set-ASQ function replaces the requesting task's current ASQ, ASR and default-receive-buffer status with the requested status.

Call Line: RVL = LSETASQ(ASRO)

Where: RVL = Returned Value (Status Code)
ASRO = ASR options

Declaration:

```
function  lsetasq(asqs : integer)
                  : integer; forward;
```

h. **LWTEVNT** - Task moves itself to Wait-for-Event state

Directive number = 36

The LWTEVNT library routine provides access to the RMS68K directive WTEVNT. The Wait-for-Event function ensures that the ASQ and ASR of the requesting task are enabled and places the task in the Wait-for-Event state.

Call Line: RVL = LWTEVNT()

Where: RVL = Returned Value (Status Code)

Declaration:

```
function  lwtevnt
                  : integer; forward;
```

A.1.2 Memory Management Directives

a. **LATTSEG** - Attach a shareable segment

Directive number = 4

The LATTSEG library routine provides access to the RMS68K directive ATTSEG. The Attach-a-Shareable-Segment function allows the requesting task control over the logical beginning address of the segment.

Call Line: RVL = LATTSEG(OPT,ATR,SGN,SGA,SGL,ADR,PBA)

Where: RVL = Returned Value (Status Code)
OPT = Options
ATR = Segment Attribute
SGN = Segment name
SGA = Physical or logical addr of new segment
SGL = Segment Length
ADR = Physical address returned from trap
PBA = Parameter Block Address

Declaration:

```
function lattseg(opt : word;
                 atr : word;
                 sgn : array[1..4] of char;
                 sga : integer;
                 sgl : integer;
             var adr : integer;
                 pba : integer)
               : integer; forward;
```

b. **LDCLSHR** - Declare a segment shareable

Directive number = 7

The LDCLSHR library routine provides access to the RMS68K directive DCLSHR. The Declare-a-Segment-Shareable function makes a non-shareable segment (contained within the address space of the requesting task) into a shareable segment so that more than one task may attach to it.

Call Line: RVL = LDCLSHR(OPT,ATR,SGN,PBA)

Where: RVL = Return Value
OPT = Options
ATR = Segment Attribute
SGN = Segment name
PBA = Parameter Block Address

Declaration:

```
function  ldclshr(opt : word;
                  atr : word;
                  sgn : array[1..4] of char;
                  pba : integer)
                : integer; forward;
```

c. **LDESEG** - Deallocate a segment

Directive number = 2

The LDESEG library routine provides access to the RMS68K directive DESEG. The Deallocate-a-Segment function deletes the specified segment from the target task's address space.

Call Line: RVL = LDESEG(TSK,SES,OPT,SGN,PBA)

Where: RVL = Returned Value (Status Code)
TSK = Task Id
SES = Session number
OPT = Options
SGN = Segment name
PBA = Parameter Block Address

Declaration:

```
function  ldeseg(tsk : integer;
                 ses : integer;
                 opt : word;
                 sgn : array[1..4] of char;
                 pba : integer)
                : integer; forward;
```

d. **LGTSEG** - Allocate a segment

Directive number = 1

The LGTSEG library routine provides access to the RMS68K directive GTSEG. The Allocate-a-Segment function enables a task to obtain a named segment of memory for itself or another task.

Call Line: RVL = LGTSEG(TSK,SES,OPT,ATR,SGN,SGA,SGL,ADR,SZ,PBA)

Where:
- RVL = Returned Value (Status Code)
- TSK = Task Id
- SES = Session number
- OPT = Options
- ATR = Segment Attribute
- SGN = Segment name
- SGA = Physical or logical addr of new segment
- SGL = Segment Length
- ADR = Physical address returned from trap
- SZ = Size of largest free block available
- PBA = Parameter Block Address

Declaration:

```
function  lgtseg(tsk : integer;
                 ses : integer;
                 opt : word;
                 atr : word;
                 sgn : array[1..4] of char;
                 sga : seg_addr;
                 sgl : integer;
             var adr : integer;
             var sz  : integer;
                 pba : integer)
                : integer; forward;
```

e. **LSHRSEG** - Grant shared segment access

Directive number = 5

The LSHRSEG library routine provides access to the RMS68K directive SHRSEG. This directive places an existing shareable segment within another task's address space.

Call Line: RVL = LSHRSEG(TSK,SES,OPT,ATR,SGN,SGA,SGL,ADR,PBA)

Where:
- RVL = Returned Value (Status Code)
- TSK = Task Id
- SES = Session number
- OPT = Options
- ATR = Segment Attribute
- SGN = Segment name
- SGA = Physical or logical addr of new segment
- SGL = Segment Length
- ADR = Physical address returned from trap
- PBA = Parameter Block Address

Declaration:

```
function  lshrseg(tsk : integer;
                  ses : integer;
                  opt : word;
                  atr : word;
                  sgn : array[1..4] of char;
                  sga : seg_addr;
                  sgl : integer;
              var adr : integer;
                  pba : integer)
                : integer; forward;
```

f. **LTRSEG** - Transfer segment

Directive number = 3

The LTRSEG library routine provides access to the RMS68K directive TRSEG. The Transfer-Segment directive removes a segment from the requesting task's address space and places it within the address space of another task.

Call Line: RVL = LTRSEG(TSK,SES,OPT,ATR,SGN,SGA,ADR,PBA)

Where: RVL = Returned Value (Status Code)
TSK = Task Id
SES = Session number
OPT = Options
ATR = Segment Attribute
SGN = Segment name
SGA = Physical or logical addr of new segment
ADR = Physical address returned from trap
PBA = Parameter Block Address

Declaration:

```
function  ltrseg(tsk : integer;
                 ses : integer;
                 opt : word;
                 atr : word;
                 sgn : array[1..4] of char;
                 sga : seg_addr;
             var adr : integer;
                 pba : integer)
                : integer; forward;
```

g. **LRCVSA** - Receive segment attributes

Directive number = 9

The LRCVSA library routine provides access to the RMS68K directive RCVSA. The Receive-Segment-Attributes directive returns a description of the specified segment to the requesting task.

Call Line: RVL = LRCVSA(TSK,SES,OPT,SGN,SGA,BFR,PBA)

Where: RVL = Returned Value (Status Code)
TSK = Task Id
SES = Session number
OPT = Options
SGN = Segment Name
SGA = Physical or logical addr of new segment
BFR = Address of buffer containing segment information
PBA = Parameter Block Address

Declaration:

```
function  lrcvsa(tsk : integer;
                 ses : integer;
                 opt : word;
                 sgn : array[1..4] of char;
                 sga : seg_addr;
                 bfr : integer;
                 pba : integer)
                : integer; forward;
```

h. **LMOVELL** - Move from logical address

Directive number = 6

The LMOVELL library routine provides access to the RMS68K directive MOVELL. The Move-Logical-Address directive requests that data be copied from the logical address space of one task to the logical address space of another task.

Call line: RVL = LMOVELL(TSK,SES,LADR,DTSK,DSES,DADR,LEN,PBA)

Where:
RVL = Returned Value (Status Code)
TSK = Task Id
SES = Session number
LADR = Source Logical Address
DTSK = Destination Taskname
DSES = Destination Session number
DADR = Destination Logical Addresses
LEN = Length of the data block
PBA = Parameter Block Address

Declaration:

```
function  lmovell(tsk  : integer;
                  ses  : integer;
                  ladr : integer;
                  dtsk : integer;
                  dses : integer;
                  dadr : integer;
                  len  : integer;
                  pba  : integer)
                 ; integer; forward;
```

i. **LMOVEPL** - Move from physical address

Directive number = 72

The LMOVEPL library routine provides access to the RMS68K directive MOVEPL. The Move-from-Physical-Address directive requests that data be copied from any physical address to a logical address within a target task's address space.

Call line: RVL = LMOVEPL(PADR,DTSK,DSES,DADR,LEN,PBA)

Where: RVL = Returned Value (Status Code)
PADR = Source Physical Address
DTSK = Destination Taskname
DSES = Destination Session number
DADR = Destination Logical Addresses
LEN = Length of the data block
PBA = Parameter Block Address

Declaration:

```
function  lmovepl(padr : integer;
                  dtsk : integer;
                  dses : integer;
                  dadr : integer;
                  len  : integer;
                  pba  : integer)
                 : integer; forward;
```

j. **LFLUSHC** - Flush user cache

Directive number = 75

The LFLUSHC library routine provides access to the RMS68K directive FLUSHC. The Flush-User-Cache directive flushes all mode entries from all cache known to the Executive.

Call Line: RVL = LFLUSHC()

Where: RVL = Returned Value (Always zero)

Declaration:

```
function  lflushc
                 : integer; forward;
```

A.1.3 Task Management Directives

a. **LABORT** - Task aborts itself

Directive number = 14

The LABORT library routine provides access to the RMS68K directive ABORT. The Abort-Self directive halts the execution of the requesting task and removes the task from memory.

Call Line: RVL = LABORT(ABC)

Where: RVL = Returned Value (Always zero)
ABC = Abort Code

Declaration:

```
function labort(abc : word)
                : integer; forward;
```

b. **LCRTCB** - Create Task Control Block (TCB)

Directive number = 11

The LCRTCB library routine provides access to the RMS68K directive CRTCB. The Create-TCB directive allocates memory for the TCB and initializes it with information from the parameter block.

Call Line: RVL = LCRTCB(TSK,SES,OPT,MNAM,MSES,IPR,LPR,ATR,ENT,UID,PBA)

Where: RVL = Returned Value (Status Code)
TSK = Task Id
SES = Session number
OPT = Options
MNAM = Monitor Task Name
MSES = Monitor Session Number
IPR = Initial Priority
LPR = Limit Priority
ATR = Task Attributes
ENT = Task Entry Point
UID = User Generated Id
PBA = Parameter Block Address

Declaration:

```
function  lcrtcb(tsk   : integer;
                 ses   : integer;
                 opt   : word;
                 mngm  : integer;
                 mses  : integer;
                 ipr   : byte;
                 lpr   : byte;
                 atr   : word;
                 ent   : integer;
                 uid   : word;
                 pba   : integer)
                : integer; forward;
```

c. **LGTSKID** - Get task ID

Directive number = 12

The LGTSKID library routine provides access to the RMS68K directive GTTASKID. In response to the input of a taskname and session number, the executive returns the target task ID.

Call Line: RVL = LGTSKID(NAME,RSES,RTSK,PBA)

Where: RVL = Returned Value (Status Code)
NAME = TSK,SES
TSK = Task ID
SES = Session Number
RSES = Session Number (Returned)
RTSK = Task ID (Returned)
PBA = Parameter Block Address

Declaration:

```
function  lgtskid(tsk   : integer;
                  ses   : integer;
              var rtsk  : integer;
              var rses  : integer;
                  pba   : integer)
                 : integer; forward;
```

d. **LGTTSKNAM** - Get taskname

Directive number = 10

The LGTTSKNAM library routine provides access to the RMS68K directive GTTASKNM. The executive returns the taskname and session number of the target task when the task ID is entered.

Call Line: RVL = LGTTSKNAM(TSK,SES,RTSK,PBA)

Where: RVL = Returned Value (Status Code)
TSK = Task ID
SES = Session Number
RTSK = Task ID (Returned)
RSES = Session Number (Returned)

Declaration:

```
function  lgttsknam(tsk   : integer;
                    ses   : integer;
                var rtsk  : integer;
                var rses  : integer;
                    pba   : integer)
                  : integer; forward;
```

e. **LRELINQ** - Relinquish execution

Directive number = 22

The LRELINQ library routine provides access to the RMS68K directive RELINQ. This directive permits a task to relinquish control of the processor to tasks of equal or slightly lower priority.

Call Line: RVL = LRELINQ()

Where: RVL = Returned Value (Status Code)

Declaration:

```
function  lrelinq
                  : integer; forward;
```

f. **LRESUME** - Resume target task

Directive number = 18

The LRESUME library routine provides access to the RMS68K directive RESUME. The executive resumes execution of a previously suspended task.

Call Line: RVL = LRESUME(TSK,SES,PBA)

Where: RVL = Returned Value (Status Code)
TSK = Task ID
SES = Session Number
PBA = Parameter Block Address

Declaration:

```
function lresume(tsk : integer;
                 ses : integer;
                 pba : integer)
               : integer; forward;
```

g. **LSETPRI** - Change priority of a task

Directive number = 24

The LSETPRI library routine provides access to the RMS68K directive SETPRI. The executive changes the current priority of the target task to the value specified.

Call Line: RVL = LSETPRI(TSK,SES,NCPR,LPR,PBA)

Where: RVL = Returned Value (Status Code)
TSK = Task ID
SES = Session Number
NCPR = New Current Priority
LPR = Limit Priority (Returned)
PBA = Parameter Block Address

Declaration:

```
function lsetpri(tsk  : integer;
                 ses  : integer;
                 ncpr : byte;
                 lpr  : byte;
                 pba  : integer)
               : integer; forward;
```

h. LSTART - Start task

Directive number = 13

The LSTART library routine provides access to the RMS68K directive START. The executive puts the target task into the READY state, based on its current priority, to wait for execution.

Call Line: RVL = LSTART(TSK,SES,OPT,MNAM,MSES,RADR,NAME,PBA)

Where: RVL = Returned Value (Status Code)
TSK = Task ID
SES = Session Number
OPT = Options
MNAM = Taskname
MSES = Monitor Session
RADR = Address of Register Area
NAME = Taskname of Started Task (Returned)
PBA = Parameter Block Address

Declaration:

```
function  lstart(tsk   : array[1..4] of char;
                 ses   : integer;
                 opt   : word;
                 mtsk  : integer;
                 mnam  : integer;
             var radr  : res_data_area   (shown below)
             var name  : array[1..4] of char;
                 pba   : integer)
                : integer; forward;
```

Record - register setup area

```
reg_data_area =
 record
   regd0 : integer;                    { taskname }
   regd1 : integer;                    { session number }
   regd2 : array[1..4] of char;        { user volume }
   regd3 : integer;                    { user number }
   regd4 : array[1..4] of char;        { catalog (chars 1-4) }
   regd5 : array[1..4] of char;        { catalog (chars 5-8) }
   regd6 : integer;                    { command line length }
   regd7 : integer;                    { LUN assignment bit map }
   rega0 : integer;
   rega1 : array[1..4] of char;        { default terminal id }
   regas : array[1..6] of integer
 end;
```

i. **LSTOP** - Stop task

Directive number = 25

The LSTOP library routine provides access to the RMS68K directive STOP. The executive stops execution of the target task and moves it to the DORMANT state with all resources still attached.

```
Call Line:  RVL  = LSTOP(TSK,SES,NAM,PBA)

    Where:  RVL  = Returned Value (Status Code)
            TSK  = Task ID
            SES  = Session Number
            NAM  = Taskname (of Stopped Task)
            PBA  = Parameter Block Address
```

Declaration:

```
 function  lstop(tsk : integer;
                 ses : integer;
             var nam : array[1..4] of char;
                 pba : integer)
               : integer; forward;
```

j. **LSUSPEND** - Suspend task

Directive number = 17

The LSUSPEND library routine provides access to the RMS68K directive SUSPEND. The executive places the requesting task into the WAIT state until a WAKEUP directive is issued by another task.

```
Call Line:  RVL  = LSUSPEND()

    Where:  RVL  = Returned Value (Status Code)
```

Declaration:

```
 function  lsuspend
                 : integer; forward;
```

k. LTERM - Task terminates itself

Directive number = 15

The LTERM library routine provides access to the RMS68K directive TERM. The executive halts execution of the requesting task and removes the task from memory.

Call Line: RVL = LTERM()

Where: RVL = Returned Value (Status Code)

Declaration:

```
function  lterm
          : integer; forward;
```

l. LTERMT - Terminate target task

Directive number = 16

The LTERMT library routine provides access to the RMS68K directive TERMT. The executive halts execution of the target task and removes the task from memory.

Call Line: RVL = LTERMT(TSK,SES,OPT,ABC,NAM,PBA)

Where: RVL = Returned Value (Status Code)
TSK = Task ID
SES = Session Number
OPT = Options
ABC = Abort Code
NAM = Name of Terminated Task
PBA = Parameter Block Address

Declaration:

```
function  ltermt(tsk : array[1..4] of char;
                 ses : integer;
                 opt : word;
                 abc : word;
             var nam : array[1..4] of char;
                 pba : integer)
                : integer; forward;
```

m. **LWAIT** - Task moves itself to WAIT state

Directive number = 19

The LWAIT library routine provides access to the RMS68K directive WAIT. The executive places the requesting task into the WAIT state until a WAKEUP directive is issued by another task.

Call Line: RVL = LWAIT()

Where: RVL = Returned Value (Status Code)

Declaration:

```
function  lwait
                  : integer; forward;
```

n. **LWAKEUP** - Wakeup target task

Directive number = 20

The LWAKEUP library routine provides access to the RMS68K directive WAKEUP. The executive moves the specified target task from the WAIT state to the READY state to await execution.

Call Line: RVL = LWAKEUP(TSK,SES,PBA)

Where: RVL = Returned Value (Status Code)
TSK = Task ID
SES = Session Number
PBA = Parameter Block Address

Declaration:

```
function  lwakeup(tsk : array[1..4] of char;
                  ses : integer;
                  pba : integer)
                 : integer; forward;
```

o. **LTSKATTR** - Receive task user number and attributes

Directive number = 23

The LTSKATTR library routine provides access to the RMS68K directive TSKATTR. The executive returns the target task's user number and attributes to the requestor.

Call line: RVL = LTSKATTR(TSK,SES,USRN,ATTR,PBA)

Where: RVL = Returned Value (Status Code)
TSK = Task ID
SES = Session Number
USRN = User Number
ATTR = User Attributes
PBA = Parameter Block Address

Declaration:

```
function  ltskattr(tsk  : integer;
                   ses  : integer;
               var usrn : word;
               var attr : word;
                   pba  : integer)
                 : integer; forward;
```

p. **LTSKINFO** - Receive copy of TCB

Directive number = 28

The LTSKINFO library routine provides access to the RMS68K directive TSKINFO. The executive moves a copy of the target task's TCB to the requestor's address space.

Call line: RVL = LTSKINFO(TSK,SES,OPT,BADR,PBA)

Where: RVL = Returned Value (Status Code)
TSK = Task ID
SES = Session Number
OPT = Options
BADR = Buffer Address
PBA = Parameter Block Address

Declaration:

```
function  ltskinfo(tsk  : integer;
                   ses  : integer;
                   opt  : word;
                   badr : integer;
                   pba  : integer)
                 : integer; forward;
```

A.1.4 Time Management Directives

a. **LDELAY** - Task moves itself to delay state

Directive number = 21

The LDELAY library routine provides access to the RMS68K directive DELAY. The executive delays the execution of the requesting task until a specified amount of time elapses.

Call Line: RVL = LDELAY(DLY)

Where: RVL = Returned Value (Status Code)
DLY = Length of time to delay

Declaration:

```
function  ldelay(dly : integer)
                 : integer; forward;
```

b. **LDLAYW** - DELAY, WTEVNT, and WAIT functions are performed

Directive number = 30

The LDLAYW library routine provides access to the RMS68K directive DLAYW. The executive delays the execution of the requesting task until one of the following events occurs: a specified amount of time elapses; an asynchronous event arrives; or a WAKEUP is sent to the waiting task.

Call Line: RVL = LDLAYW(DLY)

Where: RVL = Returned Value (Status Code)
DLY = Length of Time to Delay

Declaration:

```
function  ldlayw(dly : integer)
                 : integer; forward;
```

c. **LRQSTPA** - Request periodic activation

Directive number = 29

The LRQSTPA library routine provides access to the RMS68K directive RQSTPA. The executive activates the target task at an initial time and at optional intervals.

```
Call Line:  RVL = LRQSTPA(TSK,SES,OPT,TIME,INTV,SADR,RQID,PBA)

    Where:  RVL  = Returned Value (Status Code)
            TSK  = Task ID
            SES  = Session Number
            OPT  = Options
            TIME = Initial Time
            INTV = Interval
            SADR = Service Address
            RQID = Activation Request ID
            PBA  = Parameter Block Address

Declaration:

 function  lrqstpa(tsk  : integer;
                   ses  : integer;
                   opt  : word;
                   time : integer;
                   intv : integer;
                   sadr : integer;
                   rqid : integer;
                   pba  : integer)
                  : integer; forward;
```

d. **LGTDTIM** - Get date and time

Directive number = 74

The LGTDTIM library routine provides access to the RMS68K directive GTDTIM. The executive places the current system date and time into the specified return parameter block.

```
Call line:  RVL = LGTDTIM(CDAT,CTIM,PBA)

    Where: RVL  = Returned Value (Status Code)
           CDAT = Current System Date
           CTIM = Current System Time
           PBA  = Parameter Block Address

Declaration:

 function  lgtdtim(var cdate : integer;
                   var ctime : integer;
                       pba : integer)
                  : integer; forward;
```

e. **LSTDTIM** - Set system date and time

Directive number = 73

The LSTDTIM library routine provides access to the RMS68K directive STDTIM. The executive updates the system date and time.

Call line: RVL = LSTDTIM(NDAT,NTIM,PBA)

Where: RVL = Returned Value (Status Code)
NDAT = New System Date
NTIM = New System Time
PBA = Parameter Block Address

Declaration:

```
function  lstdtim(cdate : integer;
                  ctime : integer;
                  pba   : integer)
                : integer; forward;
```

A.1.5 Semaphore Management Directives

a. **LATSEM** - Attach to semaphore

Directive number = 41

The LATSEM library routine provides access to the RMS68K directive ATSEM. The executive allows the requesting task to use the specified semaphore.

Call Line: RVL = LATSEM(SNAM,STYP,KEY,PBA)

Where: RVL = Returned Value (Status Code)
SNAM = Semaphore Name
STYP = Semaphore Type
KEY = Semaphore Key
PBA = Parameter Block Address

Declaration:

```
function latsem(snam : array[1..4] of char;
                styp : byte;
            var key : integer;
                pba : integer)
              : integer; forward;
```

b. LCRSEM - Create semaphore

Directive number = 45

The LCRSEM library routine provides access to the RMS68K directive CRSEM. The executive creates or re-initializes the specified semaphore, and allows the requesting task to use it.

Call Line: RVL = LCRSEM(SNAM,ICNT,STYP,KEY,PBA)

Where: RVL = Returned Value (Status Code)
SNAM = Semaphore Name
ICNT = Initial Count
STYP = Semaphore Type
KEY = Semaphore Key
PBA = Parameter Block Address

Declaration:

```
function lcrsem(snam : array[1..4] of char;
                icnt : byte;
                styp : byte;
             var key : integer;
                pba  : integer)
               : integer; forward;
```

c. LDESEM - Detach from semaphore

Directive number = 44

The LDESEM library routine provides access to the RMS68K directive DESEM. The executive detaches the requesting task from the specified semaphore.

Call Line: RVL = LDESEM(SNAM,SKEY,PBA)

Where: RVL = Returned Value (Status Code)
SNAM = Semaphore Name
SKEY = Semaphore Key
PBA = Parameter Block Address

Declaration:

```
function ldesem(snam : array[1..4] of char;
                skey : integer;
                pba  : integer)
              : integer; forward;
```

d. **LDESMA** - Detach from all semaphores

Directive number = 46

The LDESMA library routine provides access to the RMS68K directive DESMA. The executive detaches the requesting task from all semaphores.

Call Line: RVL = LDESMA()

Where: RVL = Returned Value (Status Code)

Declaration:

```
function ldesma(: integer; forward;
```

e. **LSGSEM** - Signal semaphore

Directive number = 43

The LSGSEM library routine provides access to the RMS68K directive SGSEM. The executive increments the current signal count by 1. If the count is 0 or negative, the first task on the semaphore waiting list is removed from the list and placed in the ready list to await execution.

Call Line: RVL = LSGSEM(SNAM,SKEY,PBA)

Where: RVL = Returned Value (Status Code)
SNAM = Semaphore Name
SKEY = Semaphore Key
PBA = Parameter Block Address

Declaration:

```
function lsgsem(snam : array[1..4] of char;
                skey : integer;
                pba  : integer)
                : integer; forward;
```

f. LWTSEM - Wait on semaphore

Directive number = 42

The LWTSEM library routine provides access to the RMS68K directive WTSEM. The executive decrements the current signal count of the specified semaphore by 1. If the count is 0 or positive, the requesting task continues executing. If the count is negative, the requesting task is added to the semaphore waiting list.

Call Line: RVL = LWTSEM(SNAM,SKEY,PBA)

Where: RVL = Returned Value (Status Code)
SNAM = Semaphore Name
SKEY = Semaphore Key
PBA = Parameter Block Address

Declaration:

```
function lwtsem(snam : array[1..4] of char;
                skey : integer;
                pba  : integer)
               : integer; forward;
```

A.1.6 Trap Server Management Directives

a. LAKRQST - Server acknowledge request

Directive number = 54

The LAKRQST library routine provides access to the RMS68K directive AKRQST. The executive moves the target task from the waiting-on-acknowledgement list to the state indicated by the server-request-pending state.

Call Line: RVL = LAKRQST(TSK,SES,OPT,TRP,CCOD,RA0,RD0,PBA)

Where: RVL = Returned Value (Status Code)
TSK = Task ID
SES = Session Number
OPT = Options
TRP = Trap Number
CCOD = Condition Code
RA0 = Register A0
RD0 = Register D0
PBA = Parameter Block Address

Declaration:

```
function  lakrqst(tsk  : integer;
                  ses  : integer;
                  opt  : word;
                  trp  : byte;
                  ccod : byte;
                  ra0  : integer;
                  rd0  : integer;
                  pba  : integer)
                : integer; forward;
```

b. **LDERQST** - Set user/server request status

Directive Number = 53

The LDERQST library routine provides access to the RMS68K directive DEASQ.

Call Line: RVL = LDERQST(TNBR)

Where: RVL = Returned Value (Status Code)
TNBR = Trap Number

Declaration:

**** Entry at LNRQST for Enable Request Receipt

```
function lnrqst(tnbr : byte)
                : integer; forward;
```

**** Entry at LDRQST for Disable Request Receipt

```
function ldrqst(tnbr : byte)
                : integer; forward;
```

c. **LDSERVE** - Detach server function

Directive number = 52

The LDSERVE library routine provides access to the RMS68K directive DSERVE. A server task initiates an orderly shutdown of service.

Call Line: RVL = LDSERVE(TRP)

Where: RVL = Returned Value (Status Code)
TRP = Trap Number

Declaration:

```
function ldserve(trp : byte)
                 : integer; forward;
```

d. **LSERVER** - Task is made a server task

Directive number = 51

The LSERVER library routine provides access to the RMS68K directive SERVER. The executive establishes the requesting task as a server task of the trap instruction specified in the parameter block.

Call Line: RVL = LSERVER(RADR,STAT,TRAP,PBSZ,PBA)

Where: RVL = Returned Value (Status Code)
RADR = Request Service Address
STAT = Status
TRAP = Trap Instruction ID
PBSZ = Parameter Block Size
PBA = Parameter Block Address

Declaration:

```
function lserver(radr : integer;
                 stat : byte;
                 trap : byte;
                 pbsz : byte;
                 pba  : integer)
                : integer; forward;
```

A.1.7 Exception Monitor Management Directives

a. **LDEXMON** - Detach exception monitor

Directive number = 65

The LDEXMON library routine provides access to the RMS68K directive DEXMON. The executive detaches the target task from its exception monitor. The target task then resumes normal activity according to its current state.

Call Line: RVL = LDEXMON(TSK,SES,PBA)

Where: RVL = Returned Value (Status Code)
TSK = Task Id
SES = Session number
PBA = Parameter Block Address

Declaration:

```
function  ldexmon(tsk  : integer;
                  ses  : integer;
                  pba  : integer)
                 : integer; forward;
```

b. **LEXMMSK** - Set exception monitor mask

Directive number = 66

The LEXMMSK library routine provides access to the RMS68K directive EXMMSK. The specified exception monitor mask is attached to the target task. When an enabled exception occurs within the target task, the target task is placed in the wait-for-command state and an appropriate message is queued to the target task's exception monitor.

Call Line: RVL = LEXMMSK(TSK,SES,XMSK,PBA)

Where: RVL = Returned Value (Status Code)
TSK = Task ID
SES = Session Number
XMSK = Exception Monitor Mask
PBA = Parameter Block Address

Declaration:

```
function  lexmmsk(tsk  : integer;
                  ses  : integer;
                  xmsk : integer;
                  pba  : integer)
                : integer; forward;
```

c. **LEXMON** - Attach exception monitor

Directive number = 64

The LEXMON library routine provides access to the RMS68K directive EXMON. The executive attaches the target task to the exception monitor task and places the target task in the wait-for-command state. An event, indicating the attach, is queued to the exception monitor.

Call Line: RVL = LEXMON(TSK,SES,XTSK,XSES,PBA)

Where: RVL = Returned Value (Status Code)
TSK = Task ID
SES = Session Number
XTSK = Exception Monitor Task ID
XSES = Exception Monitor Session Number
PBA = Parameter Block Address

Declaration:

```
function  lexmon(tsk  : integer;
                 ses  : integer;
                 xtsk : integer;
                 xses : integer;
                 pba  : integer)
               : integer; forward;
```

d. LPSTATE - Modify task state

Directive number = 68

The LPSTATE library routine provides access to the RMS68K directive PSTATE. An exception monitor can change the state of a target task by changing the values of the target task's data registers, address registers, user stack pointer, program counter, status register and exception monitor mask.

Call Line: RVL = LPSTATE(TSK,SES,BADR,PBA)

Where: RVL = Returned Value (Status Code)
TSK = Task ID
SES = Session Number
BADR = Buffer Address
PBA = Parameter Block Address

Declaration:

```
function lpstate(tsk  : integer;
                 ses  : integer;
             var badr : new_state_info  (shown below)
                 pba  : integer)
               : integer; forward;
```

Record - new state information buffer

```
new_state_info =
  record
    regd0 : integer;
    regd1 : integer;
      .   :    .
    regd7 : integer;
    rega0 : integer;
      .   :    .
    rega7 : integer;
    pc    : integer;
    sr    : word
  end;
```

e. **LREXMON** - Run task under exception monitor control

Directive number = 69

The LREXMON library routine provides access to the RMS68K directive REXMON. An exception monitor task specifies how a target task is to be executed.

Call Line: RVL = LREXMON(TSK,SES,BADR,PBA)

Where: RVL = Returned Value (Status Code)
TSK = Task ID
SES = Session Number
BADR = Buffer Address
PBA = Parameter Block Address

Declaration:

```
function  lrexmon(tsk  : integer;
                  ses  : integer;
              var badr : ex_cntl_info   (shown below)
                  pba  : integer)
                : integer; forward;
```

Record - execution control information

```
exec_cntrl_info =
  record
    xopt : word;     { execution options}
    vloc : integer;  { value location }
    val  : integer;  { value }
    vmsk : integer;  { value mask }
    mxic : integer   { maximum instruction count }
  end;
```

f. **LRSTATE** - Receive task state

Directive number = 67

The LRSTATE library routine provides access to the RMS68K directive RSTATE. An exception monitor receives the current state of a target task.

Call Line: RVL = LRSTATE(TSK,SES,BADR,PBA)

Where: RVL = Returned Value (Status Code)
TSK = Task ID
SES = Session Number
BADR = Buffer Address
PBA = Parameter Block Address

Declaration:

```
function  lrstate(tsk  : integer;
                  ses  : integer;
              var badr : record        (shown below)
                  pba  : integer)
                : integer; forward;
```

Record - receive state information buffer

```
receive_state_info =
  record
    regd0 : integer;
    regd1 : integer;
      .   :   .
    regd7 : integer;
    rega0 : integer;
      .   :   .
    rega7 : integer;
    pc    : integer;
    sr    : word
  end;
```

A.1.8 Exception Management Directives

a. **LCISR** - Configure Interrupt Service Routine (ISR)

Directive number = 61

The LCISR library routine provides access to the RMS68K directive CISR.

Call Line: RVL = LCISR(TSK,SES,OPT,VEC,IADR,ARG,PBA)

Where:
RVL = Returned Value (Status Code)
TSK = Task ID
SES = Session Number
OPT = Options
VEC = Vector Number
IADR = Interrupt Service Routine Address
ARG = User Defined Value
PBA = Parameter Block Address

Declaration:

```
function  lcisr(tsk  : integer;
                ses  : integer;
                opt  : word;        { Directive options }
                vec  : byte;        { Vector number }
                iadr : integer;     { ISR address }
                arg  : integer;     { Argument }
                pba  : integer)
              : integer; forward;
```

b. **LSINT** - Simulate interrupt

Directive number = 62

The LSINT library routine provides access to the RMS68K directive SINT.

Call Line: RVL = LSINT(PRTY,VEC,PBA)

Where: RVL = Returned Value (Status Code)
PRTY = Interrupt Priority
VEC = Vector Number
PBA = Parameter Block Address

Declaration:

```
function lsint(prty : byte;          { Interrupt priority }
               vec  : byte;          { Vector number }
               pba  : integer)
              : integer; forward;
```

c. **LCXVCT** - Change exception vector

Directive number = 26

The LCXVCT library routine is provided as an alternative to the announce-exception-vector directive. Pascal already has issued this directive and established an exception vector table. The LCXVCT routine provides the user with a convenient way of changing one of the exception vectors by modifying the already established vector table.

Input parameters
- Exception Vector Table index
  ix = 1 Bus Error
  2 Address Error
  3 Illegal Instruction
  4 Zero divide
  5 CHK instruction
  6 Trap V instruction
  7 Privilege Violation
- New Exception Vector

Return parameters
Status value = 0 for ix's 3 and 7.
< 0 for ix's <1 or >7
= 1 for ix's 1,2,4,5 and 6 (Pascal uses these.)

```
Call Line:  RVL = LCXVCT(XVCTX,NUXVEC)

    Where:  RVL    = Returned Value (Status Code)
            XVCTX  = Exception Vector Table Index
            NUXVEC = New exception vector

Declaration:

 function  lcxvct(xcxvct : byte;
                  nuxvec : integer;
                 : integer; forward;
```

d. **LCTVCT** - Change trap vector

Directive Number = 27

The LCTVCT library routine is provided as an alternative to the announce-trap-vector directive. Pascal already has issued this directive and established a trap vector table. The LCXVCT routine provides the user with a convenient way of changing one of the trap vectors by modifying the already established vector table.

Input parameters
- Trap Vector Table index
  2-15 for TRAPS #2 - #15
- New Trap Vector

Return parameters
Status value = 0 for normal return.
< 0 for ix's <2 or >15
= 1 for 13 or 14 since Pascal uses them.

```
Call Line:  RVL = LCTVCT(TVCTX,NUTVEC)

    Where:  RVL   = Returned Value (Status Code)
            TVCTX = Trap Vector Table Index
            NUTVEC= New Trap Vector

Declaration:

 function  lctvct(tvctx  : byte;
                  nutvec : integer;
                 : integer; forward;
```

e. **LCDIR** - Configure a new directive

Directive Number = 58

Call line: RVL = LNCDIR(DNBR,OPT,DRAD,PBA)
or
RVL = LDCDIR(DNBR,OPT,DRAD,PBA)

Where: RVL = Returned Value (Status Code)
DNBR = Directive Number
OPT = Options
DRAD = Directive Routine Address
PBA = Parameter Block Address

**** Entry at LNCDIR for Disable Directive

Declaration:

```
function  lncdir(dnbr : word;
                 opt  : word;
                 drad : integer;
                 pba  : integer
                ): integer; forward;
```

**** Entry at LDCDIR for Disable Directive

Declaration:

```
function  ldcdir(dnbr : word;
                 opt  : word;
                 drad : integer;
                 pba  : integer
                ): integer; forward;
```

f. **LSNAPTRAC** - Snap of system trace table

Directive Number = 8

The LSNAPTRAC library routine provides access to the RMS68K directive SNAPTRAC.

Call line: RVL = LSNAPTRAC(BADR)

Where: RVL = Returned Value (Status Code)
BADR = Buffer Address

Declaration:

```
function  lsnaptrac(badr : integer;
                    ): integer; forward;
```

A.1.9 Driver Directives

a. **LCMR** - Channel Management Requests (CMR)

Directive Number = 60

The LCMR library routine provides access to the RMS68K directive CMR. The executive invokes the CMR handler when a directive 60 is issued.

Call Line: RVL = LCMR(PBA)

Where: RVL = Returned Value (Status Code)
PBA = Parameter Block Address

Declaration:

```
function lcmr(pba : integer;
              : integer; forward;
```

A.2 IOS DIRECTIVES

The IOS directives have been organized into four directive types. The directives are described as follows:

Paragraph A.2.1 - Data Transfer Requests
Paragraph A.2.2 - Command Function Requests
Paragraph A.2.3 - Claim/Negate Driver Events
Paragraph A.2.4 - Privileged Requests

A.2.1 Data Transfer Requests

a. **LREAD** - Read request

Code = $00 Function = $01

The LREAD library routine provides access to the IOS directive READ.

Call Line: RVL = LREAD(LUN,RRN,ADR,LGN,LDT,OPT,CAD,PBA)

Where: RVL = Returned Value
LUN = Logical Unit (1 Byte)
RRN = Random Record Number
ADR = Buffer Address
LGN = Length of Data Buffer
LDT = Length of Data Transfer
OPT = Options
CAD = Completion Address
PBA = Parameter Block Address

Declaration:

```
function  lread(lun : byte;
                var rrn : integer;
                    adr : integer;
                    lgn : integer;
                var ldt : integer;
                    opt : word;
                    rtn : integer;
                    pba : integer)
                  : integer; forward;
```

b. **LWRITE** - Write request

Code = $00 Function = $02

The LWRITE library routine provides access to the IOS directive WRITE.

Call Line: RVL = LWRITE(LUN,RRN,ADR,LGN,LDT,OPT,CAD,PBA)

Where: RVL = Returned Value
LUN = Logical Unit (1 Byte)
RRN = Random Record Number
ADR = Buffer Address
LGN = Length of Data Buffer
LDT = Length of Data Transfer
OPT = Options
CAD = Completion Address
PBA = Parameter Block Address

Declaration:

```
function  lwrite(lun : byte;
                 var rrn : integer;
                     adr : integer;
                     lgn : integer;
                 var ldt : integer;
                     opt : word;
                     cad : integer;
                     pba : integer)
                   : integer; forward;
```

c. **LOUTIN** - Output with input request

Code = $00 Function = $04

The LOUTIN library routine provides access to the IOS directive OUTIN. This call allows a task to write to a device and at the same time issue a read for response. (The operation requires an interactive device that supports the call.)

Call Line: RVL = LOUTIN(LUN,SAD,ADR,LGN,LDT,OPT,CAD,PBA)

Where: RVL = Returned Value
LUN = Logical Unit (1 Byte)
RRN = Random Record Number
ADR = Buffer Address
LGN = Length of Data Buffer
LDT = Length of Data Transfer
OPT = Options
CAD = Completion Address
PBA = Parameter Block Address

Declaration:

```
function  loutin(lun : byte;
                 var sad : integer;
                     adr : integer;
                     lgn : integer;
                 var ldt : integer;
                     opt : word;
                     cad : integer;
                     pba : integer)
                   : integer; forward;
```

d. **LUPDATE** - Update record

Code = $00 Function = $08

The LUPDATE library routine provides access to the IOS directive UPDATE. The update-record function is valid only for an assignment to a non-contiguous file. Update-record must be used when changing an existing record in a file.

Call Line: RVL = LUPDATE(LUN,RRN,ADR,LGN,LDT,OPT,CAD,PBA)

Where: RVL = Returned Value
LUN = Logical Unit (1 Byte)
RRN = Random Record Number
ADR = Buffer Address
LGN = Length of Data Buffer
LDT = Length of Data Transfer
OPT = Options
CAD = Completion Address
PBA = Parameter Block Address

Declaration:

```
function lupdate(lun : byte;
                 var rrn : integer;
                     adr : integer;
                     lgn : integer;
                 var ldt : integer;
                     opt : word;
                     cad : integer;
                     pba : integer)
                   : integer; forward;
```

e. **LDELETE** - Delete record

Code = $00 Function = $10

The LDELETE library routine provides access to the IOS directive DELETE. The Delete-Record request is valid only for an assignment to an index sequential file.

Call Line: RVL = LDELETE(LUN,RRN,OPT,CAD,PBA)

Where: RVL = Returned Value
LUN = Logical Unit (1 Byte)
RRN = Random Record Number
OPT = Options
CAD = Completion Address
PBA = Parameter Block Address

Declaration:

```
function ldelete(lun : byte;
                 var rrn : integer;
                     opt : word;
                     cad : integer;
                     pba : integer)
                   : integer; forward;
```

f. **LFORMAT** - Format disk

Code = $00 Function = $20

The LFORMAT library routine provides access to the IOS directive FORMAT. The format request causes a disk or a track to be formatted.

Call Line: RVL = LFORMAT(LUN,PSN,OPT,PBA)

Where: RVL = Returned Value
LUN = Logical Unit (1 Byte)
PSN = Physical Sector Number
OPT = Options
PBA = Parameter Block Address

Declaration:

```
function lformat(lun : byte;
                 var psn : integer;
                     opt : word;
                     pba : integer)
                   : integer; forward;
```

g. **LTBRAK** - Transmit break

Code = $00 Function = $40

The LTBRAK library routine provides access to the IOS directive TBRAK. The Transmit-Break request, which applies only to interactive devices, sends a break to the logical unit specified.

Call Line: RVL = LTBRAK(LUN,OPT,PBA)

Where: RVL = Returned Value
LUN = Logical Unit (1 Byte)
OPT = Options
PBA = Parameter Block Address

Declaration:

```
function ltbrak(lun : byte;
                opt : word;
                pba : integer)
              : integer; forward;
```

A.2.2 Command Function Requests

a. **LPOSITION** - Position

Code = $01 Function = $01

The LPOSITION library routine provides access to the IOS directive POSITION.

Call Line: RVL = LPOSITION(LUN,RRN,OPT,PBA)

Where: RVL = Returned Value
LUN = Logical Unit (1 Byte)
RRN = Random Record Number
OPT = Options
PBA = Parameter Block Address

Declaration:

```
function lposition(lun : byte;
                   var rrn : integer;
                       opt : word;
                       pba : integer)
                      : integer; forward;
```

b. **LREWIND** - Rewind

Code = $01 Function = $02

The LREWIND library routine provides access to the IOS directive REWIND.

Call Line: RVL = LREWIND(LUN,RRN,OPT,PBA)

Where: RVL = Returned Value
LUN = Logical Unit (1 Byte)
RRN = Random Record Number
OPT = Options
PBA = Parameter Block Address

Declaration:

```
function lrewind(lun : byte;
                 var rrn : integer;
                     opt : word;
                     pba : integer)
                    : integer; forward;
```

c. **LTESTIO** - Test I/O complete

Code = $01 Function = $04

The LTESTIO library routine provides access to the IOS directive TESTIO. The Test-I/O-Complete call returns with a condition code of (Z bit = 1) if there is no outstanding I/O-proceed to the specified logical unit by the task.

Call Line: RVL = LTESTIO(LUN,PBA)

Where: RVL = Returned Value
LUN = Logical Unit (1 Byte)
PBA = Parameter Block Address

Declaration:

```
function ltestio(lun : byte;
                 pba : integer)
                : integer; forward;
```

d. **LWAITO** - WAIT Only

Code = $01 Function = $08

The LWAITO library routine provides access to the IOS directive WAITO. The WAIT-only request places the task into I/O WAIT until the completion of a previous I/O-proceed request to the specified logical unit.

Call Line: RVL = LWAITO(LUN,PBA)

Where: RVL = Returned Value
LUN = Logical Unit (1 Byte)
PBA = Parameter Block Address

Declaration:

```
function lwaito(lun : byte;
                pba : integer)
               : integer; forward;
```

e. **LHALTIO** - HALT I/O

Code = $01 Function = $10

The LHALTIO library routine provides access to the IOS directive HALTIO. A HALT-I/O request cancels an I/O-proceed request which has been previously issued.

Call Line: RVL = LHALTIO(LUN,PBA)

Where: RVL = Returned Value
LUN = Logical Unit (1 Byte)
PBA = Parameter Block Address

Declaration:

```
function lhaltio(lun : byte;
                 pba : integer)
                : integer; forward;
```

f. **LBRKSRV** - Break service

Code = $01 Function = $20

The LBRKSRV library routine provides access to the IOS directive BRKSRV. The Break-Service request applies only to interactive devices. When a break condition is present on the device specified, an attention event is sent to the requesting task.

Call Line: RVL = LBRKSRV(LUN,OPT,CAD,PBA)

Where: RVL = Returned Value
LUN = Logical Unit (1 Byte)
OPT = Options
CAD = Completion Address
PBA = Parameter Block Address

Declaration:

```
function lbrksrv(lun : byte;
                 opt : word;
                 cad : integer;
                 pba : integer)
                : integer; forward;
```

g. **LCONFST** - Configure status

Code = $01 Function = $40

The LCONFST library routine provides access to the IOS directive CONFST. The Configure-Status request may be executed by any task that has an assignment for the device for which the current configuration information is desired.

Call Line: RVL = LCONFST(LUN,OPT,CPB,PBA)

Where: RVL = Returned Value
LUN = Logical Unit (1 Byte)
OPT = Options
CPB = Configuration Block Address
PBA = Parameter Block Address

Declaration:

```
function lconfst(lun : byte;
                 opt : word;
                 cpb : integer;
                 pba : integer)
                : integer; forward;
```

h. **LCONFIG** - Configure

Code = $01 Function = $80

The LCONFIG library routine provides access to the IOS directive CONFIG. The Configure-Request may be executed by any task that has an assignment for the device to be configured.

Call Line: RVL = LCONFIG(LUN,OPT,CPB,PBA)

Where: RVL = Returned Value
LUN = Logical Unit (1 Byte)
OPT = Options
CPB = Configuration Block Address
PBA = Parameter Block Address

Declaration:

```
function lconfig(lun : byte;
                 opt : word;
                 cpb : integer;
                 pba : integer)
                : integer; forward;
```

A.2.3 Claim/Negate Driver Events

a. **LNEGBRK** - Negate break service

Code = $02 Function = $01

The LNEGBRK library routine provides access to the IOS directive NEGBRK. The Negate-Break-Service request applies only to interactive devices. A task that previously requested break service may obtain release from break-service responsibility via this request.

Call Line: RVL = LNEGBRK(LUN,PBA)

Where: RVL = Returned Value
LUN = Logical Unit (1 Byte)
PBA = Parameter Block Address

Declaration:

```
function lnegbrk(lun : byte;
                 pba : integer)
                : integer; forward;
```

A.2.4 Privileged Requests

a. **LCNFDEF** - Configure defaults

Code = $80 Function = $02

The LCNFDEF library routine provides access to the IOS directive CNFDEF. The configure-default request alters the default configuration of a device.

Call Line: RVL = LCNFDEF(LUN,OPT,CPB,PBA)

Where: RVL = Returned Value
LUN = Logical Unit (1 Byte)
OPT = Options
CPB = Configuration Block Address
PBA = Parameter Block Address

Declaration:

```
function lcnfdef(lun : byte;
                 opt : word;
                 cpb : integer;
                 pba : integer)
                : integer; forward;
```

A.3 FHS DIRECTIVES

The FHS directives have been organized into two types. These types are described as follows:

Paragraph A.3.1 - Device/File Commands
Paragraph A.3.2 - LDR Directive

A.3.1 Device/File Commands

a. **LCHKPT** - Checkpoint

Code = $00 Command = $01

The LCHKPT library routine provides access to the FHS directive CHKPT. The Checkpoint function empties the buffered FMS buffers by writing to the file, or by copying the buffers to the user's input buffer. For an indexed file, the function updates the directory entry.

Call Line: RVL = LCHKPT(LUN,PBA)

Where: RVL = Returned Value
LUN = Logical Unit
PBA = Parameter Block Address

Declaration:

```
function lchkpt(lun : byte;
                pba : integer)
                  : integer; forward;
```

b. **LFDELETE** - File delete

Code = $00 Command = $02

The LFDELETE library routine provides access to the FHS directive FDELETE. The Delete function deletes the file's directory entry by zeroing out the first character of the filename field, and releases the space on the disk previously occupied by the file.

Call Line: RVL = LFDELETE(FDES,WCOD,RCOD,PBA)

Where: RVL = Returned Value
FDES = File Descriptor
WCOD = Write Code
RCOD = Read Code
PBA = Parameter Block Address

Declaration:

```
function  lfdelete(
                var fdes : file_dscrpt;   { shown below}
                    wcod : byte;
                    rcod : byte;
                    pba  : integer)
                  : integer; forward;
```

File descriptor record

```
file_dscrpt  =
  record
    vol_id     : array[1..4] of char;
    user_nmbr  : word;
    catalog    : array[1..8] of char;
    file_name  : array[1..8] of char;
    extension  : array[1..2] of char
  end;
```

c. **LFCLOSE** - File close

Code = $00 Command = $04

The LFCLOSE library routine provides access to the FHS directive FCLOSE. The Close function discontinues an assigned logical connection between a task and a file or device.

Call Line: RVL = LFCLOSE(LUN,PBA)

Where: RVL = Returned Value
LUN = Logical unit
PBA = Parameter Block Address

Declaration:

```
function lfclose(lun : byte;
                 pba : integer)
                    : integer; forward;
```

d. **LPROTECT** - Protect

Code = $00 Command = $08

The LPROTECT library routine provides access to the FHS directive PROTECT. The Protect function changes an assigned file's access permission codes.

Call Line: RVL = LPROTECT(LUN,WCOD,RCOD,PBA)

Where: RVL = Returned Value
LUN = Logical Unit
WCOD = Write Code
RCOD = Read Code
PBA = Parameter Block Address

Declaration:

```
function  lprotect(lun : byte;
                   wcod : byte;
                   rcod : byte;
                   pba  : integer)
                  : integer; forward;
```

e. **LRENAME** - Rename

Code = $00 Command = $10

The LRENAME library routine provides access to the FHS directive RENAME. The Rename function changes an assigned filename.

Call Line: RVL = LRENAME(LUN,FDES,PBA)

Where: RVL = Returned Value
LUN = Logical Unit
FDES = File Descriptor Address
PBA = Parameter Block Address

Declaration:

```
function  lrename(lun : byte;
             var fdes : file_dscrpt;        (shown below)
                 pba  : integer)
                : integer; forward;
```

File descriptor record

```
file_dscrpt  =
  record
    vol_id      : array[1..4] of char;
    user_nmbr   : word;
    catalog     : array[1..8] of char;
    file_name   : array[1..8] of char;
    extension   : array[1..2] of char
  end;
```

f. **LCHGPERM** - Change access permission

Code = $00 Command = $20

The LCHGPERM library routine provides access to the FHS directive CHGPERM. The Change-Access-Permission function allows the user to change the current access permission of a file or device which is assigned.

Call Line: RVL = LCHGPERM(LUN,OPT,WCOD,RCOD,PBA)

Where: RVL = Returned Value
LUN = Logical Unit
OPT = Options
WCOD = Write Code
RCOD = Read Code
PBA = Parameter Block Address

Declaration:

```
function  lchgperm(lun : byte;
                   opt  : word;
                   wcod : byte;
                   rcod : byte;
                   pba  : integer)
                  : integer; forward;
```

g. **LASSIGN** - Assign

Code = $00 Command = $40

The LASSIGN library routine provides access to the FHS directive ASSIGN. The Assign function establishes a logical connection between a file or device and the task through a specified logical unit under a given access permission.

Call Line:

```
RVL = LASSIGN(LUN,OPT,FDES,WCOD,RCOD,RECL,SIZ,SSSA,SSEA,SSN,FTYP,PBA)
```

Where:
- RVL = Returned Value
- LUN = Logical Unit
- OPT = Options
- FDES = File Descriptor Address
- WCOD = Write Code
- RCOD = Read Code
- RECL = Record Length
- SIZ = Size/Pointer
- SSSA = Shared Segment Starting Address
- SSEA = Shared Segment Ending Address
- SSN = Shared Segment Name
- FTYP = File Type and User Attributes
- PBA = Parameter Block Address

Declaration:

```
function  lassign(lun : byte;
                  opt  : word;
              var fdes : file_dscrpt;      (shown below)
                  wcod : byte;
                  rcod : byte;
              var recl : word;
              var siz  : siz_rec;          (shown below)
                  sssa : integer;
              var ssea : integer;
              var ssn  : integer;
              var ftyp : word;
                  pba  : integer)
                 : integer; forward;
```

File descriptor record

```
file_dscrpt  =
  record
    vol_id      : array[1..4] of char;
    user_nmbr   : word;
    catalog     : array[1..8] of char;
    file_name   : array[1..8] of char;
    extension   : array[1..2] of char
  end;
```

Size record

```
siz_rec   =
  record
    rsrvd         : byte;
    key_siz       : byte;
    fab_siz       : byte;
    data_blk_siz  : byte;
  end;
```

h. **LALLOC** - Allocate

Code = $00 Command = $80

The LALLOC library routine provides access to the FHS directive ALLOC. The allocate function reserves space on a direct-access device and in the directory-specified file type.

Call Line: RVL = LALLOC(OPT,FDES,WCOD,RCOD,RECL,SIZ,PBA)

Where: RVL = Returned Value
OPT = Options
FDES = File Descriptor Address
WCOD = Write Code
RCOD = Read Code
RECL = Record Length
SIZ = Size/Pointer
PBA = Parameter Block Address

Declaration:

```
function   lalloc(
                 opt  : word;
             var fdes : file_dscrpt;          (shown below)
                 wcod : byte;
                 rcod : byte;
                 recl : word;
                 siz  : siz_rec;              (shown below)
                 pba  : integer)
                : integer; forward;
```

File descriptor record

```
file_dscrpt  =
  record
    vol_id      : array[1..4] of char;
    user_nmbr   : word;
    catalog     : array[1..8] of char;
    file_name   : array[1..8] of char;
    extension   : array[1..2] of char
  end;
```

Size record

```
siz_rec   =
  record
    rsrvd         : byte;
    key_siz       : byte;
    fab_siz       : byte;
    data_blk_siz  : byte;
  end;
```

i. **LFCHVOL** - Fetch default volume

Code = $01 Command = $08

The LFCHVOL library routine provides access to the FHS directive FCHVOL. The Fetch-Default-Volume routine returns the requested default volume in the volume-ID field.

Call Line: RVL = LFCHVOL(FDES,PBA)

Where: RVL = Returned Value
FDES = File Descriptor Address
PBA = Parameter Block Address

Declaration:

```
function  lfchvol(var fdes : file_dscrpt;        (shown below)
                      pba  : integer)
                  : integer; forward;
```

File descriptor record

```
file_dscrpt  =
  record
    vol_id      : array[1..4] of char;
    user_nmbr   : word;
    catalog     : array[1..8] of char;
    file_name   : array[1..8] of char;
    extension   : array[1..2] of char
  end;
```

j. **LCHGLUN** - Change LUN assignment

Code = $01 Command = $10

The LCHGLUN library routine provides access to the FHS directive CHGLUN. The Change-Logical-Unit assignment allows the changing of a logical unit assignment from one task to another.

Call Line: RVL = LCHGLUN(LUA,OPT,LUB,TSK,SESS,PBA)

Where: RVL = Returned Value
LUA = Logical Unit for Calling Task
OPT = Options
LUB = Logical Unit for Called Task
TSK = Taskname
SESS = Session Number
PBA = Parameter Block Address

Declaration:

```
function  lchglun(lua  : byte;
                  opt  : word;
                  lub  : byte;
                  tsk  : array[1..4] of char;
                  sess : integer;
                  pba  : integer)
                : integer; forward;
```

k. **LFCHDEV** - Fetch device mnemonics

Code = $01 Command = $20

The LFCHDEV library routine provides access to the FHS directive FCHDEV. The Fetch-Device-Mnemonics function returns the device name, volume-id and status of all devices known to the system.

Call Line: RVL = LFCHDEV(BPTR,BLGN,PBA)

Where: RVL = Returned Value
BPTR = Buffer Pointer
BLGN = Buffer Length
PBA = Parameter Block Address

Declaration:

```
function  lfchdev(var bptr : array[1..10] of device_mnem;
                                                (shown below)
                  var blgn : device_size;       (shown below)
                      pba  : integer)
                  : integer; forward;
```

Device mnemonic record

```
device_mnem   =
  record
    device_name : integer;
    volume_id   : integer;
    reserved    : byte;
    status      : byte
  end;
```

Device size record

```
device_size   =
  record
    nmbr_entries : word;
    total_entries : word
  end;
```

1. **LFCHDIR** - Fetch directory entry

Code = $01 Command = $40

The LFCHDIR library routine provides access to the FHS directive FCHDIR. The Fetch-Directory function returns a directory entry (60 bytes) with each call. An end-of-directory status is indicated with the last directory.

Call Line: RVL = LFCHDIR(LUN,FDES,SIZ,PBA)

Where:
- RVL = Returned Value
- LUN = Logical Unit
- FDES = File Descriptor Address
- SIZ = Size/Pointer
- PBA = Parameter Block Address

Declaration:

```
function  lfchdir(lun : byte;
              var fdes : file_dscrpt;   (shown below)
              var siz  : siz_rec;       (shown below)
                  pba  : integer)
                : integer; forward;
```

File descriptor record

```
file_dscrpt  =
  record
    vol_id      : array[1..4] of char;
    user_nmbr   : word;
    catalog     : array[1..8] of char;
    file_name   : array[1..8] of char;
    extension   : array[1..2] of char
  end;
```

Size record

```
siz_rec   =
  record
    rsrvd         : byte;
    key_siz       : byte;
    fab_siz       : byte;
    data_blk_siz  : byte;
  end;
```

m. **LRTVATTR** - Retrieve attributes

Code = $01 Command = $80

The LRTVATTR library routine provides access to the FHS directive RTVATTR. The retrieve-attribute function gives the user access to physical attribute information pertaining to a particular device.

Call Line: RVL = LRTVATTR(LUN,FDES,DEVA,RECL,SIZ,FTYP,PBA)

Where: RVL = Returned Value
LUN = Logical Unit
FDES = File Descriptor Address
DEVA = Device Attributes
RECL = Record Length
SIZ = Size/Pointer
FTYP = File Type and User Attributes
PBA = Parameter Block Address

Declaration:

```
function  lrtvattr(lun : byte;
                   var fdes : file_dscrpt;    (shown below)
                   var deva : word;
                   var recl : word;
                   var siz  : siz_rec;        (shown below)
                   var ftyp : word;
                       pba  : integer)
                  : integer; forward;
```

File descriptor record

```
file_dscrpt  =
  record
    vol_id      : array[1..4] of char;
    user_nmbr   : word;
    catalog     : array[1..8] of char;
    file_name   : array[1..8] of char;
    extension   : array[1..2] of char
  end;
```

Size record

```
siz_rec   =
  record
    rsrvd          : byte;
    key_siz        : byte;
    fab_siz        : byte;
    data_blk_siz   : byte;
  end;
```

A.3.2 Loader Directive

a. **LLOADER**

Directive Number = $01

The LLOADER library routine provides access to the FHS directive LOADER.

Call Line: RVL = LLOADER(LPBA)

Where: RVL = Returned Value
LPBA = Loader Parameter Block Address

Declaration:

```
function  lloader(lpb_ptr : integer)
                          : integer; forward;
```

A.4 GENERIC TRAP ROUTINES

Generic TRAP routines have been provided for the TRAP #2 and TRAP #3 directives to allow for more efficient I/O processing. In these cases, the address of the parameter block is the only argument.

a. **LTRAP2** - TRAP #2 interface

This subroutine allows access to IOS via TRAP #2s.

Call Line: RVL = LTRAP2(PBA)

Where: RVL = Returned Value
PBA = Parameter Block Address*

Declaration:

```
function ltrap2(var bloc : ios_prm_blk)
                    : integer; forward;
```

b. **LTRAP3** - TRAP #3 interface

This subroutine allows access to FHS via TRAP #3s.

Call Line: RVL = LTRAP3(PBA)

Where: RVL = Returned Value
PBA = Parameter Block Address*

Declaration:

```
function ltrap3(var bloc : fhs_prm_blk)
                    : integer; forward;
```

THIS PAGE INTENTIONALLY LEFT BLANK.

APPENDIX B

RESIDENT RUN-TIME LIBRARY SOURCE FILE

B.1 EQUATE FILES

```
        INCLUDE   999S.RRTL.&.RRTLEQU.EQ     RRTL EQUATES

        IFNE      S$RMS
        IFEQ      LANGID-L$Pascal
          INCLUDE   9998.RR.FIOEQU.SA
        ENDC
        ENDC
```

B.2 MACRO FILE

```
        INCLUDE   999S.RRTL.&.RRTLMAC.MC     RRTL MACROS

        SECTION   8
RTLVEC  EQU     *
        DC.L    RRTLEND-RTLVEC          - LONG JSR ROUTINE VECTOR

        JUMP VECTOR TABLE

VECTBL  EQU       *
        IFNE      S$RMS
          DC.L      L$ATTSEG-*
            :             :
            :             :
          DC.L      L$CMR-*
        ENDC
        IFNE      S$IOS
          DC.L      L$READ-*
            :             :
            :             :
          DC.L      L$CNFDEF-*
        ENDC
        IFNE      S$FHS
          DC.L      L$CHKPT-*
            :             :
            :             :
          DC.L      L$LOADER-*
        ENDC
        IFNE      S$GEN
          DC.L      L$TRAP1-*   TRAP #1 SUBROUTINE
            :             :
            :             :
          DC.L      L$TRAP4-*   TRAP #4 SUBROUTINE
```

```
ENDC
IFNE        S$GP
  DC.L        L$LOC-*       LOCATION SUBROUTINE
  DC.L        L$DEF-*       PROVIDE DEFAULT DATA SUBROUTINE
ENDC
```

B.3 RUN-TIME LIBRARY SOURCE

```
IFNE        S$RMS
  INCLUDE     9998.RRTL.SRCRMS.SA          RMS Directives
ENDC
IFNE        S$IOS
  INCLUDE     9998.RRTL.SRCIOS.SA          IOS Directives
ENDC
IFNE        S$FHS
  INCLUDE     9998.RRTL.SRCFHS.SA          FHS Directives
ENDC
IFNE        S$GEN
  INCLUDE     9998.RRTL.SRCGEN.SA          Generic Routines
ENDC
IFNE        S$GP
  INCLUDE     9998.RRTL.SRCGP.SA           General Purpose Routines
ENDC
```

B.4 COMMON SUBROUTINES

```
IFNE        S$IOS
  INCLUDE     9998.RRTL.SUBSIOS.AI         IOS Directives
ENDC
IFNE        S$FHS
  INCLUDE     9998.RRTL.SUBSFHS.AI         FHS Directives
ENDC
IFNE        S$RMS
  INCLUDE     9998.RRTL.SUBSRMS.AI         RMS Directives
ENDC
END
```

APPENDIX C

CHAIN FILES

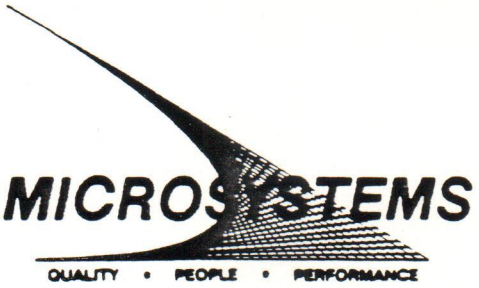
C.1 RTL.CF - Chain file to assemble the run-time library routines

```
=ASM 9998.RRTL.RRTLPASC/9998.RRTL.RRTLSRC,PRTL,PRTL;RMDZ=100
```

C.2 ACS.CF - Chain file to assemble the access routines

```
=ASM 9998.RRTL.RRTLACCS,RRTLACCS,RRTLACCS;RMDZ=100
=END
```

SUGGESTION/PROBLEM REPORT



Motorola welcomes your comments on its products and publications. Please use this form.

To: Motorola Inc.
Microsystems
2900 S. Diablo Way
Tempe, Arizona 85282
Attention: Publications Manager
Maildrop DW164

Product: _____ Manual: _____

COMMENTS: _____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Please Print

Name _____ Title _____

Company _____ Division _____

Street _____ Mail Drop _____ Phone _____

City _____ State _____ Zip _____

For Additional Motorola Publications
Literature Distribution Center
616 West 24th Street
Tempe, AZ 85282
(602) 994-6561

Four Phase/Motorola Customer Support, Tempe Operations
(800) 528-1908
(602) 438-3100

MOTOROLA

MOTOROLA GMBH GESCHÄFTSBEREICH HALBLEITER

POSTFACH 1229 · MÜNCHNER STRASSE 18 · D-8043 UNTERFÖHRING