Data General

Advanced Operating System (AOS) Shared Library Builder User's Manual

093-000191-02

Advanced Operating System (AOS) Shared Library Builder User's Manual

093-000191-02

For the latest enhancements, cautions, documentation changes, and other information on this product, please see the Release Notice (085-series) supplied with the software.

rdering No. 093-000191

Revision 02, June 1978

DataGeneral
SOFTWARE DOCUMENTATION

Advanced Operating System (AOS) Shared Library Builder User's Manual

093-000191

Revision History:

Original Release	- April 1976
First Revision	- April 1977
Second Revision	- June 1978

A vertical bar or an asterisk in the margin of a page indicates substantive change or deletion, respectively, from revision 01.

Contents

Chapter 1
Introduction to the Shared Library Builder

Terms and Conventions

The *Shared Library Builder* (SLB) accepts shared routine program files and forms shared libraries from them. The unqualified term "library" in this manual refers to shared libraries only.

Shared libraries group routines so that several different programs, running in different processes, can use them. Because the system needs only one copy of a shared routine in physical memory for several processes to share it, memory space is saved.

A number from 0 through 63 is assigned to each library in the system; library numbers 0 and 1 are reserved for system shared libraries. Two or more libraries in the same directory can have the same number as long as each program uses only one of them. The system differentiates between libraries with the same number by their unique names; a program file using a library name stores it in that program's symbol file.

Each shared library consists of a series of shared routines preceded by a single *shared library block header,* which the SLB constructs when it builds a shared library. In most cases you will never need to be aware of the header or its structure. Header structure is described in Appendix A.

Unless noted otherwise, all numbers used in this publication are decimal except core addresses, which are octal values.

Shared Routines

Shared routines are always written to be reentrant, and usually they are position-independent. A shared routine can be written which is position-dependent only if it will be bound into a specific area in the program file (Binder switch /B) and if care is taken to ensure that all absolute addresses are maintained consistently. Designing shared routines to be position-independent, and allowing the Binder to relocate them wherever required during the execution of the program, does away with these restrictions.

Discussions of shared routines in the remainder of this manual presume position-independence. Binder function switch /M specifies the number of 1K blocks which will be reserved as a single shared area for use by all shared routines.

Shared routines are loaded using one of three generalized procedure calls: ?RCALL, ?KCALL, and ?RCHAIN. Since generalized procedures can be written independently of the medium where they will ultimately reside, the Binder may change certain procedure calls into EJSR instructions. This would occur, for example, when a procedure call is made to a shared routine bound into the permanent root context. For more information on general procedure management and shared routines, consult Chapter 3 of the *AOS Programmer's Manual.*

In summary, the sequence of operations you must follow to build and use a shared library is as follows:

1) Design and prepare (using SPEED) one or more shared routines; these must be reentrant, usually are dynamically relocatable, must be shared code and will occupy one or more 1024-word pages. Define entry points in these routines with the .PENT pseudo-op (if you are using the Macroassembler).

2) Assemble the routines (using MASM). *

3) Bind these routines into one or more shared routine program files (using the Binder with the /S function switch).

4) Use the SLB to build a shared library containing one or more of these routines.

Chapter 2 describes SLB operation.

Having built one or more libraries, you then

1) Write a program which calls these routines (issuing ?RCALL, ?KCALL, or ?RCHAIN). The Binder may change ?KCALL and ?RCALL calls into EJSR instructions, speeding up access to the shared routine.

* 2) Assemble the program using MASM.

3) Bind the program with the shared library. Use Binder argument switch /M to reserve shared code area, /B to bind some or all shared routines in a library into the root (transforming ?RCALLs and ?KCALLs into EJSRs), and /X to exclude certain shared routines from being bound into the root.

4) Execute your program.

Note that you can build a library having one or more routines that call other routines residing in as yet unbuilt libraries.

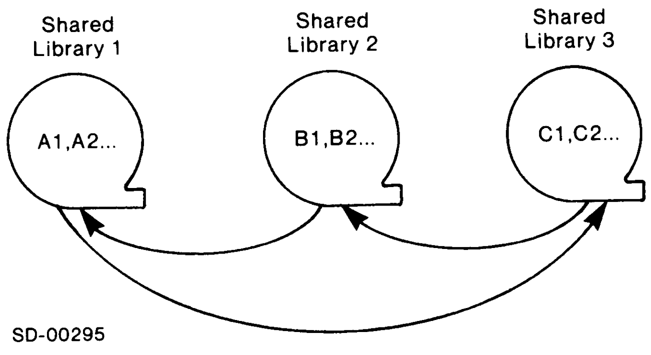

Figure 1-1. Inter-Library References

End of Chapter

Chapter 2
How to Operate the Shared Library Builder

Operating Procedures

Use the CLI to issue SLB commands. After each CLI command to the Shared Library Builder, SLB will search for argument program files bearing the extension ".PR" (even if you omitted this extension in routine names). Likewise, SLB will always produce a shared library with the extension ".SL".

The format of the SLB command is:

```
XEQ SLB/O=output-library-name library-number
          input...
```

where: *input* is the name of one or more shared routine program files with the extension ".PR".

library number is a decimal integer from 0 through 63 that you want to assign to the new shared library. (Library numbers 0 and 1 are reserved for two system libraries.)

output library name is the name that will be given to the new shared library. SLB will append the extension ".SL" to this name.

You must supply an output library name as part of the /O keyword function switch, since there is no default output filename.

You can specify an output listing file or device to receive input names and other SLB information. To name a listing file or device, use either a simple or keyword function switch:

/L

or

/L=filename

/L selects the already-specified current listing file. /L=filename specifies the listing file *filename* in place of the current listing file.

Error Messages

SLB may output certain error messages during its execution. These messages always go to the current CLI output file. A list of SLB error messages and their meanings follows:

ILLEGAL LIBRARY NUMBER - The library number you specified in the SLB command line is outside the range 0-63 inclusive.

INSUFFICIENT MEMORY - There is not enough memory available for the SLB to run.

NO CURRENT LIST FILE - You used the current listing file switch in the SLB command line ("SLB/L"), but there is no current list file specified to the CLI at this time.

NO OUTPUT FILE SPECIFIED - You did not specify the name of the output library in the SLB command line.

Example of SLB Operation

In the following example you wish to create a shared library named "TEST.SL" that contains three shared routines entitled "T1", "T2", and "T3". There are three steps you must follow:

1) Assemble T1, T2, and T3;
2) Bind T1, T2, and T3;
3) Execute SLB.

To assemble the three shared routines, give the three MASM commands:

```
XEQ MASM/L=@LPT T1)
XEQ MASM/L=@LPT T2)
XEQ MASM/L=@LPT T3)
```

MASM creates the three output listings shown in Figure 2-1.

```
 0001 T1 AOS MASM REV  00.00              14:37:47 12/11/77
                       .TITLE  T1
02      000001         .NREL   1
03                     .PENT   T1A.T1B
04                     .EXTN   T3A
05
06 000001000345 T1A:    345
07 000011000000$        T3A
08 000021000123 T1B:    123
09                     .END
**00000 TOTAL ERRORS, 00000 PASS 1 ERRORS


 0001 T2 AOS MASM REV 00.00               14:38:17 12/11/77
                       .TITLE T2
02      000001         .NREL   1
03                     .PENT   T2A T2B
04
05 000001012345 T2A:    12345
06 000011002001         .BLK    2001
07 020021000654 T2B:    654
08                     .END
**00000 TOTAL ERRORS, 00000 PASS 1 ERRORS


 0001 T3 AOS MASM REV 00.00               14:38:46 12/11/77
02      000001         .TITLE T3
03                     .NREL 1
04                     .PENT   T3A

06 000001000234 T3A:    234
07                     .END

**00000 TOTAL ERRORS, 00000 PASS 1 ERRORS
```

Figure 2-1. Assembly of Shared Routines

The shared routines in this example do not perform any operations, but are merely presented to illustrate the procedure for forming a shared library. Notice that each shared routine has one or more entry points defined by a .PENT pseudo-op, and each routine is written for the shared code partition (.NREL 1).

Shared routine T1 makes an external reference to "T3A" in shared routine T3.

Next you bind these modules with the following three commands:

```
XEQ BIND/S/L=@LPT T1)
XEQ BIND/S/L=@LPT T2)
XEQ BIND/S/L=@LPT T3)
```

Figure 2-2 shows the Binder output listings.

```
T1.PR CREATED BY AOS BINDER ON 12/11/77 AT 14:39:21
T1

000000 WORDS OF ABSOLUTE DATA
ZMAX: 000050
NMAX: 000473
START  OF SHARED: 07600
LENGTH OF SHARED: 002000

PARTITION    TYPE    START   END    #OF OVERLAY AREAS
000007     SHR CD   076000 076002  000000

                ?CSZE    000000
                ?ZMAX    000050
                ?NMAX    000473
P               T1A      076000
U               T3A      076001
P               T1B      076002
                ?CLOC    177777

T2.PR CREATED BY AOS BINDER ON 12/11/77 AT 14:39:56
T2

000000 WORDS OF ABSOLUTE DATA
ZMAX: 000050
NMAX: 000473
START OF SHARED: 074000
LENGTH OF SHARED: 004000

PARTITION    TYPE    START    END    #OF OVERLAY AREAS
000007     SHR CD   074000 076002   000000

                ?CSZE    000000
                ?ZMAX    000050
                ?NMAX    000473
P               T2A      074000
P               T2B      075002
                ?CLOC    177777

T3.PR CREATED BY AOS BINDER ON 12/11/77 AT 14:40:31
T3

000000 WORDS OF ABSOLUTE DATA
ZMAX: 000050
NMAX: 000473
START OF SHARED: 076000
LENGTH OF SHARED: 002000

PARTITION    TYPE    START    END    #OF OVERLAY AREAS
000007     SHR CD   076000 076000   000000

                ?CSZE    000000
                ?ZMAX    000050
                ?NMAX    000473
P               T3A      076000
                ?CLOC    177777
```

Figure 2-2. Binding of Shared Routines

In the output listing for T1.PR, "T1" is listed as being the title of T1.PR. Zero words are defined for the absolute data partition. ZMAX stands at 50, so there is no unshared code in ZREL memory. NMAX stands at 473 since several databases required by the system have been built starting at 400; these databases include the user status table and a task control block.

Shared code always resides at the upper end of the 32K word context, and the AOS system always allocates memory for this code in multiples of 1024-words. Thus the Binder listing shows the start of the shared code partition to be octal 76000 and the length of this partition to be octal 2000 words. The shared code partition is type "7", and no overlay areas are listed, since shared routines do not use user overlays.

?CSZE and ?CLOC indicate the size and starting location respectively of unlabeled common. There is none, so none is indicated. The ?ZMAX and ?NMAX symbols show the end of ZREL and the unshared code partition, respectively. T1A and T1B are flagged with a "P" to indicate that they are .PENT symbols; T3A, an external reference, was unresolved at the time of the bind; you can ignore this, since that reference will be supplied when you build the shared library. Notice that BIND does not signal an error.

Listings output during the bind of T2 and T3 are similar to the listing of T1.

Figure 2-3 shows the Shared Library Builder listing produced by the following command:

```
XEQ SLB/L=@LPT/O=TEST 53 T1 T2 T3)
```

The name of the shared library is "TEST.SL", and its number is 53 decimal.

The listing gives the titles of each of the shared routines, in the order that they were named in the SLB command line.

```
TEST.SL CREATED BY AOS SHARED LIBRARY BUILDER ON 12/11/77 AT 14:41:1
LIBRARY NUMBER = 53

T1

T2

T3
```

Figure 2-3. Building the Shared Library

End of Chapter

Appendix A
Internal Shared Library Structure

Most users will never be aware of internal shared library structure. You need to understand this structure only if:

1) you plan to build shared libraries without listing the SLB, or
2) you intend to read the headers (as the Binder does).

Otherwise, you may disregard this appendix.

When you bind one or more object modules into a shared routine (using the Binder /S switch), you build a program file whose code (the shared routine proper) is placed in 1024-word shared pages, starting at the top of the 32K context:

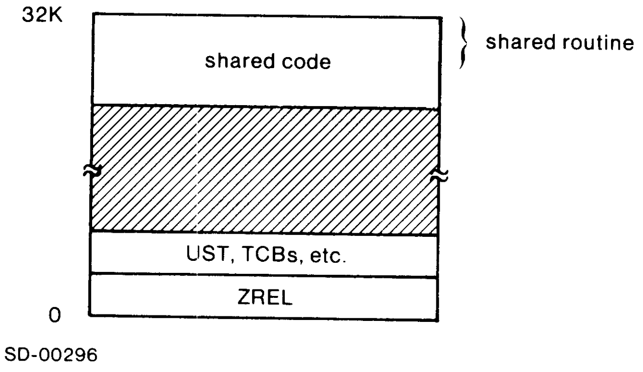

Figure A-1. Shared Routine Placement

When the Shared Library Builder constructs a shared library from one or more of these program files, it builds a shared header, and follows this with only the shared code portions of the shared routine program files.

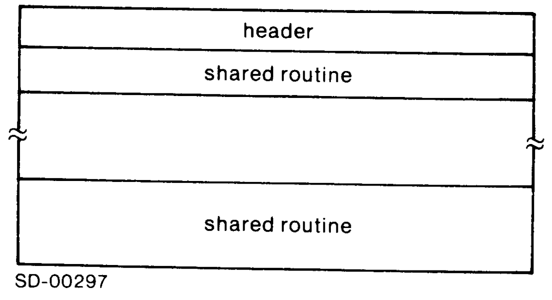

Figure A-2. Shared Library Overview

Shared Library Headers

The first five words of the shared library header have fixed contents; the rest of the header varies in length, and describes items such as the different .PENT symbols in all the shared routines. Header structure is diagrammed in Figure A-3.

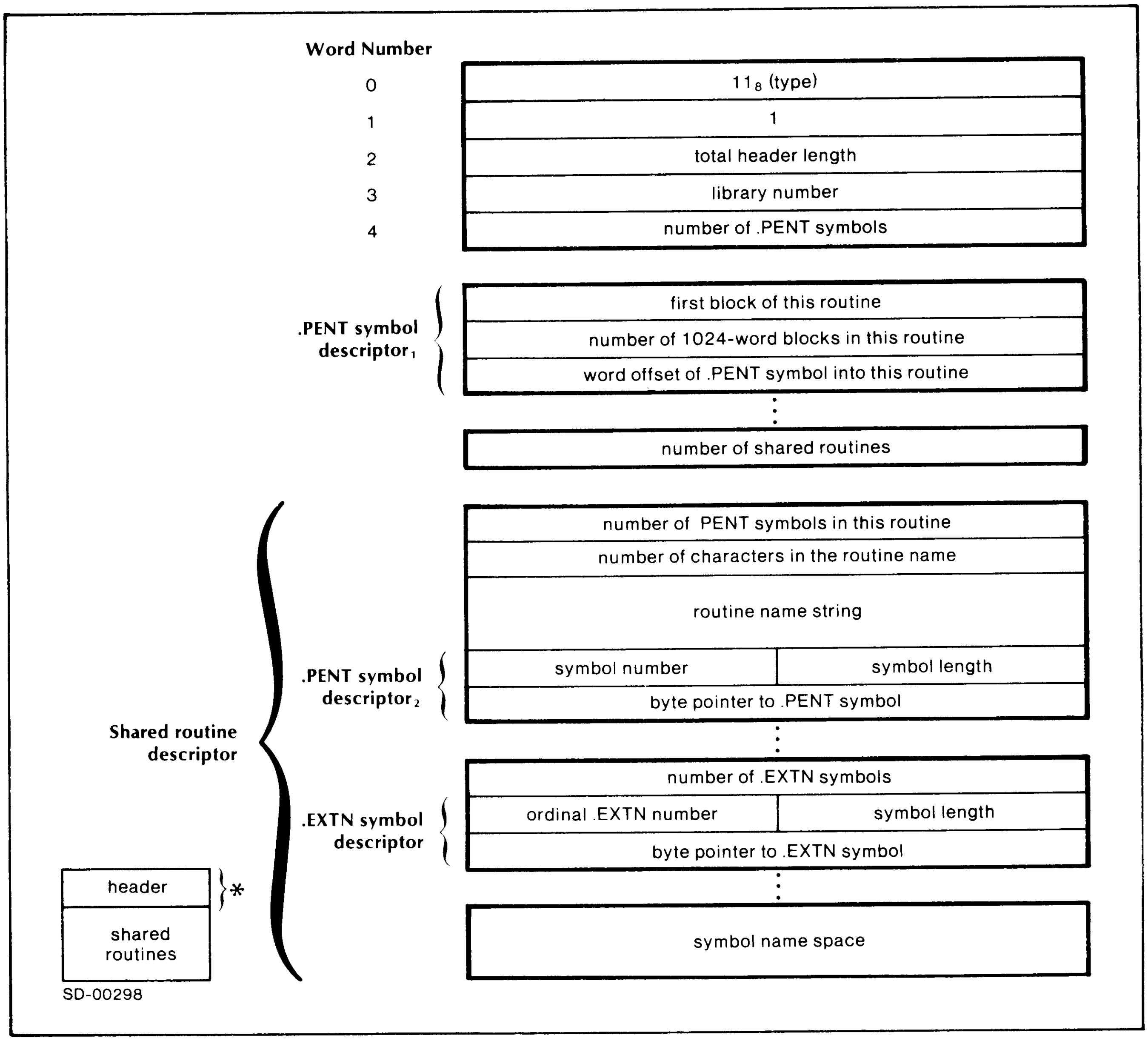

*Figure A-3. Shared Library Header**

As Figure A-3 shows, words 0 and 1 contain two unchanging constants. Word 2 lists the total length of the header in multiples of 1024-words. Word 3 lists the number (0-63) assigned to this library, and word 4 lists the total number of symbols described by the .PENT pseudo-op in this shared library.

Every .PENT symbol has a pair of .PENT descriptors (1 and 2); each descriptor$_1$ is three words long, and each descriptor$_2$ is two words long. All of the descriptor$_1$ follow immediately word number 4 in the same order that the .PENTs occur in the shared routines.

Each descriptor$_1$ has the following structure. Word 0 lists the start of the shared routine containing the .PENT associated with this descriptor. Since each shared routine's size is a multiple of 1024 words, this starting address is in units of 1024 words, commencing with the first routine in the library. Thus the first routine (following immediately the end of the header) has a starting address of 0, the second has an address of 1 or more, etc. The second word in each descriptor$_1$ lists the number of 1024-word sections in the routine, and the third word describes the word offset (commencing with 0) of the .PENT symbol in the routine.

Following the last descriptor$_1$ is a word that lists the total number of shared routines. Following this word is a series of shared routine descriptors, one for each shared routine in the library.

Each shared routine descriptor has three major sections. The first lists the number of .PENTs in the routine, the number of characters in the name of the routine, and the routine name string. The second section contains as many .PENT descriptors$_2$ as there are .PENTs in this routine.

The first word of each descriptor$_2$ consists of two bytes: the left byte contains the relative .PENT symbol number in the entire library, and the right byte contains the size in characters of the symbol. The second word of each descriptor$_2$ contains a byte pointer (relative to the start of the header) to the .PENT symbol name in name space.

Following the last descriptor$_2$ is the final section of the shared routine descriptor. The first word of this section counts the number of external symbols in this routine. Folowing this count are a series of two-word .EXTN descriptors, one for each external in the routine. Each descriptor lists the external symbol's unique ordinal number within the library and its length, and points to the symbol name in name space.

End of Appendix

Index

Within this index, the legend "f" following a page number means "and the following page"; "ff" means "and the following pages".

DataGeneral

Software Documentation Remarks Form

How Do You Like This Manual?

Title ______________________ No. __________

We wrote the book for you, and naturally we had to make certain assumptions about who you are and how you would use it. Your comments will help us correct our assumptions and improve our manuals. Please take a few minutes to respond.

If you have any comments on the software itself, please contact your Data General representative. If you wish to order manuals, consult the Publications Catalog (012-330).

Who Are You?

☐ EDP Manager
☐ Senior System Analyst
☐ Analyst/Programmer
☐ Operator
☐ Other ______________

What programming language(s) do you use? ______________

How Do You Use This Manual?

(List in order: 1 = Primary use)

_____ Introduction to the product
_____ Reference
_____ Tutorial Text
_____ Operating Guide
_____ ______________

Do You Like The Manual?

Yes	Somewhat	No	
☐	☐	☐	Is the manual easy to read?
☐	☐	☐	Is it easy to understand?
☐	☐	☐	Is the topic order easy to follow?
☐	☐	☐	Is the technical information accurate?
☐	☐	☐	Can you easily find what you want?
☐	☐	☐	Do the illustrations help you?
☐	☐	☐	Does the manual tell you everything you need to know?

Comments?

(Please note page number and paragraph where applicable.)

From:

Name ______________ Title ______________ Company ______________

Address ______________________________ Date ______________

FOLD DOWN FIRST FOLD DOWN

FIRST
CLASS
PERMIT
No. 26
Southboro
Mass. 01772

BUSINESS REPLY MAIL

No Postage Necessary if Mailed in the United States

Postage will be paid by:

Data General Corporation

Southboro, Massachusetts 01772

ATTENTION: Software Documentation

FOLD UP SECOND FOLD UP

SD-00742A

STAPLE