

TE: May 11, 1978 .

PE-T-445

: ALL R & D Personnel .

JM: C. Hannauer

SUBJECT: USER CREATION OF A SHARED LIBRARY (Rev 15)

The mechanism for sharing libraries described in this document may be temporary in nature, and may not be supported by Prime beyond Rev. 15. Users are hereby advised not to use these tools to build their own libraries unless they are willing to support the implied mechanisms themselves, including changes to PRIMOS and SEG.

This document describes the method presently available for creating a shared library. Three subjects are dealt with:

- General Discussion of Shared Libraries
- PRIMOS IV Support for Shared Libraries
- Preparing a Shared Library
- Using SEG to Build a Shared Library

GENERAL DISCUSSION OF SHARED LIBRARIES

Rev. 15 mechanism for sharing libraries permits creation of several independent libraries or parts of libraries. It is anticipated, for example, that the Fortran Library will consist of several parts. For ease of reference each is referred to as a "package". There may be more than one package per segment and more than one segment per package. Each package is considered a separate shared library by the operating system. When a package is installed it must be assigned a number by PRIMOS, or, alternately must tell PRIMOS which package is being installed. Normally it is preferable to let PRIMOS assign the package number. Under some circumstances it may be reasonable for the package to declare its own number.

Library sharing is handled by making use of the Direct Entry Call mechanism introduced at Rev. 14. A Direct Entry Call is managed by satisfying an external reference with a fault pointer at load time. (A fault pointer has bit 1 of the high order word set).

When one of these pointers is encountered at run time the hardware makes a fault and the operating system examines its collection of direct entry routines. At Rev. 14, if the routine was not found, PRIMOS IV aborted with a POINTER FAULT. For Rev. 15, additional code was added to PRIMOS IV. If there are any shared libraries the fault will be passed on to each library in turn until it is satisfied or found to be missing.

Shared libraries reside in shared segments and each must have associated with it a "Ring 3 Fault Handler" which processes any faults associated with it by the operating system. The ring 3 fault handler examines the collection of direct entry calls known to it and if the fault is not satisfied by any of these the fault handler returns to PRIMOS IV.

The ring 3 fault handler is loaded in ring 3 with the library and performs two functions when a faulting call can be satisfied by its library. One of these is always performed and consists of replacing the fault pointer with a pointer to the actual location in the library which satisfies the fault. The faulting instruction is then restarted and execution of the user's program proceeds normally.

The other function is initialization. It is performed the first time a library is invoked in the course of running a SEG runfile. For example, the first time a call is made to a shared Fortran Library routine the initialization is performed. Subsequent calls to any other shared Fortran Library routines will not cause initialization to be performed. However, should a new SEG runfile be invoked, or the same one restarted, the whole process begins again.

Initialization is required as most libraries have impure initialized areas which have to be set up in a private user segment. Examples of this kind of area are impure link frames and initialized common. Initialization consists of moving a template copy of the initialized

determined private user segment. For Rev. 15 segment 6001 has been available for this purpose to avoid conflicts with normal SEG ring. However, segment 6001 is a limited resource and must be used by all shared libraries. Libraries which have extensive requirements for impure initialized or uninitialized areas will have to use of segments in the 4000 range. See the discussion below for details of the initialization process.

Final component required for shared libraries is a mechanism for installing the shared library known to PRIMOS IV. When PRIMOS IV is cold started, no shared libraries are present. Installing a shared library must be done each time PRIMOS IV is brought up and consists of loading the shared library into its shared segment(s) and informing PRIMOS of its existence. PRIMOS then adds the library to its table of shared libraries. This is usually accomplished by running a program which is loaded with the ring 3 fault handler and the library in one SEG load sector. For details, see below.

SER VISIBLE PRIMOS IV SUPPORT FOR SHARED LIBRARIES

There are two new PRIMOS routines to support shared libraries. DYNT's these routines are included in DIRECV>MAIN.

NEW (<package no.> , <calf> , <code>)

This routine is called by the library installing program to tell PRIMOS of a new library. For an example see DIRECV>MAIN.

<package no.> is the number of this package. If omitted, the next free number is assigned.

<calf> is the label of the CALF instruction which begins execution of the ring 3 fault handler.

<code> is returned as zero if the call succeeds, ESROOM if no package number was specified and there the maximum number of packages is already installed, or ESBPARG if the call has bad parameters.

SNXT This routine tells PRIMOS that this fault handler could not satisfy the fault. After giving the call the fault handler should restore the user program's registers and PRTN. The fault will re-occur but this time PRIMOS will pass control to the library with package number <package no.>+1 if there is one, or give an error if there is not. For an example see DIRECV>R3POFH. This routine must be called in PMA as follows:

```
CALL LIBNXT
AP <package no.>,S
AP SB%,SL
```

<package no.> is the package number of the calling routine.

3 PREPARING A SHARED LIBRARY

A shared library consists of four separate parts. These are:

A replacement library library object module which contains "DYNT's" for the routines to be shared in place of the actual code.

A ring 3 fault handler linked to the library

An installation program linked to the library

The shared library itself

3.1 USE AND CREATION OF THE DYNT LIBRARY

A DYNT is generated by the PMA DYNT pseudo-op. When DYNT is encountered by SEG at load time, a special linkage block is created which contains the name of the routine. All references to that routine are then satisfied by a fault pointer to this block.

A DYNT is created by writing a PMA routine of the following form for each routine to be shared:

```

SEG
DYNT F00           /*for direct entry call "F00"
END

```

Each DYNT should be a separate module (have its own END statement). The library file containing the DYNTS should - as usual - begin with an RFL and end with an SFL. These are added using EDB.

3.2 THE RING 3 FAULT HANDLER

Each "package" must have its own ring 3 fault handler. To make it easier to create shared libraries a standard ring 3 fault handler is provided, however, a description of the functionality required is included below for those needing to write their own ring 3 fault handler.

The source of the standard ring 3 fault handler is called R3POFH and it is to be found in UFD DIRECTV on the Master Disk. R3POFH is written in PMA. A customized copy of R3POFH is required for each package. The customizing is accomplished by assembling it with a hash table contained in a \$INSERT file named HTAB (see below).

R3POFH will move one initialized area to the specified locations in a private user segment. It attempts to locate the required external reference through a hash table search and consequently is most efficient for libraries with multiple entries. It will work, however, for libraries with only one entry.

R3POFH is coordinated with the standard MAIN ----

provided in SEG for moving data in SEG run files. In particular R3POFH supplies as an external name a 5 word block, SEGSBK, into which SEG's Loader will record the details of the template move. (See the description of loading a shared library which follows in Section 4.)

The hash table, HTAB, is most easily created by running the utility program #HASHR in UFD DIRECTV. The input to #HASHR is a simple list of the shared routines contained in the library. Not all routines loaded in the library need be included. Only those routines to which the programmer wishes users to have access are needed. They should be entered one per line in the input file. The resulting hash table contains 5 words per entry, 3 words for the entry point name and 2 for the IP which will be filled in at Load time. #HASHR is mostly self describing. The requested hash modulus is the number of entries in the table. Obviously it must be at least as great as the number of routines to be hashed. When the hash table has been created #HASHR reports the average depth of search to access an entry. #HASHR will cycle until the user CTL-P's out of it, thus it is possible to try several table sizes until an optimum search depth is reported.

The user needing to write his own ring 3 fault handler is referred to R3POFH for an example. The ring 3 fault handler must begin with a CALF instruction, followed by an RSV. It then locates its DYN through information stored on the stack and uses the text string to locate the routine in its own tables. If the routine is found the ring three fault handler replaces the fault pointer with the correct pointer, restores the registers (RRST) and restarts the faulting instruction. If it is not found the fault handler calls LIBNXT with its package number and then restarts the instruction which will then fault again (but to a new package).

3.3 THE INSTALLATION PROGRAM

Each "package" must be installed by a separate call to the new PRIMOS IV routine LIBNEW (see above). Several packages can in theory be installed at once, however, a standard routine is provided which will install one and which is coordinated with R3POFH. The source of this routine is located in UFD DIRECTV as MAIN. MAIN does not need to be customized for each package. Assuming that MAIN has been loaded as described in section 4, it will either cause PRIMOS IV to assign a package number or will pass on to PRIMOS IV as follows:

```
R LI4000          /*have PRIMOS IV assign the package number.
R LI4000 1/6      /*tell PRIMOS IV to assign number 6
```

In addition to the installation routine, MAIN contains the two DYN's, LIBNEW and LIBNXT, required for communication with PRIMOS.

of the functionality of MAIN follows.

The installation routine must perform at least two operations for the library. The first of these is to snap the link to LIBNXT for the ring 3 fault handler. The ring 3 fault handler is originally installed by the SHARE command with write access so that this address may be stored in the shared library segment containing the fault handler. The install program must set up the IP so that the library may later be shared with read/execute access only.

Secondly, the installation routine must call LIBNEW to install the library either passing the package number to be used or accepting one returned by LIBNEW. If PRIMOS assigns the package number, it must also be passed to the ring 3 fault handler.

3.4 SPECIAL PREPARATION OF LIBRARY ROUTINES FOR SHARING

No special preparation of library routines is required for loading as a shared library. As a first attempt at creating a shared library it is reasonable to simply load the existing library as described in the Section 4 and proceed. For final installation as a shared library in use system wide there are some efficiency considerations which may make it appropriate to massage the library somewhat prior to loading it as a shared library. A general discussion of loading a shared library will be helpful at this time for understanding the discussion of library optimization which follows.

The ring 3 fault handler and all pure procedure are loaded into in shared segments. All COMMON blocks, impure procedure and impure link frames must be loaded into private user segments. Pure link frames may be loaded either in private or shared segments. When the library load is complete, SEG is instructed to move a copy of the impure area into one of the shared segments from which it will serve as the template for the ring 3 fault handler initialization routine.

Shared libraries are supposed to benefit users by reducing SEG restore times and by a system wide reduction of memory utilization through the use of the shared memory image of the library routines. There are at least two ways in which the goal of benefiting the user can be subverted. First paging activity can be increased as a result of the dispersal of the user's run image over more pages and segments than was the case with the unshared library. Secondly performance can be degraded by excessive package initialization times.

It is probably not feasible to completely eliminate the need for package initialization. However, an attempt can be made to minimize it. In some cases the size of link frames can be reduced. In particular FORTRAN routines can be recompiled with the DYNAM option to put most local variables on the stack. This must be done carefully so that variables which are

also be modified to put most local variables on the stack. In this process many link frames will become pure. These can be loaded into the shared procedure segment(s) which removes them from the initialization process entirely.

Normal SEG loading usually puts COMMON blocks in amongst the link frames. At load time uninitialized COMMON blocks may be defined with the A/SY command prior to loading the library object files. In this way uninitialized COMMON can be gotten out of the way of initialized COMMON and impure link frames. Alternatively the Loader's CO command may be used to separate COMMON from link frames.

When the template impure area is declared to SEG's Loader only the initialized area needs to be defined. In this way if uninitialized areas have been separated from initialized areas, the overall length of the initialization can be minimized.

The second step which may be taken is to organize the procedure load so that routines which call each other or which are likely to be called together are loaded contiguously. It may also be advantageous to load some routines on page boundaries using the Loader's "P/LO" command. Routines with pure link frames can also be loaded under the MI option which causes the link frame to be loaded right after the procedure frame in the procedure segment. This will help reduce paging activity by reducing the number of pages which must be active at any one time.

A word on finding pure link frames may be useful here. First, a link frame is pure only if at no time during the execution of a program will any legitimate attempt be made to store a new value in any of its locations. At the present time link frames may customarily contain ECB's, pointers to external names and COMMON blocks, local variables and constants. Local variables (including arrays) and pointers to external names are likely to make the link frame impure. We have discussed the techniques for reducing or eliminating local variables from the link frame.

Pointers to external names may be impure because they may be fault pointers to direct entry calls either to PRIMOS IV or to other libraries. There are two ways of determining whether such pointers are present in any given link frame. The first is to examine a listing concordance for the presence of such names. The second is to individually load each routine with SEG's loader to determine if such routines are called by it.

LOADING A SHARED LIBRARY USING SEG

As has been described above, a shared library is loaded together with a ring 3 fault handler and an installation routine. The installation routine is best loaded into segment 4000 so that it occupies only one segment and can be started up as an R-mode program. The ring 3 fault

frames to be loaded into the shared segments, they should be loaded under the MI option of SEG's Loader so that they will be contiguous with their procedure frames. In the annotated example which follows it is assumed that the user is familiar with SEG's Loader and has read the documentation for SEG Rev. 15. Further, it is assumed that the user plans to optimize the load of the library and that there are some routines with pure link frames which have been separated from the remainder of the routines so that their link frames may be loaded in the procedure segment.

The first responsibility of user planning to share a library is to coordinate the utilization of segments below 4000 so that the new shared library will not occupy segments assigned for other purposes. Secondly, if part of segment 6001 is to be used, usage of this segment must also be coordinated with existing shared libraries. As of the writing of this document, segment 6001 is unused above 54000 (octal). In the example below it is assumed that this is still the case. Segment 2020 has been selected for the pure procedure only for purposes of the example below.

The following is a sample command file:

```

SEG
LOAD #FINST
MI          /*allow mixing of procedure and data.
SP          /*Read RUNIT into segment 4000
           for running MAIN as an R-mode
           program, no argument needed.
A/SY DUMMY PROC 6001 53000 /*move up to free locs. in 6001
           treating 6001 as a procedure
           segment for loading under
           the MI option.
A/SY LOWS 6001 0          /*a symbol for the bottom of
           the initialized area.
A/SY COMMN1 PROC 6001 2000 /*declare uninitialized COMMON.
S/LO B_R3POFH 0 2020 2020 /*load the ring 3 fault handler,
           link and procedure in seg. 2020
S/LO SHR1 0 2020 6001    /*SHR1 does not have pure
           link frames.
P/LO SHR2 2020 6001     /*load SHR2 on a page boundry for
           efficiency at execution time.
D/LO SHR3
S/LO SHR4 0 2020 2020   /*load the first module with
           pure link frames.
D/LO SHR5              /*SHR5 also has pure link frames.
S/LO SHR6 0 2020 6001  /*back to impure link frames.
D/LO SHR7              /*the last of the user's library
D/LI SFTNLB           /*pure Fortran library (shared).
S/LI IFTNLB 0 6001 6001 /*impure library must be
           initialized along with impure
           data. This may not be necessary
           if a LOAD COMPLETE was obtained
           after loading SFTNLB.

```



```

D/PL          /*if no LOAD COMPLETE was obtained.
MA 2          /*check the load.
MV LOWS SEGBLK 2020 /*move the initialized area in
                segment 4000 to the top of the
                shared procedure segment (2020).
MA SHLIBMAP   /*get a map (its a good idea).
RE           /*SEG now saves runfile automatically.
SH          /*split out sharable segments
                and segment 4000.
LI          /*(response to 'TWO CHARACTER ID')
DELETE       /*done with SEG run file.
GU          /*return to PRIMOS command level.
CO TTY

```

At least two files will have been created when the SH command was given at SEG command level. The first of these will be named LI2020 - in the example. LI2021, etc. may also be created if the library is large. These files are the shared library and should be shared at PRIMOS IV startup time as part of the shared library installation. The second (or last) file will be named LI4000. This file contains the installation program - MAIN. Since it was loaded after the SP command was given to SEG's loader it is a self contained run file and can be run from PRIMOS IV command level to install the library. The following command stream is an example of shared library installation. These commands can only be given from the system console.

```

OPR 1          /*turn special privledges on
SHARE LI2020 2020 700 /*load library into segment 2020
                /*with write access
R LI4000       /*install the shared library
SHARE 2020    /*re-share 2020 with read/execute
                /*access only
OPR          /*turn special privledges off

```

If it is necessary to reinstall the library with out bringing the system down (this may be a risky proposition) and the user knows the package number and the user is using the standard supplied installation routine the package number (for example 6) may also be supplied when running LI4000 as follows:

```
R LI4000 1/6
```