# IAS Task Builder Reference Manual

Order Number: AA–2533E–TC

This manual introduces and describes the IAS Task Builder.

**Operating System and Version:** IAS Version 3.4

**May 1990**



The postpaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist in preparing future documentation.



This document was prepared using VAX DOCUMENT, Version 1.2

# Contents

EXAMPLES

## FIGURES

## TABLES

# Preface

## Purpose of This Manual

This manual introduces you to the concepts and capabilities of IAS task building. It expands on the summaries of the PDS LINK command and the MCR TKB command given in the *IAS PDS User's Guide* and the *IAS MCR User's Guide*, respectively.

Examples are used to introduce and describe features of the Task Builder. These examples proceed throughout the manual from the simplest case to the most complex. You might want to try out some sequences to test your understanding of the Task Builder.

You should be familiar with the PDP-11 computer, its peripheral devices, and the software supplied with the IAS system.

This manual is organized and written as a reference manual, assuming a system programmer level of expertise. Data processing terms and concepts familiar at such a level are therefore not defined.

## Structure of This Document

This manual has ten chapters.

- Chapter 1 outlines the capabilities of the Task Builder.
- Chapter 2 describes the command sequences used for building tasks under PDS.
- Chapter 3 describes the command sequences used for building tasks under MCR.
- Chapter 4 defines qualifiers and switches.
- Chapter 5 defines options.
- Chapter 6 describes memory allocation for the task and for the system and gives examples of the memory allocation file.
- Chapter 7 describes the overlay capability and the language used to define an overlay structure.
- Chapter 8 describes the two methods used for loading overlay segments.
- Chapter 9 introduces shareable global areas.
- Chapter 10 presents supervisor-mode libraries.
- The appendixes list error messages and give detailed descriptions of the structures used by the Task Builder.
- Appendix G is a glossary of terms.

## Associated Documents

The following manuals are prerequisite sources of information for readers of this manual:

- *IAS Executive Facilities Reference Manual*
- *IAS PDS User's Guide*
- *IAS MCR User's Guide*

Other documents related to the contents of this manual are described briefly in the *IAS Master Index and Documentation Directory*. The directory defines the intended readership of each document in the IAS documentation set and provides a brief summary of the contents of each manual.

# 1 INTRODUCTION

This chapter introduces the IAS Task Builder and describes the role of the Task Builder in the IAS operating system.

The fundamental executable unit in the IAS system is the task. A routine becomes an executable task image, according to the following sequence:

1 The routine is written in a source language supported by the IAS system.

2 The routine is entered as a source file, through an editor.

3 The routine is translated to an object module, using the appropriate language translator.

4 The object module is converted to a task image using the Task Builder.

5 Finally, the task is run.

If errors are found in the routine as a result of translating or executing the task, you edit the text file created in step 2 to correct the errors, then repeats steps 3 through 5.

If a single routine is to be executed, you provide the object module file name to be used as Task Builder input.

In typical applications, several routines are run rather than a single module. In this case the user names each of the object module files. The Task Builder then links the object modules, resolves any references to any shareable global areas, and produces a single task image that is ready for execution.

The Task Builder makes a set of assumptions (defaults) about the task image based on typical usage and storage requirements. These assumptions can be changed by including qualifiers or switches and options in the task-building command sequence.

The Task Builder can also produce a memory allocation file. This file gives information about how the task is mapped into memory. The user can examine the memory allocation file to identify what support routines and storage reservations are included in the task image. Further, the Task Builder can produce a symbol table file suitable for input to the Task Builder during the build of another task. For example, such a procedure is used in binding tasks to shareable global areas.

To reduce the amount of memory required by the task, the overlay capability can be used to divide the task into overlay segments.

Overlaying a task enables more code and/or data to be fitted into the available 32K of virtual address space. Overlays may be either disk resident, in which case they are reloaded from disk each time they are required, or memory resident. Memory resident overlays remain resident in memory once loaded and are mapped as required using the Memory Management directives (see the IAS System Directives Reference Manual). Disk resident overlays save physical as well as virtual memory.

If the task is configured as an overlay structure (that is, a multi-segment task), overlay segments are loaded using either the autoload or manual method.

The autoload method makes the loading of overlays transparent to the user. Loading of the overlay segments is accomplished automatically by the Overlay Runtime System according to the structure defined by the user at the time the task was built.

The manual load method requires that explicit calls to the Overlay Runtime System be included in the coding of the task, and gives the user full control over the loading process.

If the task communicates with another task or makes use of common subroutines to save memory, the Task Builder enables you to link to existing shareable global areas and to create new shareable global areas for future reference.

You can become familiar with the capabilities of the Task Builder by degrees. Chapter 2 and Chapter 3 give basic information about task building commands for PDS and MCR users. This information is sufficient to handle many applications. The remaining chapters deal with special features and capabilities for handling complex applications and tailoring the task image to suit the application. The appendixes include detailed information about the structure of the input and output files processed by the Task Builder, details of non-standard versions of the Task Builder, lists of error messages and reserved symbols and a glossary of terms used in this manual.

This manual describes the handling of an example application, CALC. In the first treatment of CALC, you build a task using all the default assumptions. Successive treatments illustrate the main points of each chapter in a realistic manner. Qualifiers and options are added as they are required, an overlay structure is defined when the task increases in size, the loading of overlays is optimized, and finally a shareable global area is added.

The memory allocation (MAP) files for the various stages of task development are included. The effect of a change can be observed by examining the map for the previous example and the map for the example in which the change is made.

# 2 PDS Commands

## 2.1 Introduction

This chapter describes PDS command sequences used to build tasks. Each command sequence is presented (using examples) from the simplest case to the most complex. The commands are then summarized as a set of syntactic rules. The example at the end of this chapter illustrates a task-building sequence for a typical application.

## 2.2 PDS Commands

If you write a FORTRAN program that you enter through a text editor as file CALC.FTN, you should then type the following commands in response to the program development system (PDS) prompt for input:

```
PDS> FORTRAN CALC
PDS> LINK CALC
PDS> RUN CALC
```

The first command (FORTRAN) causes the default FORTRAN compiler to translate the source language of the file CALC.FTN into a relocatable object module in the file CALC.OBJ. The second command (LINK) causes the Task Builder to process the file CALC.OBJ to produce the task image file CALC.TSK. Finally, the third command (RUN) causes the task to execute.

This example includes the command:

```
PDS> LINK CALC
```

This command illustrates the simplest LINK command sequence. It produces a task file, CALC.TSK, and is equivalent to the following command sequence:

```
PDS> LINK/TASK:CALC
FILE? CALC
```

### 2.2.1 LINK Command Formats

Typical LINK command formats are presented below:

```
PDS> LINK[command qualifiers] parameters
```

or:

```
PDS> LINK[command qualifiers]
FILE? parameters
```

or:

```
$LINK[command qualifiers] parameters
```

where:

- [command qualifiers] = task attributes and optional Task Builder output files. See Chapter 4 for a complete description of command qualifiers.
- parameters = one or more input file specifications.

## 2.2.2 LINK Command

A LINK command contains up to three different output files (a task image file, a memory allocation (MAP) file, and a symbol definition file) that you specify by command qualifiers. One or more input file specifications must also be included as parameters in a LINK command. Input and output files are identified using standard IAS file specifications.

When input file specifications are entered on the same line as the command qualifiers, at least one space is required between the last command qualifier and the first input file specification. If an input file specification is not entered on the same line as the qualifiers, PDS prompts FILE? and waits for input. When more than one input file specification is entered, the file specifications must be separated with one or more spaces, or tabs and/or a comma.

The Task Builder combines the input files to create a single executable task image and produces the output files as determined by the command qualifiers. A task image file is produced either by default or by the explicit use of the /TASK qualifier. Generation of the task image file can be inhibited by prefixing the TASK keyword with the letters NO, that is, /NOTASK inhibits the generation of a task image file. A memory allocation file, which identifies the size and location of the components within the task, is produced on the line printer by explicit use of the /MAP qualifier. Explicit use of the /MAP:filespec qualifier also produces a memory allocation file.

The /SYMBOLS qualifier must be specified to produce a symbol definition file, which contains the global symbol definitions in the task and their virtual or relocatable addresses in a format suitable for reprocessing by the Task Builder.

Output files assume the file name of the first input file unless a file specification is included with their respective qualifiers. The default file types are .TSK for the task image file, .MAP for the memory allocation file, and .STB for the symbol definition file.

Typical LINK commands and their interpretations are presented in Table 2–1 to illustrate the various LINK command formats.

**Table 2–1 Typical LINK Command Formats**

| Command | Interpretation |
|---|---|
| PDS> LINK/TASK:IMG1/MAP:MP1/SYMBOLS:SF1<br>FILE? IN1 | The task image file is IMG1.TSK, the memory MP1.MAP, allocation file is the symbol definition file is SF1.STB. and the input file is IN1.OBJ. |
| PDS> LINK/TASK:IMG1 IN1,IN2 | The task image file is IMG1.TSK, and the input files are IN1.OBJ and IN2.OBJ. |
| PDS> LINK/MAP:MP1 IN1,IN2 | The task image file is IN1.TSK, the memory allocation file is MP1.MAP, and the input files are IN1.OBJ and IN2.OBJ. |
| PDS> LINK/SUMBOLS:SF1 IN1 | The task image file is IN1.TSK, the symbol definition file is SF1.STB, and the input file is IN1.OBJ. |

Table 2–1 (Cont.) Typical LINK Command Formats

| Command | Interpretation |
|---|---|
| PDS> LINK/NOTASK/MAP:MP1<br>FILE? IN1 | This is a diagnostic run with no output files other than a map. However, any errors encountered will produce relevant error messages. Such a run is useful when a task has been found to exceed its memory limits. The input file is IN1.OBJ. |

## 2.2.3 Multiple Line Input

The LINK command can contain any number of command qualifiers to designate the desired task attributes and one or more input file specifications. If the command is too long to be entered on a single line (greater than 70 characters) or you wish to use more than one line, type a hyphen (-) as the last character in a line and continue the command on the next line.

For example, the sequence:

```
PDS> LINK/TASK:IMG1/MAP-
/SYMBOLS:SF1 IN1,IN2,IN3
```

produces the same results as the following command line:

```
PDS> LINK/TASK:IMG1/MAP/SYMBOLS:SF1 IN1,IN2,IN3
```

This sequence causes the Task Builder to process input files IN1.OBJ, IN2.OBJ. and IN3.OBJ, producing task image file IMG1.TSK and symbol definition file SF1.STB. The memory allocation file is output by default to the line printer, but it is not retained.

## 2.2.4 Options

Options specify the characteristics of the task being built. If you type the command qualifier /OPTIONS with the LINK command, PDS prompts for input by displaying OPTIONS? on the line following the last line of the command. You should enter one of the task builder options and terminate the line. Prompting continues on successive lines until you type a slash (/) as the first character after an OPTIONS? prompt to end the option input sequence. For example:

```
PDS> LINK/OPTIONS
FILE? IN1,IN2,IN3
OPTIONS? PRI=100
OPTIONS? SGA=JRNAL:RO
OPTIONS? /
```

In this sequence, the PRI=100 and SGA=JRNAL:RO are entered. The syntax and interpretation of each IAS Task Builder option is presented in Chapter 4.

The general form of an option is a keyword followed by an equal sign (=) and an argument list. The arguments in the list are separated from one another by colons. In the example given, the first option consists of the keyword PRI and a single argument 100 indicating that the task is to be assigned the priority 100. The second option consists of the keyword SGA and an argument list JRNAL:RO, indicating that the task accesses a shareable global area named JRNAL and the access is read-only.

Some options have argument lists that can be repeated. The symbol comma (,) separates the argument lists. For example:

```
OPTIONS? SGA=JRNAL:RO,RFIL:RW
```

In this command, the first argument list indicates that the task has requested read-only access to the shareable global area (SGA) JRNAL. The second argument list indicates that the task has requested read-write access to the shareable global area RFIL.

The following two sequences are equivalent:

```
OPTIONS? SGA=JRNAL:RO,RFIL:RW
```

and

```
OPTIONS? SGA=JRNAL:RO
OPTIONS? SGA=RFIL:RW
```

## 2.2.5 Indirect Command File Facility

You can enter the LINK command and any options directly or as a text file to be invoked later through the indirect command file facility.

To use the indirect command file facility, you first prepare a file that contains the required commands. Then, the contents of the indirect command file are processed by typing @ followed by the file specification.

If you prepare the text file AFIL.CMD as follows:

```
LINK/TASK:IMG1/MAP:MP1/OPTIONS
IN1,IN2,IN3
PRI=100
SGA=JRNAL:RO
/
```

Later, you can type:

```
PDS> @AFIL
```

When the symbol @ is encountered, search for commands is directed to the file specified following the @ symbol. While PDS is accepting input from an indirect file, prompting messages are not displayed on the terminal. The one-line command that references the indirect file AFIL.CMD is equivalent to the following keyboard sequence:

```
PDS> LINK/TASK:IMG1/MAP:MP1/OPTIONS
FILE? IN1,IN2,IN3
OPTIONS? PRI=100
OPTIONS? SGA=JRNAL:RO
OPTIONS? /
```

When PDS encounters a slash in the indirect file, the link command input is terminated. The Task Builder is invoked to build the task and, upon completion, the Task Builder exits to PDS. However, if PDS encounters an end-of-file in the indirect file before a slash, it returns its search for commands to the terminal and prompts for input.

Three levels of nesting are permitted in file references, that is, the indirect file referenced in a command sequence can contain a reference to another indirect file, which in turn references a third indirect file.

Suppose the file BFIL.CMD contains all the standard options that are used by a particular group of users at an installation. That is, every programmer in the group uses the options in BFIL.CMD. To include these standard options in a task building file, modify AFIL.CMD to include an indirect file reference to BFIL.CMD as a separate line in the option sequence.

Then the contents of AFIL.CMD are:

```
LINK/TASK:IMG1/MAP:MP1/OPTIONS
IN1,IN2,IN3
PRI=100
SGA=JRNAL:RO
@BFIL
/
```

Suppose the contents of BFIL.CMD are:

```
STACK=100
UNITS=5
ASG=DT1:5
```

The terminal equivalent of the command:

```
PDS> @AFIL
```

is then:

```
PDS> LINK/TASK:IMG1/MAP:MP1/OPTIONS
FILE? IN1,IN2,IN3
OPTIONS? PRI=100
OPTIONS? SGA=JRNAL:RO
OPTIONS? STACK=100
OPTIONS? UNITS=5
OPTIONS? ASG=DT1:5
OPTIONS? /
PDS>
```

An indirect file reference within an indirect command file must appear as a separate line. For example, if AFIL.CMD were modified by adding the @BFIL reference on the same line as the SGA=JRNAL:RO option, the substitution would not take place and an error would be reported.

A command file that contains only the command qualifiers and parameters for a LINK command can be used as an indirect command file. In this case, the LINK command must be stated explicitly before the indirect reference. For example, if file AFIL.CMD contains the following:

```
/TASK:IMG1/MAP:LP0:
IN1,IN2
```

then the command file is invoked indirectly by typing:

```
PDS> LINK @AFIL
```

This sequence is equivalent to:

```
PDS> LINK/TASK:IMG1/MAP:LP0:
FILE? IN1,IN2
```

## 2.2.6 Comments

Comment lines can be included at any point in the sequence. A comment line begins with an exclamation mark (!) and is terminated by a carriage return. All text on such a line is a comment. Comments can be included in an option line. In this case, the text between the exclamation mark and the carriage return is a comment.

Consider the annotation of the file described in Section 2.2.5; you add comments to provide more information about the purpose and the status of the task. Specifically, some identifying lines are added along with notes on the function of the input files and the shareable global area. Then, a comment on the current status of the task is added at the end of the file. The content of the file is as follows:

```
!
task 33a
!
! data from group e-46 weekly
!
! in1 contains processing routines
! in2 contains statistical tables
! in3 contains additional controls
link/task:img1/map:mp1/options -
in1,in2,in3
pri=100
sga=jrnal:ro  !rate tables
! task still in development
/
```

## 2.2.7 File Specification

The examples so far have been illustrated in terms of filenames. The standard IAS conventions for file specifications are used for all task building. For any file, you can specify the device, the User File Directory (UFD), the filename, the filetype, and the version number.

The file specification has the form:

```
device:[ufd]filename.filetype;version
```

For example:

```
PDS> LINK/TASK:IMG1/MAP:MP1
FILE? IN1,IN2,IN3
```

when the files are specified by name only, the default assumptions for device:,[ufd], filename, filetype, and version are applied. For example, if the user's default UFD which was specified at authorization time (or changed for the session by SET DEFAULT) is [200,200] and the user's default device is SY0:, the task image file specification of the example is assumed to be:

```
SY0:[200,200]IMG1.TSK;1
```

That is, the task image file is produced on the user's default device under UFD [200,200]. The default filetype for a task image file is .TSK and if the name IMG1.TSK is new, the version number is 1. The default settings for all the command qualifiers also apply. Qualifier defaults are described in full in Chapter 4.

Consider the following commands:

```
PDS> LINK/CHECKPOINT/DEBUG/TASK:[20,23]IMG1/MAP:TI:
FILE? IN1,IN2.OBJ;3,IN3
```

This sequence of commands produces the task image file IMG1.TSK under UFD [20,23] on the user's default device. The task image is checkpointable and contains the standard debugging aid (ODT). The memory allocation file is produced on the user's terminal. The task is built from the latest versions of IN1.OBJ and IN3.OBJ and an early version, number 3, of IN2.OBJ. The input files are all found in the user's default UFD on the user's default device.

For some files, a device specification is sufficient. In the above example, the memory allocation file is fully specified by the device TI:. The memory allocation file is produced on the terminal, but it is not retained.

In this example, CHECKPOINT and DEBUG qualifiers are used. The format and meaning for each qualifier are given in Chapter 4.

## 2.3 EXAMPLE: VERSION 1 OF CALC

An example task, CALC, is developed in this manual from the simple case given here through successive refinements and increasing complexity. The successive versions of CALC are designed to summarize the major points of each chapter and to illustrate possible uses for the facilities described.

As the first step in the development of the task CALC, three separate FORTRAN routines are entered by means of a text editor, translated by the FORTRAN IV compiler, and built into a task by the Task Builder.

All example tasks in this manual assume that FORTRAN IV is the default FORTRAN compiler.

The routines are:

- RDIN - which reads and analyzes input data and selects a data processing routine on the basis of the analysis
- PROC1 - which processes the input according to a specified set of rules
- RPRT - which outputs the results as series of reports

The three routines communicate with each other through a common block named DTA.

In these examples, all files are in the user's default directory unless otherwise specified. See the *IAS PDS User's Guide*.

### 2.3.1 Entering the Source Language

Enter and save the source for the FORTRAN programs of the example CALC with the text editor EDIT. Invoke EDIT and type in the source for the FORTRAN programs. The relevant parts of the programs are shown below:

```
PDS> EDIT
FILE? RDIN.FTN
[EDI -- CREATING NEW FILE]
INPUT
C   READ AND ANALYZE INPUT DATA,
C
C     SELECT A PROCESSING ROUTINE
C
C     ESTABLISH COMMON DATA BASE
C
    COMMON /DTA/ A(200), I
C     READ IN RAW DATA
    READ (6,1) A
1   FORMAT (200F6.2)
    . . .
C     CALL DATA PROCESSING ROUTINE
    CALL PROC1
C     GENERATE REPORT
    CALL RPRT
    . . .
    END

*EX
[EXIT]

PDS> EDIT
FILE? PROC1.FTN
[EDI -- CREATING NEW FILE]
INPUT
    SUBROUTINE PROC1
C   FIRST DATA PROCESSING ROUTINE
C   COMMUNICATION REGION
    COMMON /DTA/A(200),I
    ...
    RETURN
    END

*EX
[EXIT]

PDS> EDIT
FILE? RPRT.FTN
[EDI -- CREATING NEW FILE]
INPUT
      SUBROUTINE RPRT
C   INTERIM REPORT PROGRAM
C   COMMUNICATION REGION
    COMMON /DTA/ A(200),I
    ...
    RETURN
    END

*EX

[EXIT]
```

## 2.3.2 Compiling the FORTRAN Programs

The FORTRAN programs are compiled by the following sequence:

```
PDS> FORTRAN RDIN
PDS> FORTRAN PROG1
PDS> FORTRAN RPRT
```

The first command directs the FORTRAN IV compiler to take source input from RDIN.FTN and place the relocatable object code in RDIN.OBJ. The remaining commands perform similar actions for the source files PROG1.FTN and RPRT.FTN.

### 2.3.3 Building the Task

The task image for the three programs is built as follows:

```
PDS> LINK/TASK:CALC.TSK;1/MAP:MP1
FILE? RDIN,PROG1,RPRT
```

The task building command specifies the name of the task image file (CALC.TSK;1), the name of the memory allocation file (MP1.MAP), and the names of the input files (RDIN.OBJ, PROG1.OBJ and RPRT.OBJ). The task makes use of all the default assumptions for qualifiers and options.

## 2.4 Summary of Syntax Rules

In the syntax rules that follow, the symbol ... indicates repetition. For example,

```
input-filespec, ...
```

means one or more input-filespec items separated by spaces, tabs and/or commas; that is, one of the following forms:

```
input-filespec
input-filespec, input-filespec
input-filespec, input-filespec, input-filespec
...
```

As another example,

```
arg: ...
```

means one or more arguments separated by colons.

Another example,

```
OPTIONS? option
...
```

means one or more options.

As a final example, an item in brackets:

```
[command qualifiers]
```

means the entry is optional and the brackets are not a part of the command. This rule has one exception: brackets must be used to enclose a [UFD] specification, see Rule 6 in Section 2.4.1, below.

## 2.4.1 Syntax Rules

The syntax rules are as follows:

1 A task-building command can have one of several forms. The first form is a single line:

```
PDS> LINK[command qualifiers] parameters
```

or:

```
$LINK[command qualifiers] parameters
```

The second form has additional lines for input file names:

```
PDS> LINK[command qualifiers]
FILES? parameters
```

The third form allows the specification of options:

```
PDS> LINK[command qualifiers]/OPTIONS
FILES? parameters
OPTIONS? option-line
...
OPTIONS? terminating-symbol
```

The terminating symbol is a single slash (/).

The fourth form allows the use of indirect command files in one of the following formats:

```
PDS> @indirect-filespec
```

or:

```
PDS> LINK @indirect-filespec
```

where indirect-filespec is a file specification following standard IAS conventions.

2 The [command qualifiers] list contains one or more command qualifiers in the following format:

```
/keyword
```

or:

```
/NOkeyword
```

The keywords for the command qualifiers are presented in Chapter 4.

3 The parameter list contains one or more input file specifications following the standard IAS conventions (see 6. below).

4 An option-line can be one of the following:

```
option
```

or:

```
@indirect-filespec
```

where indirect-filespec is a file specification.

5 An option has the form:

```
keyword = argument-list, ...
```

where the argument-list is

```
arg: ...
```

The syntax for each of the options is given in Chapter 5.

6 A file specification conforms to standard IAS conventions and has the following form:

```
device:[ufd]filename.filetype;version
```

The components are defined as follows:

- device = name of the physical device where the volume containing the desired file is mounted. The name consists of two ASCII characters followed by a 1- or 2-digit octal unit number and a colon (:); for example, LP0: or DT1:. A logical device name can also be used.
- ufd = UFD where the file is recorded. [ufd] has the form

  ```
  [group, member]
  ```

  where group and member are both in the range 1 through 377 (octal).

  For example, member 220 of group 200 would require the following entry:

  ```
  [200,220]
  ```

- filename = name of the desired file. The file name can be from 1 to 9 alphanumeric characters, for example, CALC.
- filetype = 3-character filetype identification. Filename and filetype are always separated by a period (.). Files with the same name but a different function are distinguished from one another by the filetype; for example, CALC.TSK and CALC.OBJ might be the task file and object file, respectively, for the program CALC.
- version = octal version number of the file in the range 1 through 77777 (octal). Filetype and version are always separated by a semicolon (;). Various versions of the same file are distinguished from each other by this number; for example, CALC.OBJ;1 and CALC.OBJ;2.

The device, the UFD code, the filetype, and the version specifications are all optional.

Table 2–2 shows the default assumptions applied to missing components of a file specification.

**Table 2–2 File Specification Defaults**

| Item | Default | |
|---|---|---|
| device | User's current default device | |
| ufd | User's current default [ufd] | |
| filetype | Task image | TSK |
| | Memory allocation | MAP |
| | Symbol definition | STB |

Table 2–2 (Cont.) File Specification Defaults

| Item | Default | |
|---|---|---|
| | Object module | OBJ |
| | Object module library | OLB |
| | Overlay description | ODL |
| | Command | CMD |
| version | For an input file, the highest-numbered existing version. | |
| | For an output file, one greater than the highest-numbered existing version. | |

# 3 MCR COMMANDS

## 3.1 Introduction

This chapter describes MCR command sequences that can be used to build tasks. Each command sequence is presented (using examples), from the simplest case to the most complex. All commands are then summarized by a set of syntactic rules. The example at the end of this chapter illustrates a task building sequence for a typical application.

If you enter a FORTRAN program through a text editor as file PROG, type the following commands in response to the monitor console routine (MCR) prompt for input:

```
MCR>FOR CALC=PROG
MCR>TKB IMG=CALC
MCR>INS IMG
MCR>RUN IMG
```

The first command (FOR) causes the FORTRAN compiler to translate the source language of the file PROG.FTN into a relocatable object module in the file CALC.OBJ. The second command (TKB) causes the Task Builder to process the file CALC.OBJ to produce the task image file IMG.TSK. The third command (INS) causes Install to add the task to the directory of executable tasks. Finally, the fourth command (RUN) causes the task to execute.

The example just given includes the command:

```
MCR>TKB IMG=CALC
```

This command illustrates the simplest use of the Task Builder. It gives the name of a single file as output and the name of a single file as input. This chapter describes, first by example and then by syntactic definition, the complete facility for the specification of input and output files to the Task Builder.

### 3.1.1 Task Command Line

The task command line contains the output file specifications followed by an equal sign and the input file specifications. The task command line can have up to three output files and any number of input files.

The ouput files must be given in a specific order. The first file named is the task image file, the second is the memory allocation file, and the third is the symbol definition file. The memory allocation file contains information about the size and location of components within the task. The symbol definition file contains the global symbol definitions in the task and their virtual or relocatable addresses in a format suitable for re-processing by the Task Builder. The Task Builder combines the input files to create a single executable task image.

Any of the output file specifications can be omitted. When all three output files are given, the task-command line has the form:

```
task-image, mem-allocation, symbol-definition=input, ...
```

The following commands show the way the output filenames are interpreted.

**Table 3–1 Output Filename Defaults in the Task Command Line**

| Command | Output Files |
|---|---|
| MCR>TKB IMG1,MP1,SF1=IN1 | The task image file is IMG1.TSK, the memory allocation file is MP1.MAP, and the symbol definition file is SF1.STB. |
| MCR>TKB IMG1=IN1 | The task image file is IMG1.TSK. |
| MCR>TKB ,MP1=IN1 | The memory allocation file is MP1.MAP. |
| MCR>TKB ,,SF1=IN1 | The symbol definition file is SF1.STB. |
| MCR>TKB IMG1,,SF1=IN1 | The task image file is IMG1.TSK and the symbol definition file is SF1.STB. |
| MCR>TKB =IN1 | This is a diagnostic run with no output files. However, any errors encountered will produce a relevant error message. |

## 3.1.2 Multiple Line Input

Although you can have only three output files, you can have any number of input files. When several input files are used, a more flexible format, consisting of several lines, is necessary. This multiline format is also required for the inclusion of options, as discussed in the next section.

If you type TKB alone, MCR invokes the Task Builder. The Task Builder then prompts for input until it receives a line consisting of only the terminating sequence (//).

The sequence:

```
MCR>TKB
TKB>IMG1,MP1=IN1
TKB>IN2,IN3
TKB>//
```

produces the same result as the single line command:

```
MCR>TKB IMG1,MP1=IN1,IN2,IN3
```

This sequence produces the task image file IMG1.TSK and the memory allocation file MP1.MAP from the input files IN1.OBJ, IN2.OBJ, and IN3.OBJ.

The output file specifications and the separator (=) must appear on the first TKB command line. Input file specifications can begin or continue on subsequent lines.

The terminating symbol (//) directs the Task Builder to stop accepting input, build the task, and return to the MCR level.

## 3.1.3 Options

Use options to specify the characteristics of the task being built. If you type a single slash (/), the Task Builder requests option information by displaying ENTER OPTIONS: and prompting for input.

```
MCR>TKB
TKB>IMG1,MP1=IN1
TKB>IN2,IN3
TKB>/
ENTER OPTIONS:
TKB>PRI=100
TKB>SGA=JRNAL:RO
TKB>//
MCR>
```

In the above sequence, the user entered the options PRI=100 and SGA=JRNAL:RO, then typed a double slash to end option input.

**WARNING: For an overlaid task, where the input file has the /MP switch (see the MA command in Chapter 4), Task Builder automatically expects options. The single slash must not be entered if options are required. For example:**

```
MCR>TKB
TKB>OVTSK,OVTSK=OVODL/MP
ENTER OPTIONS:
TKB>TASK=...OVT
TKB>ASG=TI:1:2
TKB>//
```

The syntax and interpretation of each Task Builder option are given in Chapter 5.

The general form of an option is a keyword followed by an equal sign (=) and an argument list. The arguments in the list are separated from one another by colons. In the example given, the first option consists of the keyword PRI and a single argument 100 indicating that the task is to be assigned the priority 100. The second option consists of the keyword SGA and an argument list JRNAL:RO, indicating that the task accesses a shareable global area (SGA) named JRNAL and the access is read-only.

More than one option can be given on a line. The symbol exclamation mark (!) is used to separate options on a single line. For example:

```
TKB>PRI=100 ! SGA=JRNAL:RO
```

is equivalent to the two separate lines

```
TKB>PRI=100
TKB>SGA=JRNAL:RO
```

Some options have argument lists that can be repeated. The symbol comma (,) is used to separate the argument lists. For example:

```
TKB>SGA=JRNAL:RO,RFIL:RW
```

In this command, the first argument list indicates that the task has requested read-only access to the shareable global area JRNAL. The second argument list indicates that the task has requested read-write access to the shareable global area RFIL.

The following sequences are equivalent:

- Sequence 1:

```
TKB>SGA=JRNAL:RO,RFIL:RW
```

- Sequence 2:

```
TKB>SGA=JRNAL:RO ! SGA=RFIL:RW
```

- Sequence 3:

```
TKB>SGA=JRNAL:RO
TKB>SGA=RFIL:RW
```

## 3.1.4 Multiple Task Specification

If more than one task is to be built, the symbol,(/) (slash), can be used to direct the Task Builder to stop accepting input, build the task, and request information for the next task build.

Consider the sequence:

```
MCR>TKB
TKB>IMG1=IN1
TKB>IN2,IN3
TKB>/
ENTER OPTIONS:
TKB>PRI=100
TKB>SGA=JRNAL:RO
TKB>/
TKB>IMG2=SUB1
TKB>//
MCR>
```

The Task Builder accepts the output and input file specifications and the option input, then stops accepting input when it encounters the (/) during option input. The Task Builder builds IMG1.TSK and returns to accept more input.

## 3.1.5 Indirect Command File Facility

Enter the sequence of commands to the Task Builder directly or entered as a text file and later invoked through the indirect command file facility.

To use the indirect command file facility, first prepare a file that contains the user command input for the desired interaction with the Task Builder. Then, the contents of the indirect command file are invoked by typing @ followed by the file specification.

For example, the text file AFIL can be prepared as follows:

```
IMG1,MP1=IN1
IN2,IN3
/
PRI=100
SGA=JRNAL:RO
//
```

Later, you can type:

```
MCR>TKB @AFIL
```

When the Task Builder encounters the symbol @, it directs its search for commands to the file specified following the @ symbol. When the Task Builder is accepting input from an indirect file, it does not display prompting messages on the terminal. The one-line command that enables the Task Builder to accept commands from the indirect file AFIL is equivalent to the keyboard sequence:

```
MCR>TKB
TKB>IMG1,MP1=IN1
TKB>IN2,IN3
TKB>/
ENTER OPTIONS:
TKB>PRI=100
TKB>SGA=JRNAL:RO
TKB>//
```

When the Task Builder encounters a double-slash in the indirect file, it terminates indirect file processing, builds the task, and exits to MCR upon completion.

However, if the Task Builder encounters an end-of-file in the indirect file before a double slash, it returns its search for commands to the terminal and prompts for input.

The Task Builder permits three levels of nesting in file references, that is, the indirect file referenced in a terminal sequence can contain a reference to another indirect file, which in turn references a third indirect file.

Suppose the file BFIL.CMD contains all the standard options that are used by a particular group at an installation. That is, every programmer in the group uses the options in BFIL.CMD. To include these standard options in a task building file, you modify AFIL to include an indirect file reference to BFIL.CMD as a separate line in the option sequence.

Then the contents of AFIL.CMD are:

```
IMG1,MP1=IN1
IN2,IN3
/
PRI=100
SGA=JRNAL:RO
@BFIL
//
```

Suppose the contents of BFIL.CMD are:

```
STACK=100
UNITS=5 ! ASG=DT1:5
```

The terminal equivalent of the command

```
MCR>TKB @AFIL
```

is then:

```
MCR>TKB
TKB>IMG1,MP1=IN1
TKB>IN2,IN3
TKB>/
ENTER OPTIONS:
TKB>PRI=100
TKB>SGA=JRNAL:RO
TKB>STACK=100
TKB>UNITS=5 ! ASG=DT1:5
TKB>//
MCR>
```

The indirect file reference must appear as a separate line. For example, if AFIL.CMD were modified by adding the @BFIL reference on the same line as the SGA=JRNAL:RO option, the substitution would not take place and an error would be reported.

## 3.1.6 Comments

Comment lines can be included at any point in the sequence. A comment line begins with a semicolon (;) and is terminated by a carriage return. All text on such a line is a comment. Comments can be included in option lines. In this case, the text between the semicolon and the carriage return is a comment.

Consider the annotation of the file just described; the user adds comments to provide more information about the purpose and the status of the task. Specifically, some identifying lines are added along with notes on the function of the input files and shareable global area. Then, a comment on the current status of the task is added at the end of the file. The content of the file is as follows:

```
;
; TASK 33A
;
; DATA FROM GROUP E-46 WEEKLY
;
IMG1,MP1=
;
;           PROCESSING ROUTINES
;
    IN1
;
;           STATISTICAL TABLES
;
            IN2
;
;           ADDITIONAL CONTROLS
;
            IN3
/
PRI=100
;
SGA=JRNAL:RO ; RATE TABLES
;
; TASK STILL IN DEVELOPMENT
;
//
```

## 3.1.7 File Specification

The examples so far have been illustrated in terms of filenames. The Task Builder adheres to the standard conventions for file specifications. For any file, you can specify the device, the user file directory (UFD), the filename, the type, the version number, and any number of switches.

Thus, the file specification has the form:

```
device:[ufd] filename.type;version/sw...
```

For example:

```
MCR>TKB
TKB>IMG1,MP1=IN1
TKB>IN2,IN3
TKB>//
```

when the files are specified by name only, the default assumptions for device:,[ufd], filename, type, version and switch settings are applied. For example, if the user identification code under which you logged in was [200,200], the task image file specification of the example is assumed to be:

```
SY0:[200,200]IMG1.TSK;1
```

That is, the task image file is produced on the system device (SY0) under user file directory [200,200]. The default type for a task image file is TSK and since the name IMG1.TSK is new, the version number is 1. The default settings for all the task image switches also apply. Switch defaults are described in full in Chapter 4.

Consider the following commands:

```
MCR>TKB
TKB>[20,23]IMG1/CP/DA,LP:=IN1
TKB>IN2;3,IN3
TKB>//
```

This sequence of commands produces the task image file IMG1.TSK under user file directory [20,23] on the system device. The task image is checkpointable and contains the standard debugging aid. The memory allocation file is produced on the line printer. The task is built from the latest versions of IN1.OBJ and IN3.OBJ and an early version, number 3, of IN2.OBJ. The input files are all found on the system device.

For some files, a device specification is sufficient. In the above example, the memory allocation file is fully specified by the device LP. The memory allocation file is produced on the line printer, but is not retained as a file.

In this example, switches CP and DA are used. The code, syntax and meaning for each switch are given in Chapter 4.

## 3.2 EXAMPLE: VERSION 1 OF CALC

An example task, CALC, is developed in this manual from the simple case given here through successive refinements and increasing complexity. The successive versions of CALC are designed to summarize the major points of each chapter and to illustrate possible uses for the facilities described.

As the first step in the development of the task CALC, three separate FORTRAN routines are entered by means of a text editor, translated by the FORTRAN compiler, and built into a task by the Task Builder.

The routines are:

- RDIN - which reads and analyzes input data and selects a data processing routine on the basis of the analysis.
- PROCI - which processes the input according to a specified set of rules.
- RPRT - which outputs the results as a series of reports.

The three routines communicate with each other through a common block named DTA.

In these examples, all files are in the UFD under which the user logged in to the system via the MCR>HEL[LO] command (see the IAS MCR User's Guide) unless otherwise specified.

## 3.2.1 Entering the Source Language

Enter and file the source for the FORTRAN programs of the example CALC by means of the text editor EDI. The user invokes EDI and types in the source for the FORTRAN programs. The relevant parts of the programs are shown below:

```
MCR>EDI RDIN.FTN
[EDI -- CREATING NEW FILE]
INPUT
C     READ AND ANYLZE INPUT DATA,
C
C       SELECT A PROCESSING ROUTINE
C
C       ESTABLISH COMMON DATA BASE
C
        COMMON /DTA/ A(200), I
C       READ IN RAW DATA
        READ (6,1) A
1     FORMAT (200F6.2)
      . . .
C       CALL DATA PROCESSING ROUTINE
        CALL PROC1
C       GENERATE REPORT
        CALL RPRT
      . . .
      END

*EX
[EXIT]
MCR>EDI PROC1.FTN
[EDI -- CREATING NEW FILE]
INPUT
        SUBROUTINE PROC1
C     FIRST DATA PROCESSING ROUTINE
C     COMMUNICATION REGION
      COMMON /DTA/A(200),I
      ...
      RETURN
      END

*EX
[EXIT]
MCR>EDI RPRT.FTN
[EDI -- CREATING NEW FILE]
INPUT
      SUBROUTINE RPRT
C     INTERIM REPORT PROGRAM
C     COMMUNICATION REGION
      COMMON /DTA/ A(200),I
      ...
      RETURN
      END
*EX
[EDI -- EXIT]
```

### 3.2.2 Compiling the FORTRAN Programs

Compile the FORTRAN programs by the following sequence:

```
MCR>FOR
FOR>RDIN,LRDIN=RDIN
FOR>PROC1,LPROC1=PROC1
FOR>RPRT,LRPRT=RPRT
```

The first command invokes the FORTRAN compiler. The second command directs the compiler to take source input from RDIN.FTN, place the relocatable object code in RDIN.OBJ and write the listing in LRDIN.LST. The remaining commands perform similar actions for the source files PROC1 and RPRT.

### 3.2.3 Building the Task

The task image for the three programs is built in the following way:

```
MCR>TKB CALC;1,LP:=RDIN,PROC1,RPRT
```

The task building command specifies the name of the task image file (CALC.TSK;1), The device for the memory allocation file (LP) and the names of the input files (RDIN.OBJ, PROC1.OBJ and RPRT.OBJ). The task makes use of all the default assumptions for switches and options.

## 3.3 Summary of Syntax Rules

In the syntax rules, the symbol . . . indicates repetition. For example,

```
input-spec, ...
```

means one or more input-spec items separated by commas, that is, one of the following forms:

```
input-spec
input-spec, input-spec
input-spec, input-spec, input-spec
 . . .
```

Examples:

```
arg: ...
```

means one or more arg items separated by colons.

```
TKB>input-line
...
```

means one or more of the indicated TKB input-line items.

### 3.3.1 Syntax Rules

The syntax rules are as follows:

1 A task-building-command can have one of several forms. The first form is a single line:

```
MCR>TKB task-command-line
```

The second form has additional lines for input file names:

```
MCR>TKB
TKB>task-command-line
TKB>input-line
...
TKB>terminating-symbol
```

The third form allows the specification of options:

```
MCR>TKB
TKB>task-command-line
TKB>/
ENTER OPTIONS:
TKB>option-line
...
TKB>terminating-symbol
```

The fourth form has both input lines and option lines:

```
MCR>TKB
TKB>task-command-line
TKB>input-line
...
TKB>/
ENTER OPTIONS:
TKB>option-line
...
TKB>terminating-symbol
```

The terminating symbol can be:

/ if more than one task is to be built, or
// if control is to return to MCR.

2 A task-command-line has one of the three forms:

```
output-file-list = input-file, ...
= input-file, ...
@indirect-file
```

where indirect-file is a file specification as defined in Rule 7.

3 An output-file-list has one of the three forms:

```
task-file, mem-allocation-file, symbol-file
task-file, mem-allocation-file,
task-file
```

where task-file is the file specification for the task image file; mem-allocation-file is the file specification for the memory allocation file; and symbol-file is the file specification for the symbol definition file. Any of the specifications can be omitted, so that, for example, the form:

```
task-file,,symbol-file
```

is permitted.

4 An input-line has either of the forms:

```
input-file, ...
@indirect-file
```

where input-file and indirect-file are file specifications.

5 An option-line has either of the forms:

```
option ! ...
@indirect-file
```

where indirect-file is a file specification.

6 An option has the form:

```
keyword = argument-list, ...
```

where the argument-list is

```
arg: ...
```

The syntax for each of the options is given in Chapter 4.

7 A file specification conforms to standard conventions. It has the form

```
device:[ufd]filename.type;version/sw...
```

The components are defined as follows:

- device - is the name of the physical device on which the volume containing the file is mounted. The name consists of two ASCII characters followed by an optional 1- or 2-digit octal unit number; for example, 'LP' or 'DT1'.
- ufd - is the user file directory number consisting of two octal numbers each of which is in the range of 1 through 377 (octal). These numbers must be enclosed in brackets and separated by a comma, and must be in the following format:

  ```
  [group,member]
  ```

  For example, member 220 of group 200 would use the following entry:

  ```
  [200,220]
  ```

- filename - is the name of the file. The file name can be from 1 to 9 alphanumeric characters, for example, CALC.
- type - is the 3-character type identification. Files with the same name but different function are distinguished from one another by the file type; for example, CALC.TSK and CALC.OBJ.
- version - is the octal version number of the file in the range 1 through 77777 (octal).

Versions of the same file are distinguished from each other by this number; for example, CALC;1 and CALC;2.

- sw - is a switch specification. More than one switch can be used, each separated from the previous one by a '/'. The switch is a 2-character alphabetic name which identifies the switch option. The permissible switch options and their syntax are given in Chapter 4.

The device, the user file directory code, the type, the version, and the switch specifications are all optional.

Table 3–2 applies to missing components of a file specification.

**Table 3–2 File Specification Defaults**

| Item | Default | |
|---|---|---|
| device | SY0, the system device | |
| group | The group number currently in effect[1] | |
| member | The member number currently in effect[1] | |
| type | Task image | TSK |
| | Memory allocation | MAP |
| | Symbol definition | STB |
| | Object module | OBJ |
| | Object module library | OLB |
| | Overlay description | ODL |
| | Indirect command | CMD |
| version | For an input file, the highest-numbered existing version. | |
| | For an output file, one greater than the highest-numbered existing version. | |
| switch | (The default for each switch is given in Chapter 4.) | |

[1]If an explicit device or [ufd] is given, it becomes the default for subsequent files separated by commas on the same side of the equal (=) sign. For example: DT1:IMG1,MP1=IN1,DF:IN2,IN3

| File | Device |
|---|---|
| IMG1.TSK | DT1 |
| MP1.MAP | DT1 |
| IN1.OBJ | SY0 |
| IN2.OBJ | DF0 |
| IN3.OBJ | DF0 |

# 4 Qualifiers and Switches

## 4.1 Introduction

This chapter describes how you can modify the actions of the Task Builder (TKB) by using:

MCR switches
PDS qualifiers

When you use the PDS LINK command, you can include qualifiers that control Task Builder output by specifying simple task attributes, optional task builder output files, and so on. PDS qualifiers are described in Section 4.2.

MCR switches are equivalent to PDS qualifiers. MCR switches are described in Section 4.3.

## 4.2 PDS Qualifiers

With PDS, qualifiers are applied to either the LINK command or to file specifications within the command. These are called command qualifiers and file qualifiers.

### 4.2.1 Command Qualifiers

When entered in a LINK command, each qualifier is preceded by a slash, and either the complete keyword or a unique abbreviation of the keyword is typed following the slash. If a qualifier is not specified, default assumptions are made; therefore, you must negate a positive default assumption by typing the letters NO before the keyword (or abbreviation) if the corresponding function is not desired. For example, the command qualifier /NOTASK inhibits the generation of the task image file.

### 4.2.2 Examples

The following command sequences illustrate the use of qualifiers and file specifications, and the resulting interpretation.

| Terminal Sequence | Interpretation |
|---|---|
| PDS> LINK/TASK:IMG1/CHE/DEB-/NOMAP IN1/NOCON | The task IMG1.TSK is checkpointable and includes the LB0:[1,1]ODT.OBJ debugging aid. Use the first object module in input file IN1. |
| PDS> LINK/TASK:IMG2/PRI/MAP: (MP1/SHO)FILE? IN2 [1,1]EXEC.STB | The task IMG2.TSK is an executive privileged task. The short form of the memory allocation file MP1.MAP is requested. The inputs for the task are the file IN2.OBJ and the symbol definition file SY0:[1,1]EXEC.STB that links the task to the subroutines and data base of the Executive. |
| PDS> LINK/TASK:IMG3/DEB:DBG1FILE? IN3 LB1/LIB:(SUB1:SUB2) LB1/LIB | The task IMG3.TSK contains the input file IN3.OBJ, the modules SUB1 and SUB2 from the library file LB1, and the debugging aid DBG1.OBJ. The library file LB1.OLB is specified a second time without arguments so that the Task Builder will search the file for undefined global references. |
| PDS> LINK/TASK:IMG4/EXIT:5-/OVERLAY:TREE | The Task IMG4.TSK is built from the overlay description contained in the file TREE.ODL. If more than five diagnostics occur, the Task Builder aborts the run. |

## 4.3 MCR Switches

The syntax for a file specification in an MCR command, as given in Section 3.1.7, is:

```
dev:[ufd]filename.type;version/sw-1/sw-2.../sw-n
```

The file specification concludes with optional switches: sw-1, sw-2, ..., sw-n.

When a switch is not specified, the Task Builder establishes a setting for the switch, called a default assumption.

A switch is designated by a two-character code. The code is an indication that the switch applies or does not apply. For example, if the switch code is CP (task can be checkpointed), the recognized switch settings are:

| | |
|---|---|
| /CP | The task is checkpointable. |
| /-CP | The task is not checkpointable. |
| /NOCP | The task is not checkpointable. |

Switches are used primarily for the following purposes:

- To designate the task attributes recorded in the task image file during task build and in the System Task Directory (STD) entry on Install.
- To instruct TKB to interpret the input file in a special way (for example, /DA is used when the task contains a debugging aid).
- To control the listing of the memory allocation file (for example, /SH is used to request the short memory allocation file).

### 4.3.1 Task Builder Switches

This section describes the switches recognized by the Task Builder. For each switch, the following information is given:

1 The MCR switch mnemonic.

2 The default assumption made if the MCR switch is not present.

3 The PDS qualifier mnemonic.

4 The default assumption made if the PDS qualifier is not present.

5 The file(s) to which the switch or qualifier can be applied.

6 A description of the effect of the switch on the Task Builder.

Table 4–1 gives an alphabetical listing and summary information about the switch codes enabled by the Task Builder. The subsections that follow the table give a more detailed description for each switch.

**Table 4–1 MCR Switches and PDS Qualifiers**

| MCR Switch | MCR Default | PDS Qualifier | PDS Default | Applicable File Type | Effect on Task Builder |
|---|---|---|---|---|---|
| /AB | /AB | /ABORT | /ABOR | T | Task can be aborted. |
| /CC | /CC | /CONCATENATED | /CONCATENATED | I | Input file can contain more than one object module. |
| /CO | /-CO | none | none | T | Causes task builder to build a shared global area. |
| /CP | /CP | /CHECKPOINT | /CHEC | T | Task can be checkpointed. |
| /CR | /-CR | /CROSS_REFERENCE | /NOCRO | M | Memory allocation is to include a global symbol cross reference listing. |
| /DA | /-DA | /DEBUG | /NODE | T,I | Task contains a debugging aid. |
| /DL | /-DL | /DEFAULT LIBRARY | none | I | Specified library file is a replacement for the default system object module library. |
| /DS | /DS | /DISABLE | /DISA | T | Task can be disabled. |
| /FP | /FP | /FLOATING_POINT | /FLOA | T | Task used the floating point processor. |
| /FR | /FR | /FLUSH_RECEIVE_ QUEUES | /FLUSH | T | Task receive queues are flushed each time it exits. |
| /FU | /-FU | /FULL_SEARCH | /NOFULL | T | Search all co-tree overlay segments for matching definition or reference when processing modules from the default object module library. |
| /FX | /-FX | /FIX | /NOFIX | T | Task can be fixed in memory. |
| /HD | /HD | /HEADER | /HEAD | T,S | Task can be fixed in memory. |
| /LB | /-LB | /LIBRARY | /NOLIBRARY | I | Input file is an object module. |
| /LI | /-LI | none | none | T | Instructs TKB to build a shared library. |
| /MA | /MA | /MAP | /MAP | M | Include all modules in the memory allocation file. |
| /-MA | /MA | none | none | I | Exclude all modules in this input file from the memory allocation file. |
| /MP | /-MP | /OVERLAY_DESCRIPTION | /NOOV | I | Input file contains an overlay description. |

**Key to Applicable File Type**

T—Task image file
S—Symbol definition file
M—Memory allocation file
I—Input file

Table 4–1 (Cont.) MCR Switches and PDS Qualifiers

| MCR Switch | MCR Default | PDS Qualifier | PDS Default | Applicable File Type | Effect on Task Builder |
|---|---|---|---|---|---|
| /MU | /-MU | /MULTIUSER | /NOMU | T | Task is multiuser. |
| /NM | /-NM | none | none | .TSK | Tells TKB to inhibit two diagnostic messages. |
| /OR | /OR | /RUN_TIME_SYSTEM | /RUN | T | Runtime system is included in overlaid task. |
| /PI | /-PI | /POSITION_INDEPENDENT | /NOPO | T,S | Task code is position independent. |
| /PR | /-PR | /PRIVILEGED | /NOPR | T | Task has privileged access rights. |
| /RO | /-RO | /RESIDENT_OVERLAYS | /NORES | T | Memory-resident overlay operator (!) is enabled so task can be built with memory-resident overlays. |
| /RW | /-RW | /READ_WRITE | /NOREA | T | Task has read-write access to read-only code. |
| /SE | /SE | /RECEIVE | /RECE | T | Send data can be received. |
| /SP | /SP | none | none | M | Memory allocation file is spooled. |
| /SQ | /-SQ | /SEQUENTIAL | /NOSEQ | T | Task p-sections are allocated sequentially. |
| /SR | /-SR | /REQUEST | /NOREQ | T | Send [by reference] and request/resume accepted from non real-time directive privileged tasks. |
| /SS | /-SS | /SELECT | /NOSELECT | I | Selective Symbol Search. |
| /TR | /-TR | /TRACE | /NOTR | T | Task is to be traced. |
| /UN | /UN | /SYMBOLS:(filespec/[NO] UNDEFINED_SYMBOLS) | /UNDEF | S | Include references to undefined symbols in symbol table file. |
| /UR | /UR | none | none | M | Print undefined references on initiating terminal. |
| /WN | /WN | /WAIT_FOR_NODES | /WAIT | T | System waits a certain period of time for nodes to become available. |
| /XT:n | /-XT | /EXIT:n | /EXIT:1 | T | Task Builder exits after n errors, where n is a decimal number. |
| none | none | /LARGE_SYMBOL_TABLE | /NOLAR | none | Task Builder will have a large internal symbol table. |
| none | none | /MAP | /NOMAP | M | Produces memory allocation file. |
| /FI | /FI | /FILES | /NOFIL | M | Include file-by-file analysis of allocation. |
| /FU | /FU | /FULL | /NOFULL | M | Include all modules in map. |

**Key to Applicable File Type**

T—Task image file
S—Symbol definition file
M—Memory allocation file
I—Input file

Table 4–1 (Cont.) MCR Switches and PDS Qualifiers

| MCR Switch | MCR Default | PDS Qualifier | PDS Default | Applic-able File Type | Effect on Task Builder |
|---|---|---|---|---|---|
| /NA | /NA | /NARROW | /WIDE | M | Make map in 72-column format. |
| /SH | /SH | /SHORT | /SHORT | M | Make summary map. |
| /WI | /WI | /WIDE | /WIDE | M | Make map in 132-column format. |
| /UR | /UR | /UNDEFINED_ REFERENCES | /UNDEF | M | Print undefined references on initiating terminal. |
| none | none | /OPTIONS | /NOOP | none | Apply Task Builder options specified after command string. |
| none | none | /SYMBOLS | /NOSY | S | Produces a symbol table file. |
| none | none | /TASK[:filespec] | /TASK | T | Produces a task image file. |

**Key to Applicable File Type**

T—Task image file
S—Symbol definition file
M—Memory allocation file
I—Input file

# /ABORT (/AB)

## PDS QUALIFIER

/ABORT
/ABOR (Default)

## MCR SWITCH

/AB
/AB (Default)

### file

task image

### effect

The Task Builder clears the nonabortable flag in the task label block flag word.

### meaning

The task can be aborted when it is running.

**Note: A task running under the control of the IAS scheduler can always be aborted, even if it is built non-abortable.**

# /CHECKPOINT (/CP)

## PDS QUALIFIER

/CHECKPOINT
/CHECKPOINT (Default)

## MCR SWITCH

/CP
/CP (Default)

### file

task image

### effect

The Task Builder clears the noncheckpointable flag in the task label block flags word.

### meaning

Task can be checkpointed.

# /CONCATENATED

## PDS QUALIFIER

/CONCATENATED
/CONCATENATED

## MCR SWITCH

/CC
/CC

### file

input

### effect

The Task Builder includes in the task image all the modules in the file. If this switch is negated, the Task Builder includes in the task image only the first module in the file.

### meaning

The file can contain one or more than one object module.

# /CROSS_REFERENCE (/CR)

## PDS QUALIFIER

/CROSS_REFERENCE
/NOCRO (Default)

## MCR SWITCH

/CR
/-CR (Default)

### file

memory allocation

### effect

A cross reference listing, as described in Section 6.4, is appended to the memory allocation file. PDS users must have privileges that enable use of TCP and chaining. The system manager sets such privileges when authorizing a PDS user. For further details see the *IAS System Management Guide*.

### meaning

A global symbol cross reference listing is to be produced.

# /DEBUG[:filespec] (/DA)

## PDS QUALIFIER

/DEBUG[:filespec]
/NODEBUG (Default)

## MCR SWITCH

/DA
/-DA (Default)

### file

task image or input

### effect

If filespec is not specified, the Task Builder links the task with the system's debugging aid (ODT) contained in the file LB0:[1,1]ODT.OBJ.

If filespec is specified the Task Builder links the task with the debugging aid contained in the specified file. The user-generated debugging aid must be in object format. See Appendix E for information on including a debugging aid.

### meaning

The task image file is to include a debugging aid.

# /DEFAULT_LIBRARY:filespec (/DL)

## PDS QUALIFIER

/DEFAULT_LIBRARY:filespec
LB:[1,1]SYSLIB.OLB (Default)

## MCR SWITCH

/DL
LB:[1,1]SYSLIB.OLB (Default)

### file

input

### effect

This file, which must be an object module library, will be searched instead of the system library LB0:[1,1]SYSLIB.OLB when Task Builder is resolving undefined global symbol references.

If the specified library is empty (that is, no modules have been inserted into it) the effect is as though there were no default library. The DL switch can be applied only to a single input file.

### meaning

The specified file is used in place of the system object module library.

- If /DL is specified—Use filespec as Default Library
- If /-DL is specified—Use No Default Library
- If nothing is specified—Use LB0:[1,1]SYSLIB.OLB

# /DISABLE (/DS)

## PDS QUALIFIER

/DISABLE
/DISABLE (Default)

## MCR SWITCH

/DS
/DS (Default)

### file

task image

### effect

The Task Builder clears the non-disable flag in the task label flags word.

### meaning

The task can be disabled.

# /EXIT:n (/XT:n)

## PDS QUALIFIER

/EXIT:n
/EXIT:1 (Default)

## MCR SWITCH

/XT:n
/-XT (Default)

### file

task image

### effect

The Task Builder exits after n (decimal) error diagnostics have been produced. If n is not specified, it is assumed to be 1.

### meaning

The Task Builder exits after n error diagnostics have been produced. The number of diagnostics can be specified as a decimal or octal number, using the convention:

- For MCR:
  - n. = A decimal number
  - #n or n = An octal number
- For PDS:
  - n = A decimal number (always)

If n is not specified, it is assumed to be 1.

# /FIX (/FX)

## PDS QUALIFIER

/FIX
/NOFIX (Default)

## MCR SWITCH

/FX
/-FX (Default)

### file

task image

### effect

The Task Builder clears the non-fixable flag in the task label block flags word. Note that a fixed task cannot be checkpointed even when built as checkpointable.

### meaning

The task can be fixed in memory.

# /FLOATING_POINT (/FP)

## PDS QUALIFIER

/FLOATING_POINT
/FLOATING_POINT (Default)

## MCR SWITCH

/FP
/FP (Default)

### file

task image

### effect

The Task Builder allocates 25 words in the task header for the floating point save area.

### meaning

The task used the Floating Point Processor.

# /FLUSH_RECEIVE_QUEUES (/FR)

## PDS QUALIFIER

/FLUSH_RECEIVE_QUEUES
/FLUSH (Default)

## MCR SWITCH

/FR
/FR (Default)

### file

task image

### effect

N/A

### meaning

The task is to have its receive queues (data and references) flushed each time it exits. If this qualifier is negated, information in the receive queues will be retained until it is received.

# /FULL_SEARCH (/FU)

## PDS QUALIFIER

/FULL_SEARCH
/NOFULL_SEARCH (Default)

## MCR SWITCH

/FU
/-FU (Default)

### file

task image

### effect

If the switch is negated, unintended global references between co-tree overlay segments are eliminated. Global Definitions from the default library are restricted in scope to references in the main root and the current tree. Use of this switch is described in Chapter 7, Section "Resolution of Global Symbols from the Default Library".

### meaning

The Task Builder searches all co-tree overlay segments for a matching definition or reference when processing modules from the default object module library.

# /HEADER (HD)

## PDS QUALIFIER

/HEADER
/HEADER (Default)

## MCR SWITCH

/HD
/HD (Default)

### file

task image or symbol definition

### effect

The Task Builder constructs a header in the task image. The contents of the header are described in Section C.2. If you are using the RUN command to run or install a task, you must build the task with a header.

### meaning

A header is to be included in the task image. You must use the negated form of this qualifier (/NOHEADER) when building a shareable global area.

# /LARGE_SYMBOL_TABLE

## PDS QUALIFIER

/LARGE_SYMBOL_TABLE
/NOLARGE_SYMBOL_TABLE (Default)

## MCR SWITCH

Specify the Slow Task Builder; by specifying STB at the MCR> prompt.

### file

None

### effect

Invokes the task . . . STB instead of the usual task . . . TKB.

### meaning

Select a version of the Task Builder that has a large internal symbol table (that is, the slow Task Builder (see Section F.3).

# /LIBRARY (/LB)

## PDS QUALIFIER

/LIBRARY
/NOLIBRARY (Default)

## MCR SWITCH

/LB
/-LB (Default)

### file

input

### effect

1 If no arguments are specified, the Task Builder searches the file to resolve undefined global references and extracts from the library for inclusion in the task image any modules that contain definitions for such references.

2 If arguments are specified, the Task Builder includes only the named modules in the task image.

**Note: If you want the Task Builder to search a library file both to resolve global references and to select named modules for inclusion in the task image, the library file must be named twice. The first time it must be specified with the LB switch and no arguments to direct the Task Builder to search the file for undefined global references, and a second time with the desired modules to direct the Task Builder to include those modules in the task image being built.**

## meaning

This switch has two forms:

1 Without arguments: LB

2 With arguments: LB:mod-1:mod-2 . . . :mod-8

The interpretation of the switch depends on the form.

1 If the switch is applied without arguments, the input file is assumed to be a library file of reclocatable object modules (created by the Librarian) that is to be searched for the resolution of undefined global references.

2 If the switch is applied with arguments, the input file is assumed to be a library file of relocatable object modules from which the modules named in the argument list are to be taken for inclusion in the task image. The module names are those defined at assembly time by the .TITLE directive (or if no .TITLE directive, the filename (first 6 characters) when inserted by the Librarian). Up to a maximum of eight modules can be specified.

# /MAP (/MA)

## PDS QUALIFIER

/MAP
/MAP (Default)

**Note: /NOMAP, implicitly or explicitly qualifying an input file, is overridden by the memory allocation file qualifier /FULL (see Table 4–1 and Section "MAP [:filespec] or MAP:(filespec/qualifers)".**

## MCR SWITCH

/MA switch on input file.
/MA (Default)

### file

input

### effect

All modules are included in the memory allocation file.

### meaning

The input file is to be included in the memory allocation map.

# /MAP[:filespec] or /MAP:(filespec/qualifiers)

## PDS QUALIFIER

/MAP[:filespec]
/NOMAP (Default)

## MCR SWITCH

Include a MAP file specification.

### file

memory allocation

### effect

If you specify the filespec, you can omit the file. This this case, the Task Builder assumes the .MAP filetype.

If filespec is not specified, the memory allocation file is printed on the line printer.

The following qualifiers can be applied to filespec:

| | |
|---|---|
| /FILES | Include file-by-file analysis of memory allocation and symbol definition. This produces a separate section for each input module showing the PSECT allocations and symbols defined in the module.<br>MCR equivalent: /-SH |
| /FULL | Include all modules in the memory allocation file, even those that explicitly or by default have the /NOMAP input file qualifier.<br>MCR equivalent: /MA |
| /NARROW | Produce a map 72 characters wide, suitable for printing on a terminal.<br>MCR equivalent: /-WI |
| /SHORT | Produce summary map, equivalent to /NOFILES/NOFULL.<br>MCR equivalent: /SH /-MA |
| /UNDEFINED_ REFERENCES | Print any undefined references on the terminal that initiated the task build.<br>MCR equivalent: /UR |

| | |
|---|---|
| /WIDE | Produce a map 132 characters wide, suitable for printing on a line printer.<br>MCR equivalent: /WI |
| Default | filespec qualifiers: /SHORT /WIDE /UNDEF |

## meaning

Produce a memory allocation file.

# /MULTIUSER (/MU)

## PDS QUALIFIER

/MULTIUSER
/NOMULTIUSER (Default)

## MCR SWITCH

/MU
/-MU (Default)

### file

task image

### effect

The multi-user is set in the task label block flags word and any read-only section of the root segment is aligned on a disk boundary.

### meaning

Multiple versions of the task can run simultaneously. Note that only read-write parts of the task will be duplicated in memory.

# /OPTIONS

## PDS QUALIFIER

/OPTIONS
/NOOPTIONS (Default)

## MCR SWITCH

/ (slash in a line by itself)
No default

### file

None

### effect

In interactive (PDS>) mode, the Task Builder issues an "OPTIONS?" prompt after the input files have been specified. The user enters an option specification and after each option specification is received, another prompt is issued. To terminate the list of options, the user types a slash (/) as the first character following the prompt.

In batch mode or when using an indirect command file, one or more options are expected to be specified in the LINK command. In MCR mode one or more options are expected to be specified in the lines following the switch. A slash (/) in the first character position of a line terminates the list of options.

### meaning

Apply Task Builder options following options qualifier or switches in the LINK command.

# /OVERLAY_DESCRIPTION:filespec (/MP)

## PDS QUALIFIER

/OVERLAY_DESCRIPTION:filespec
/NOOVERLAY_DESCRIPTION (Default)

## MCR SWITCH

/MP
/-MP (Default)

### file

input

### effect

The Task Builder receives all the input file specifications from this file and allocates memory as directed by the overlay description.

**Note:**

1 **After /MP the Task Builder automatically prompts for options. If options are required they must be entered straight away, not preceded by the input consisting of a single slash.**

2 **When an overlay description file is specified as the input file for a task, it must be the last input file specified. Other input files are automatically assigned to the ROOT segment of the task.**

### meaning

Link the task according to the overlay structure defined in the file identified by filespec. Overlay descriptions are discussed in Chapter 7, Section "Resolution of Global Symbols from the Default Library."

# /POSITION_INDEPENDENT (/PI)

## PDS QUALIFIER

/POSITION_INDEPENDENT
/NOPOSITION_INDEPENDENT (Default)

## MCR SWITCH

/PI
/-PI (Default)

### file

task image or symbol definition

### effect

The Task Builder sets the Position Independent Code (PIC) attribute flag in the task label block flag word.

### meaning

The task contains only position independent code or data. This qualifier should only be used in conjunction with /NOHEADER when building a shareable global area. Position independent shareable global areas are described in Section 9.4.

# /PRIVILEGED (/PR)

## PDS QUALIFIER

/PRIVILEGED
/NOPRIVILEGED (Default)

## MCR SWITCH

/PR
/-PR (Default)

### file

task image

### effect

The Task Builder sets the Privileged Attribute flag in the task label block flag word.

### meaning

The task is executive privileged with respect to memory access rights. The task can access the external page, and the SCOM data area (including node pool) in addition to its own task space. Executive privileged tasks are described in Section 6.2.1.

# /READ_WRITE (/RW)

## PDS QUALIFIER

/READ_WRITE
/NOREAD_WRITE (Default)

## MCR SWITCH

/RW
/-RW(Default)

### file

task image

### effect

This enables you to debug read-only code online.

### meaning

Task is to have read-write access to code specified as read-only.

# /RECEIVE (/SE)

## PDS QUALIFIER

/RECEIVE
/RECEIVE (Default)

## MCR SWITCH

/SE
/SE (Default)

### file

task image

### effect

None

### meaning

The task is able to receive data sent to it by the SEND DATA and SEND BY REFERENCE directives. If the qualifier is negated, any attempt to send data or send data by reference to the task will fail as though it was not installed.

# /REQUEST (/SR)

## PDS QUALIFIER

/REQUEST
/NOREQUEST (Default)

## MCR SWITCH

/SR
/-SR (Default)

### file

task image

### effect

None

### meaning

The task is to be built so that the Executive allows the following directives to be issued to the task from non-real-time directive privileged tasks:

| | |
|---|---|
| VSDR$/SDRQ$ | Send data and request or resume receiver. |
| SRFR$ | Send data by reference and request or resume receiver. |

# /RESIDENT_OVERLAYS (/RO)

## PDS QUALIFIER

/RESIDENT_OVERLAYS
/NORES (Default)

## MCR SWITCH

/RO
/-RO (Default)

### file

task image

### effect

The memory-resident overlay operator (!) is enabled, and is used to construct a task image that contains one or more memory-resident overlay segments. If this switch is negated, the operator is checked for correct syntactical usage, but no memory-resident overlay segments are created.

### meaning

The memory-resident overlays, as described in Section 7.1.2.

# /RUN_TIME_SYSTEM (/OR)

## PDS QUALIFIER

/RUN_TIME_SYSTEM
/RUN_TIME (Default)

## MCR SWITCH

/OR
/OR (Default)

### file

task image

### effect

None

### meaning

If this switch is negated, the overlay run-time system and its associated control area will not be included in an overlaid task. This type of task cannot be run in the normal way but might be useful for special applications.

# /SELECT (/SS)

## PDS QUALIFIER

/SELECT
/NOSELECT (Default)

## MCR SWITCH

/SS
/-SS (Default)

### file

input

### effect

The Task Builder includes only the required symbol definitions from the specified file as distinct from all global symbols of that file. This qualifier is useful when an input file is the symbol table output (.STB file) of another task build, because it reduces the size of symbol table searches.

### meaning

The input file is to be used only to define the required symbols.

# /SEQUENTIAL (/SQ)

## PDS QUALIFIER

/SEQUENTIAL
/NOSEQUENTIAL (Default)

## MCR SWITCH

/SQ
/-SQ (Default)

### file

task image

### effect

The Task Builder does not re-order the program sections alphabetically. This qualifier must not be used for modules that rely upon alphabetical program section allocation; in IAS such modules include FORTRAN I/O handling and File Control System modules from SYSLIB.

### meaning

The task image is constructed from the specified program sections in the order stated in the LINK command. Chapter 6, Section "Sequential Allocation of P-sections" describes the allocation of the task image and gives an example that shows the allocation performed under the default assumption and the allocation performed when the /SEQUENTIAL qualifier is specified.

# /SYMBOLS[:filespec]

## PDS QUALIFIER

/SYMBOLS[:filespec]
/NOSYMBOLS (Default)

## MCR SWITCH

Include a symbol table file specification.

### file

None

### effect

If you specify filespec, you can omit the file type field. In this case, the Task Builder assumes it is .STB.

If filespec is not specified, the first input file name becomes the symbol definition file name and .STB becomes the file type.

### meaning

Produce a symbol definition file.

# /SYMBOLS:(filespec[/NO]UNDEFINED_SYMBOLS) (/UN)

## PDS QUALIFIER

/SYMBOLS:(filespec[/NO]UNDEFINED_SYMBOLS
/UNDEF (Default)

## MCR SWITCH

/UN
/UN (Default)

### file

symbol table

### effect

None

### meaning

The symbol table (STB) file is used to include references for symbols that were undefined in the task. If this qualifier is negated, undefined symbols will be ignored when the STB file is generated.

# /TASK[:filespec]

## PDS QUALIFIER

/TASK[:filespec]
/TASK (Default)

## MCR SWITCH

Include a Task File specification.

### file

None

### effect

If you specify filespec, you can omit the file type. In this case, the Task Builder assumes the .TSK filetype.

### meaning

Produce a task image file.

# /TRACE (/TR)

## PDS QUALIFIER

/TRACE
/NOTRACE (Default)

## MCR SWITCH

/TR
/-TR (Default)

### file

symbol table

### effect

The Task Builder sets the T bit in the initial processor status (PS) word of the task. When the task is executed, a trace trap occurs on the completion of each instruction.

### meaning

The task is to be traced.

# /WAIT_FOR_NODES (/WN)

## PDS QUALIFIER

/WAIT_FOR_NODES
/WN (Default)

## MCR SWITCH

/WN
/WN (Default)

### file

task image

### effect

Executive will stall the task and not allow the directive to complete until either sufficient nodes have been obtained or a certain period (normally 500 clock ticks) has elapsed without finding sufficient nodes. In the latter case an error return will be made to the task.

### meaning

Many system directives require space to be allocated from the system node pool. If there is insufficient space available, the use of this qualifier will cause the directive to wait a short time for nodes to become available. If WAIT_FOR_NODES is not specified, and insufficient nodes are available, an immediately error return will be made to the task.

# 5 Task Builder Options

Where more complex specifications are needed to describe a modification (for example, numeric values, names or lists) you must use Task Builder options. The Task Builder options are identical for both MCR and PDS users and are summarized in Table 5–1.

The task builder user includes options to supply task characteristics that require a more complex specification than can be included using a qualifier (PDS) or switch (MCR).

You always input options as a result of a prompt. In PDS, you get the prompt "OPTIONS?" by including the qualifier "/OPTIONS" in the command (see Section 2.2.5). In MCR, you get the prompt "ENTER OPTIONS:" by typing a single slash (/) in response to a "TKB>" prompt (see Section 3.1.3).

Options fall into six categories, each of which is identified by the following mnemonics:

1 ident

  Identification options identify task characteristics. The task name, priority, (UIC), and partition can be specified by the use of options in this category.

2 alloc

  Allocation options modify the task memory allocation. The size of the stack, program-sections in the task, and FORTRAN work areas and buffers can be adjusted by the use of options in this category.

3 share

  Storage sharing options indicate the task's intention to access a shareable global area.

4 device

  Device specifying options specify the number of units required by the task and the assignment of physical devices to logical unit numbers (LUNs).

5 alter

  Content altering options define a global symbol and value or introduce patches in the task image.

6 synch

  Synchronous trap options define synchronous trap vectors.

Table 5–1 lists all the options alphabetically, including a brief description and interest range for each. Some of these options are of interest to all users of the system, some only to the FORTRAN programmer, and some primarily to the MACRO-11 programmer. The interest range is indicated by the following codes:

- F—Of interest to FORTRAN programmers only.
- M—Of interest to MACRO-11 programmers only.
- FM—Of interest to both FORTRAN and MACRO-11 programmers.

The table also lists the mnemonic for the category to which the option belongs. The remainder of the chapter gives more detailed descriptions of each option by category.

**Note: Real-time users can override many of these options when the task is explicitly installed. See the *IAS PDS User's Guide* or *IAS MCR User's Guide*.**

**If /NOOPTIONS is specified explicitly or by default in a PDS LINK command, the task is linked to the system SGA SYSRES; see Chapter 9, Section 9.1.2.**

Table 5–1 Task Builder Options

| Option | Meaning | Interest | Category |
|---|---|---|---|
| ABSPAT | Declare absolute patch values. | M | alter |
| ACTFIL | Declare number of files open simultaneously. | FM | alloc |
| ALVC | Provide the user with ABSolute (default) and DEFerred auto-load vectors. | FM | alter |
| ASG | Declare device assignment to logical units. | FM | device |
| ATRG | Declare the number of attachment descriptor blocks to be created in the task header. | FM | alloc |
| BASE | Define lowest virtual address. | FM | alloc |
| CMPRT | Declares completion routine for supervisor-mode library. | FM | ident |
| EXTSCT | Declare extension of a program section. | M | alloc |
| EXTTSK | Extend task memory allocation at install time. | M | alloc |
| FMTBUF | Declare extension of buffer used for processing format strings at run-time. | F | alloc |
| GBLDEF | Declare a global symbol definition. | M | alter |
| GBLINC | Includes symbols in the .STB file | M | alter |
| GBLPAT | Declare a series of patch values relative to a global symbol. | M | alter |
| GBLREF | Declare a global symbol reference. | FM | alter |
| GBLXCL | Declares global symbols to be excluded from the .STB file. | H,M | alter |
| IDENT | Declares the identification of the task. | H,M | ident |
| MAXBUF | Declare an extension to the FORTRAN record buffer. | F | alloc |
| MAXEXT | Declare maximum task extension. | FM | alloc |
| ODTV | Declare the address and size of the debugging aid synchronous system trap (SST) vector. | M | synch |
| PAR | Declare partition name and dimensions. | FM | ident |
| POOL | Declare pool usage limit. | FM | alloc |
| PRI | Declare priority. | FM | ident |
| RESAPR | Reserve APRs for use by memory management directives. | FM | alloc |
| RESSGA | Declare task's intention to access a shareable global area. | FM | share |
| RESSUP | Declares task's intention to access a resident supervisor-mode library. | H,M | share |
| SGA | Declare task's intention to access a shareable global area | FM | share |
| STACK | Declare the size of the stack. | FM | alloc |
| SUPLIB | Declares task's intention to access a system-owned supervisor-mode library. | H,M | share |
| SYMPAT | Declare a series of symbolic patch values. | M | alter |

Table 5–1 (Cont.) Task Builder Options

| Option | Meaning | Interest | Category |
|---|---|---|---|
| TASK | Declare the name of the task. | FM | ident |
| TOP | Define highest virtual address. | FM | alloc |
| TSKV | Declare the address of the task SST vector. | M | synch |
| UIC | Declare the user identification code under which the task runs. | FM | ident |
| UNITS | Declare the highest logical unit number. | FM | device |
| VSECT | Declare the virtual base address and size of a program section. | FM | alloc |

## 5.1 Identification Options

The identification options are used to specify task identifying information. These options are of interest to all real-time users.

The identification options specify the name of the task, the UIC, the priority, and the partition for real-time tasks. The UIC can be specified by a real-time user when the task is explicitly installed or when it is run. If such a specification is not made, the system uses the UIC established when the task was built. The task runs under the most recently specified UIC.

These options have no effect if the task is run under timesharing.

The identification options are as follows:

- CMPRT (Completion Routine)
- ALVC (Auto-Load Vector)
- IDENT (Task Identification)
- PAR (Partition)
- PRI (Priority)
- TASK (Task Name)
- UIC (User Identification Code)

# CMPRT (Completion Routine)

Use this option to identify a shared global area as a supervisor-mode library. The CMPRT option requires an argument that specifies the entry point of the completion routine in the library. The completion routine switches the processor from supervisor to user mode and returns program control to the user task after the supervisor-mode library subroutine that was called from the user task has executed.

Two completion routines are available in SYSLIB:

- $CMPCS restores only the carry bit in the user-mode PS.
- $CMPAL restores all the condition code bits in the user-mode PS.

These routines perform all the necessary overhead to switch the processor from supervisor to user mode and return program control to the user task at the instruction following the call to a supervisor-mode library subroutine.

Although you can write your own completion routines, it is best to use either $CMPCS or $CMPAL whenever possible.

## SYNTAX

**CMPRT**= *name*

where:

- name = 1- to 6-character Radix-50 name identifying the completion routine.

### default

None

# ALVC (Auto-Load Vector)

The ALVC option selects either node access to ABSolute (default) or DEFerred auto-load vectors.

## SYNTAX

**ALVC**= *ABS*

**ALVC**= *DEF*

where:

- ABS = Absolute addressing mode
- DEF = Deferred option

**NOTE: To replace $AUTO with your own auto-load routine, insert the name of the user-suppled auto-load routine in .NAUTO and specify ALVC=DEF option at taskbuild time. The RMS-11 V2.0 mapping routines use this method extensively. To intercept references that require auto-load services, RMS-11 temporarily swaps the $AUTO entry-point in .NAUTO with its own mapping routine. RMS-11 then sets the necessary segments and window descriptors before it transfers control to the $AUTO routine.**

### default

ABS

# IDENT (Task Identification)

The IDENT option changes the identification of the task from the one originally specified in the .IDENT MACRO-11 statement in the first .MAC file to the one specified in the option.

If you do not use the IDENT option, the Task Builder uses the identification of the first input .MAC file that it encounters.

## SYNTAX

**IDENT**= *name*

where:

- name = Any 1- to 6-character Radix-50 name for use as task identification. You can use any Radix-50 character that is correct for use in the MACRO-11 .IDENT statement.

### default

TKB supplies no default name. If you use the IDENT option, you must specify a name.

# PAR (Partition)

Unless it is explicitly overridden when a task is installed or run, for real-time tasks, the PAR option identifies the partition where the task runs.

**NOTE: For timesharing tasks, the task runs in the timesharing partition irrespective of this option.**

## SYNTAX

**PAR**= *pname*

where:

- pname = Name of the partition

### default

Timesharing partition for tasks running under control of the timesharing scheduler.

The default partition (specified during system generation) for real-time tasks.

# PRI (Priority)

For real-time tasks, the PRI option declares the priority at which the task executes. If priority is not specified when the task is installed, the priority declared in the PRI option is used.

**NOTE: For timesharing tasks, the task runs at the timesharing priority irrespective of this option.**

## SYNTAX

**PRI**= *priority-number*

where:

- priority-number = Decimal integer in the range 1 - 250

### default

System default priority.

# TASK (Task Name)

The TASK option specifies the installed task name.

## SYNTAX

**TASK**= *task-name*

where:

- task-name = 1- to 6-character alphanumeric name identifying the task.

### default

For tasks run using the PDS RUN filename command, the task is run with a task name in the following form:

- JOBnnn—For timesharing systems
- TTnnx—For multiuser systems, where nn = terminal unit number

For real-time tasks and tasks run using the MCR RUN filename command, the default taskname is the first six characters of the task image file name.

# UIC (User Identification Code)

For real-time tasks, the UIC option declares the UIC under which the task will run if no UIC was specified at execution request or when the task was installed.

**NOTE: On timesharing systems, the task runs under the UIC allocated when the user logged in, irrespective of specification of the UIC option.**

## SYNTAX

**UIC**= *[group,member]*

where:

- group = Octal number in the range 1 - 377 that specifies the group.
- member = Octal number in the range 1 - 377 that specifies the member number.

### default

The UIC determined from the user name at login time.

## 5.2 Allocation Options

The allocation options direct the Task Builder to change allocations affecting memory.

The allocation options are as follows:

- ACTFIL (Number of Active Files)
- ATRG (Attachment Descriptors)
- BASE (Base Address)
- EXTSCT (Program Section Extension)
- EXTTSK (Extend Task Space)
- FMTBUF (Format Buffer Size)
- MAXBUF (Maximum Record Buffer Size)
- MAXEXT (Maximum Extension)
- POOL (Pool Limit)
- RESAPR (Reserve APRs)
- STACK (Stack Size)
- TOP (Top Address)
- VSECT (Virtual Program Section)

# ACTFIL (Number of Active Files)

The ACTFIL option declares the number of files that the task can have open simultaneously. For each active file, an allocation of 520 bytes (or, if MAXBUF is specified, MAXBUF+8) is made.

If the number of active files used by a task is less than the default assumption of four, you can use the ACTFIL option to save space. If the number of active files is more than the default assumption, you must use the ACTFIL option to direct the Task Builder to make the additional allocation so that the task can run. If you use double buffered file control services (FCS), the ACTFIL specification must also be doubled.

The FORTRAN object time system (OTS) and file control services (FCS) must be included in the task image for the extension to take place. The p-section that is extended has the reserved name "$$FSR1".

## SYNTAX

**ACTFIL**= *file-max*

where:

- file-max = Decimal integer indicating the maximum number of files that can be open at the same time.

### default

ACTFIL = 4

# ATRG (Attachment Descriptors)

The ATRG option declares the number of attachment descriptors blocks to be created in the task header.

## SYNTAX

**ATRG**= *max-regions*

where:

- max-regions = Decimal integer in the range 0 to 240 that declares the maximum number of regions to which the task can simultaneously attach. Attachment descriptor blocks are automatically generated for resident overlays and SGAs.

### default

ATRG = 0

# BASE (Base Address)

The BASE option specifies the base address of the task to be at a particular 4K boundary.

Use the BASE option when you create SGA images that are not position-independent. The BASE (and TOP) options are primarily used to locate SGAs and must not be used when building normal tasks.

Task image addresses are normally allocated upward from zero. A non-position-independent library file must appear in the same virtual address range of each task that shares it. To avoid conflicts with task addresses, you can allocate the library toward the top of the virtual address range (that is, 140000 to 177776), by using a base address declaration (see also Section "TOP").

The BASE option overrides any previous TOP specification.

## SYNTAX

**BASE**= *bound:high*

**BASE**= *bound:low*

where:

- bound = Decimal number between 0 and 28 that specifies the lowest 4K boundary of the image.

### default

0

# EXTSCT (Program Section Extension)

The EXTSCT option declares an extension in size for a p-section. P-sections and their attributes are described in Chapter 6, Section 6.1.9.

If the p-section has the attribute CON (concatenated), the section is extended by the specified number of bytes. If the p-section has the attribute OVR (overlay), the section is extended only if the length of the extension is greater than the length of the p-section.

For example, suppose that p-section BUFF is 200 bytes long and the option below is given:

```
EXTSCT = BUFF:250
```

The extension specified for the p-section depends on the CON/OVR attribute; specifically:

- CON—The extension is 250 bytes.
- OVR—The extension is 50 bytes.

The extension occurs when the p-section name is encountered in an input object file or in the overlay description file.

## SYNTAX

**EXTSCT**= *p-sect-name:extension*

where:

- p-sect-name = 1- to 6-character alphanumeric name that specifies the p-section to be extended.
- extension = Octal integer that specifies the number of bytes by which to extend the p-section.

### default

None

# EXTTSK (Extend Task Space)

The size of the read/write space of the task is to be extended at Install time.

This parameter can be overridden by the Install or Run qualifier /INCREASE. If the EXTTSK option has been overridden, the task must be removed and reinstalled without the /INCREASE qualifier to revert back to the EXTTSK option.

This option is used in conjunction with the .LIMIT directive to the assembler and the system directive Get Task Parameters. It is useful in saving disk space that would otherwise be allocated (for example, for initially empty buffers). The Install /INCREASE qualifier provides the ability to vary the size of such buffers.

## SYNTAX

**EXTTSK**= *task-extension*

where:

- task-extension = Decimal number of words by which Install extends the upper read/write area of the task. The value is rounded up to the next 32-word block boundary.

### default

0

# FMTBUF (Format Buffer Size)

The FMTBUF option declares the length of internal working storage allocated for the parsing of format specifications at run-time. The length of this area must equal or exceed the number of characters in the longest format string to be processed.

Run-time processing occurs whenever an array is referenced as the source of formatting information within a FORTRAN I/O Statement. The program section to be extended has the reserved name "$$OBF1".

## SYNTAX

**FMTBUF**= *max-format*

where:

- max-format = Decimal integer larger than the default that specifies the number of characters in the longest format specification.

### default

FMTBUF = 132

# MAXBUF (Maximum Record Buffer Size)

The MAXBUF option declares the maximum record buffer size required for all files used by the task.

Use this option whenever you process a file where the maximum record size exceeds the default buffer length specified during system generation.

The FORTRAN Object Time System must be included in the task image for the extension to take place. The p-section that is extended has the reserved name "$$IOB1".

## SYNTAX

**MAXBUF**= *max-record*

where:

- max-record = A decimal integer, larger than the default, that specifies the maximum record size in bytes.

### default

MAXBUF = 132

# MAXEXT (Maximum Extension)

The MAXEXT option declares the maximum number of 32-word blocks by which the task can extend itself. To perform this extension, use the Extend Task directive (EXTK$) or the EXTTSK FORTRAN subroutine. The *IAS System Directives Reference Manual* describes the EXTK$ directive fully.

## SYNTAX

**MAXEXT**= *maximum-extension*

where:

- maximum-extension = Octal number of 32-word blocks in the range 0–2000.

### default

MAXEXT = 2000

# POOL (Pool Limit)

The POOL option declares the maximum number of 8-word pool nodes that the task can use simultaneously. Use these pool nodes for forming I/O request nodes and to process certain system directives. For a full description of system node pool usage, see the *IAS Executive Facilities Reference Manual.* If the task POOL allocation is too small, directives might fail with the error IE.UPN (unavailable pool node).

## SYNTAX

**POOL**=*pool-limit*

where:

- pool-limit = Decimal number of 8-word nodes in the range 1 to 255. For multiuser tasks this indicates the pool limit for each version.

### default

POOL=40

# RESAPR (Reserve APRs)

The RESAPR option reserves APRs for use at runtime by the Memory Management directives.

The task builder does not allocate the APRs specified in the directive Builder for resident overlays, global areas, or the task pure area.

## SYNTAX

**RESAPR**= *a1[:a2...]*

where:

- a1:a2:.... = APRs (in the range 1 to 7) to be reserved

### default

None

# STACK (Stack Size)

The STACK option declares the maximum size of the stack required by the task.

The stack is an area of memory used for temporary storage, subroutine calls, and interrupt service linkages. The stack is referenced by hardware register SP (the stack pointer).

## SYNTAX

**STACK**= *stack-size*

where:

- stack-size = Decimal integer that specifies the number of words required for the stack.

### default

STACK = 256

# TOP (Top Address)

The TOP option declares the ending address of a task to be within a 4K boundary.

This option is the same as the BASE option except that it allows definition of the last 4K boundary rather than the first 4K boundary.

The TOP option overrides any previous BASE specification.

## SYNTAX

**TOP**= *bound:high*

**TOP**= *bound:low*

where:

- bound = Decimal number between 0 and 28 that specifies the highest 4K boundary of the image.

### default

None

# VSECT (Virtual Program Section)

The VSECT option enables you to specify the virtual base address, virtual length, and physical memory allocated to the p-section.

## SYNTAX

**VSECT**= *p-section name:base:window:[:physical-length]*

where:

- p-sect-name = 1- to 6-character program section name.
- base = Octal value specifying the virtual base address of the program section in the range 0-177777. This value must be a multiple of 4K if used with the mapping directives.
- window = Octal value specifying the amount of virtual address space allocated to the p-section. Base plus window size must not exceed 177777 (octal).
- physical length = Octal value specifying the amount of physical memory to be allocated to the section in units of 64-byte blocks. This value, when added to the task image size (and any previous allocation) must not cause the total to exceed 2.2 million bytes. If unspecified, zero is assumed.

### default

Window defaults to the value allocated; physical length defaults to zero.

### 5.2.1 Example of Allocation Options

If the FORTRAN routines contained in file GRP1 use eight files simultaneously, and the maximum record length in one of these files is 160 characters, you can use the following terminal sequence to build the task:

```
PDS> LINK/TASK:IMG1/MAP:MP1/OPTIONS
FILE? GRP1
OPTIONS? ACTFIL=8
OPTIONS? MAXBUF=160
OPTIONS? /
```

**or:**

```
TKB>IMG1,MP1=GRP1
TKB>/
ENTER OPTIONS:
TKB>ACTFIL=8
TKB>MAXBUF=160
TKB>/
```

## 5.3 Storage-Sharing Options

You can use two options to indicate a task's intention to access an SGA.

1 SGA option

Use the SGA option to access public SGAs that contain commonly used routines or data. The task and symbol table files are expected to reside in UFD[1,1] on the pseudo device LB0: (normally the system disk).

2 RESSGA option

Use the RESSGA option to specify a device and UFD. You can also use this option to access SGAs that are private to a single user or group of users.

It is sometimes necessary to control access by non-owners to a data area or to enable writing to a code area. The access required by a particular task (read-only or read/write) and declared in either the SGA or RESSGA option is always subject to the access granted to non-owners by the SGA itself. The latter (read-only, read/write or no access) is determined when the SGA is installed. See the *IAS PDS User's Guide* or the *IAS MCR User's Guide* for a description of the appropriate INSTALL command.

The *IAS Executive Facilities Reference Manual* describes the different types of SGAs you can use.

**Note: If /NOOPTIONS was specified explicitly or by default in a PDS LINK command, the task is automatically linked to the public SGA SYSRES. See Chapter 9, Section 9.1.2.**

The storage-sharing options are as follows:

- RESSGA (Shareable Global Area)
- RESSUP (Resident Supervisor-Mode Library)
- SGA (Shareable Global Area)

# RESSGA (Shareable Global Area)

The RESSGA option declares a shareable global area for use by the task. RESSGA enables a full file-specification.

**NOTE: The RESSGA option supersedes the RESCOM and RESLIB options in previous versions of IAS. RESCOM and RESLIB are still recognized by the Task Builder for compatibility. Their effect is identical to that of specifying RESSGA.**

## SYNTAX

**RESSGA** = *filespec/access-code[:apr]*

where:

- filespec = Form dev:[ufd]filnam. No filetype can be specified and "filnam" must be six characters or less, since it is also the name of the shareable global area.
- access-code = As for the SGA option.
- apr = As for the SGA option.

### default

dev: and [ufd] default to the user default device and UFD.

# RESSUP (Resident Supervisor-Mode Library)

The RESSUP option declares that your task intends to access a user-owned, supervisor-mode library. The term *user-owned* means that the library and the symbol definition file associated with it can reside under any UFD that you choose. You can specify the UFD and remaining portions of the file specification. You must not place comments on the line with RESSUP.

## SYNTAX

**RESSUP**= *file-specification/[-]SV[:apr]*

where:

- file-specification = Memory image file of the supervisor-mode library.
- /[-]SV = Code /SV or /-SV to indicate whether TKB includes mode-switching vectors within the user task. If you specify /SV, TKB includes a 4-word, mode-switching vector within the address space of the user task for each call to a supervisor-mode library subroutine. If you specify /-SV, you must provide your own mode-switching vector. Providing your own mode-switching vectors is useful if your library contains threaded code. It is best to use the system-supplied vectors whenever possible.
- apr = Integer in the range 0 through 7 that specifies the first Supervisor Active Page Register that you want TKB to reserve for your supervisor-mode library. You can specify an APR only for position-independent, supervisor-mode libraries. The default is the lowest available APR.

  The library at virtual 0 must have the CSM dispatcher present in the system-supplied completion routine described in Chapter 9.

**NOTE: TKB expects to find a symbol definition file with the same name as that of the memory image file but with a file type of .STB, on the same device and under the same UFD as that of the memory image file.**

**Regardless of the version number you give in the file specification, TKB uses the latest version of the .STB file.**

### default

When you omit portions of the file specification, the following defaults apply:

- Terminal default directory
- Device—SY0:
- File type—.TSK
- File version—Latest

# SGA (Shareable Global Area)

The SGA option declares a shareable global area residing on LB0: under [1,1] for use by the task.

**NOTE: The SGA option supersedes the COMMON and LIBR options in previous versions of IAS. COMMON and LIBR are still recognized by the Task Builder for compatibility. Their effect is identical to specifying SGA.**

## SYNTAX

**SGA**= *SGA-name:access-code[:apr]*

where:

- SGA-name = 1- to 6-character alphanumeric name of the SGA.
- access-code = Either RW (read/write) or RO (read-only) to indicate the type of access required for the task.
- apr = Integer in the range 1 to 7 that specifies the first Active Page Register to be reserved for the common block. The apr is optional but must not be specified for non-position-independent common areas.

## default

None

# SUPLIB (Supervisor-Mode Library)

This option declares that your task intends to access a system-owned, supervisor-mode library. The term *system-owned* means that TKB expects to find the supervisor-mode library and the symbol definition file associated with it in UFD [1,1] on device LB:.

## SYNTAX

**SUPLIB**= *name:[-]SV[:apr]*

where:

- name = 1- to 6-character Radix-50 name specifying the system-owned, supervisor-mode library. TKB expects to find a symbol definition file having the same name as that of the library with a file version of .STB under [1,1] of device LB:.
- :[-]SV = Code /SV or /-SV to indicate whether TKB includes mode-switching vectors within the user task. If you specify /SV, TKB includes a 4-word mode-switching vector within the address space of the user task for each call to a supervisor-mode library subroutine. If you specify /-SV, you must provide your own mode-switching vector. Providing your own mode-switching vectors is useful if your library contains threaded code. It is best to use the system-supplied vectors whenever possible.
- apr = Integer in the range of 0 through 7 that specifies the first Supervisor Active Page Register that TKB is to reserve for the library. You can specify an APR only for position-independent, supervisor-mode libraries. The default is the lowest available APR. The library at virtual 0 must have the CSM dispatcher present in the system-supplied completion routine described in Chapter 9.

### default

None

### 5.3.1 Example of Storage Sharing Options

If the task composed of the MACRO-11 programs TST1 and TST2 accesses a shareable common area DTST that contains data, and a shareable library area STST that contains code, both held in LB0:[1,1], you can use the following terminal sequence to build the task:

```
PDS> LINK/TASK:CHK/MAP/OPTIONS
FILE? TST1,TST2
OPTIONS? SGA=DTST:RW
OPTIONS? SGA=STST:RO
OPTIONS? /
```

**or:**

```
TKB>CHK,LP:=TST1,TST2
TKB>/
ENTER OPTIONS:
TKB>SGA=DTST:RW
TKB>SGA=STST:RO
TKB>/
```

If the shareable global areas are not in LB0:[1,1]:, you can use the following sequence for the same task:

```
PDS> LINK/TASK:CHK/MAP/OPTIONS
FILE? TST1, TST2
OPTIONS? RESSGA=[200,30]DTST/RW
OPTIONS? RESSGA=DB1:[200,30]STST/RO
OPTIONS? /
```

**or:**

```
TKB>CHK,LP:=TST1,TST2
TKB>/
ENTER OPTIONS:
TKB>RESSGA=[200,30]DTST/RW
TKB>RESSGA=DB1:[200,30]STST/RO
TKB>/
```

## 5.4 Device Specifying Options

The two options in this category are of interest to all system users. The UNITS option declares the maximum logical input/output unit number (LUN) that the task uses. All integers from one through the declared maximum are then made available to the task. The ASG option declares the devices that are assigned to these LUNs.

The maximum LUN declared cannot be less than the highest unit assigned.

Since the options are processed as they are encountered, to increase the number of LUNs and assign devices to these LUNs you should enter the UNITS option first, then the ASG option. Entering the options in the reverse order can produce an error message.

The device specifying options are as follows:

- ASG (Device Assignment)
- UNITS (Logical Unit Usage)

# ASG (Device Assignment)

The ASG option declares the physical device that is assigned to one or more units.

## SYNTAX

**ASG**= *device-name:unit-num-1:unit-num-2:...:unit-num-n*

where:

- device-name = 2-character alphabetic device name followed by a 1 or 2-digit decimal unit number.
- unit-num-1 = Decimal integers indicating the unit-num-2 logical unit numbers.
- . . .
- unit=num-n

### default

ASG = SY0:1:2:3:4, TI0:5, CL0:6

# UNITS (Logical Unit Usage)

The UNITS option declares the maximum logical unit number used by the task.

## SYNTAX

**UNITS**= *max-units*

where:

- max-units = Decimal integer in the range 0 to 250 which specifies the maximum logical unit number.

### default

UNITS = 6

### 5.4.1 Example of Device Specifying Options

Suppose the FORTRAN programs specified in the file GRP1 require nine logical units. The device assignments for units 1 through 6 agree with the default assumptions and logical units 7,8 and 9 are assigned to DECtape 1 (DT1). The command sequence of the example shown in Section "MAXEXT" is changed to include device assignment options, as follows:

```
PDS> LINK/TASK:IMG1/MAP:MP1/OPTIONS
FILE? GRP1
OPTIONS? UNITS=9
OPTIONS? ASG=DT1:7:8:9
OPTIONS? /
```

**or:**

```
TKB>IMG1,MP1=GRP1
TKB>/
ENTER OPTIONS:
TKB>UNITS=9
TKB>ASG=DT1:7:8:9
TKB>/
```

## 5.5 Storage Altering Options

These options alter the task image and are of interest primarily to the MACRO-11 programmer. The GBLDEF option declares a global symbol and value. The options ABSPAT, GBLPAT and SYMPAT introduce patches into the task image.

The storage altering options are as follows:

- ABSPAT (Absolute Patch)
- GBLDEF (Global Symbol Definition)
- GBLINC (Include Global Symbols)
- GBLPAT (Global Relative Patch)
- GBLREF (Global Symbol Reference)
- GBLXCL (Exclude Global Symbols)
- SYMPAT (Symbolic Patch)

# ABSPAT (Absolute Patch)

The ABSPAT option declares a series of patches starting at the specified base address. Up to eight patch values can be given.

**NOTE: All ABSPAT patches must be within the segment memory limits or a fatal error is generated.**

## SYNTAX

**ABSPAT**= *seg-name:address:val-1:val-2:...:val-8*

where:

- seg-name = 1- to 6-character alphanumeric name of the segment.
- address = Octal address of the first patch. The address can be on a byte boundary, but two bytes are always modified for each patch.
- val-1 = Octal number in the range 0 to 177777 to be assigned to address.
- val-2 = Octal number in the range 0 to 177777 to be assigned to address+2.
- . . . - . . .
- val-8 = Octal number in the range 0 to 177777 to be assigned to address+16(octal).

### default

None

# GBLDEF (Global Symbol Definition)

The GBLDEF option declares the definition of a global symbol.

The symbol definition is considered absolute.

## SYNTAX

**GBLDEF**= *symbol-name:symbol-value*

where:

- symbol-name = 1- to 6-character alphanumeric name of the defined symbol.
- symbol-value = Octal number in the range 0 to 177777 that is assigned to the defined symbol.

### default

None

# GBLINC (Include Global Symbols)

The GBLINC option directs TKB to include the symbol or symbols specified in this option in the .stb file being generated by the link operation where this option appears. This option is intended for use when you create shared global areas, in particular shared libraries, when you want to force particular modules to be linked to your task that reference this library. The global symbol references specified by this option must be satisfied by some module or GBLDEF specification when you build the task.

## SYNTAX

**GBLINC**= *symbol-name,symbol=name,...,symbol-name*

where:

- symbol-name = Symbol to be included.

### default

None

# GBLPAT (Global Relative Patch)

The GBLPAT option declares a series of patch values starting at an offset relative to a global symbol. Up to eight patch values can be given.

## SYNTAX

**GBLPAT**= *seg-name:sym-name[+/-offset]:val-1:val-2:...:val-8*

where:

- sym-name = 1- to 6-character alphanumeric name that specifies the global symbol.
- offset = Octal number used to specify the offset from the global symbol.
- seg-name = Identical to that defined for ABSPAT
- val-1
- val-2
- . . .
- val-8

### default

None

# GBLREF (Global Symbol Reference)

The GBLREF option declares a global symbol reference. The reference originates in the root segment of the task.

## SYNTAX

**GBLREF**= *symbol-name*

where:

- symbol name = 1- to 6-character name of a global symbol reference

### default

None

# GBLXCL (Exclude Global Symbols)

The GBLXCL option keyword directs TKB to exclude from the symbol definition file of a shared global area the symbol(s) specified in the option.

## SYNTAX

**GBLXCL**= *symbol-name,symbol-name...,symbol-name*

where:

- symbol-name = Symbol(s) to be excluded.

### default

None

# SYMPAT (Symbolic Patch)

The SYMPAT option declares a series of symbolic patch values starting at an offset relative to a global symbol. Up to three patch values can be given.

All patches must be within the segment address limits. If they are not, or if a segment does not exist, or if a specified symbol can not be found in the segment, a diagnostic error is generated.

All symbols used in the value specification must already be defined or referenced within the segment being patched. The SYMPAT directive cannot be used to create a new reference from one segment to a symbol defined in another.

## SYNTAX

**SYMPAT** = *seg-name:sym-name[+/-offset]:val-1[:val-2[:val-3]]*

where: [ seg-name | sym-name | offset ] [ are identical to those defined for GBLPAT ]

- val-n = One of the following formats:

  sym-name
  sym-name+offset
  sym-name-offset
  literal
  -literal

  where:

  - sym-name = 1- to 6-character name of a symbol defined or referenced in the segment.
  - offset = Octal number in the range 0 to 177777.
  - literal = Octal number in the range 0 to 177777.

### 5.5.1 Example of Storage Altering Options

Suppose that in the example composed of the MACRO-11 programs TST1 and TST2, GAMMA is a referenced symbol whose value is to be specified when the task is built. The user defines the symbol GAMMA to have the value 25 and introduces 10 numerical patch values at addresses relative to the global symbol DELTA. The user also introduces patch values at addresses relative to the global symbols ALPHA and BETA. Some of these values are themselves in symbolic form.

The terminal sequence of the example shown in Section "SUBLIB" is modified to include the options GBLPAT, GBLDEF and SYMPAT as follows:

```
PDS> LINK/TASK:CHK/MAP:LP0:/OPTIONS
FILE? TST1,TST2
OPTIONS? SGA=DTST:RW:5,STST:RO
OPTIONS? GBLDEF=GAMMA:25
OPTIONS? GBLPAT=TST1:DELTA:1:5:10:15:20:25:30:35
OPTIONS? GBLPAT=TST1:DELTA+20:40:45
OPTIONS? SYMPAT=TST1:ALPHA:PSI:EPSLON-20:30
OPTIONS? SYMPAT=TST1:BETA+20:12737:PSI+1:MU
OPTIONS? /
```

**or:**

```
TKB>CHK,LP:=TST1,TST2
TKB>/
ENTER OPTIONS:
TKB>SGA=DTST:RW:5,STST:RO
TKB>GBLDEF=GAMMA:25
TKB>GBLPAT=TST1:DELTA:1:5:10:15:20:25:30:35
TKB>GBLPAT=TST1:DELTA+20:40:45
TKB>SYMPAT=TST1:ALPHA:PSI:EPSLON-20:30
TKB>SYMPAT=TST1:BETA+20:12737:PSI+1:MU
TKB>/
```

## 5.6 Synchronous Trap Options

Two options are available to declare that the specified vector address is to be preloaded into the task header, thus enabling the task to receive control on the occurrence of synchronous traps. These options are of interest primarily to the MACRO-11 programmer.

The synchronous trap options are as follows:

- ODTV (ODT SST Vector)
- TSKV (Task SST Vector) [list-element]...

# ODTV (ODT SST Vector)

The ODTV option declares a global symbol to be the address of the ODT SST vector. The defined global symbol must exist in the part of the task that is always in memory.

## SYNTAX

**ODTV**= *symbol-name:vector-length*

where:

- symbol-name = 1- to 6-character alphanumeric name of a global symbol.
- vector-length = Decimal integer in the range of 1 to 32 that specifies the length of the SST vector in words.

### default

None

# TSKV (Task SST Vector)

The TSKV option declares a global symbol as the address of the task SST vector. The defined symbol must exist in the part of the task that is always in memory.

## SYNTAX

**TSKV**= *symbol-name:vector-length*

where:

- symbol-name = As defined for ODTV vector-length

## default

None

## 5.7 Example: CALC.TSK;2

Suppose that in the first execution of the task CALC, several logical errors are found. The user corrects the program and is now ready to make the changes in the program and some adjustments in the task image file based on the information obtained about the size of the task in the first task build.

In this example, the user modifies the text file for the program, recompiles the program, and rebuilds the task so that only one active file buffer is reserved.

### 5.7.1 Correcting the Errors in Program Logic

The FORTRAN source language for the program "RDIN.FTN" is corrected as follows:

```
C READ AND ANALYZE INPUT DATA
C SELECT A PROCESSING ROUTINE
C
C ESTABLISH COMMON DATA BASE
C
COMMON /DTA/ A(200), I
C READ IN RAW DATA
READ (6,1) A
1 FORMAT (200 F6.2)
...
CALL PROC1
...
CALL RD1
...
CALL RPRT
END
SUBROUTINE RD1
...
RETURN
END
```

Next, the program "RDIN.FTN" is recompiled as follows:

```
PDS> FORTRAN RDIN
```

**or:**

```
MCR>FOR RDIN,RDIN=RDIN
```

Observe that the corrections to "RDIN.FTN" included the addition of a subroutine "RD1". The object file produced by the FORTRAN compiler as a result of the above terminal sequence now contains two object modules.

### 5.7.2 Building the Task

The user knows from the program logic that only one file is open at a time, but the Task Builder assumes that four files are open simultaneously. Therefore, the user can use the ACTFIL option to reduce the space required for the task.

The task is built with the following terminal sequence:

```
PDS> LINK/TASK:CALC.TSK;2/MAP:(/SHORT)/OPTIONS
FILE? RDIN,PROC1,RPRT
OPTIONS? PAR=GEN
OPTIONS? ACTFIL=1
OPTIONS? /
```

**or:**

```
TKB>CALC;2,LP:=RDIN,PROC1,RPRT
TKB>/
ENTER OPTIONS:
TKB>PAR=GEN
TKB>ACTFIL=1
TKB>/
```

The effect of these options on the memory allocation is seen in Chapter 6, Section 6.5. After the description of the task and memory allocation files, the memory allocation files for the first two examples are given.

# 6 Memory Allocation

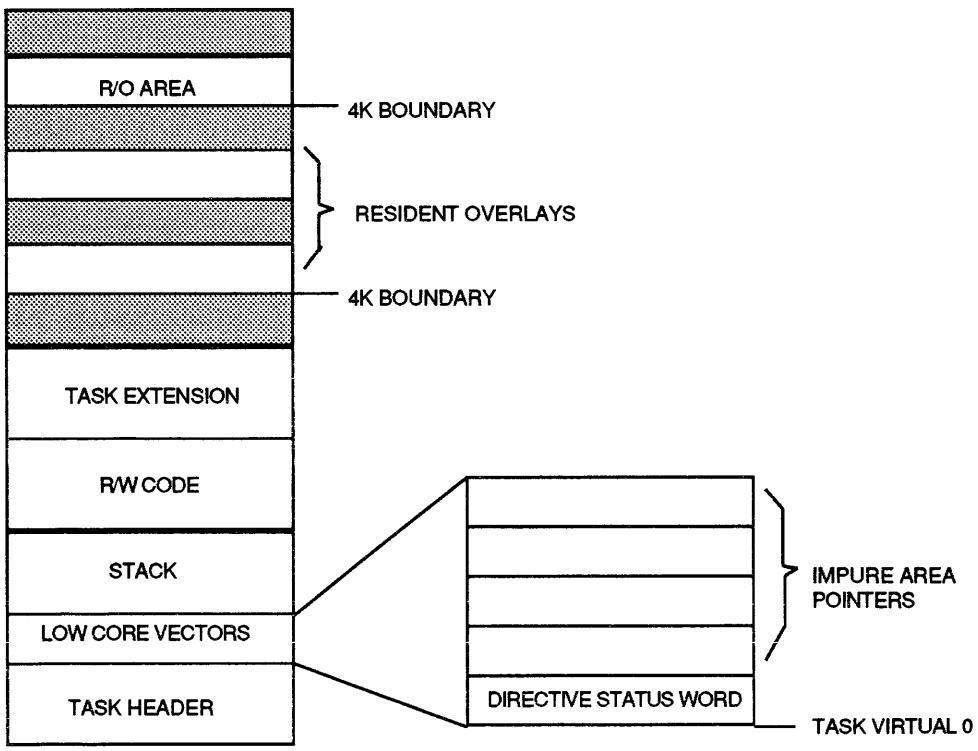
This chapter describes the allocation of task and system memory. The memory allocation file is described in detail and examples of memory allocation files are given. The memory allocation files for the example CALC.TSK;1 of Chapter 2 and CALC.TSK;2 of Chapter 3 are included and analyzed. The effect of the options used in CALC.TSK;2 can be observed by comparing the two memory allocation files.

## 6.1 TASK MEMORY

Task memory in IAS consists of a header, a stack, and a set of areas called program sections (p-sections). Each p-section has attributes from which the Task Builder can determine its base and length.

Task memory layout for a single-segment task is shown in Figure 6–1.

**Figure 6–1 Task Memory Layout**



### 6.1.1 Task Header

The task header contains task parameters and data required by the Executive for controlling execution of a task. It also provides an area for saving information about the task when a switch is made to another task. It is resident at all times when the task is resident, but is not a part of the task's virtual address space. Further details about the task header can be found in Appendix C, Section C.2 and in the *IAS Executive Facilities Reference Manual*.

### 6.1.2 Directive Status Word (DSW)

Virtual location zero of every IAS task is a word reserved for the Executive to report the status of all executive directives issued by the task. This is known as the Directive Status Word (DSW).

### 6.1.3 Impure Area Pointers

The words following the directive status word are used as pointers to the following areas of the task.

| Address | Use |
|---|---|
| 2 | Address of FCS data storage area |
| 4 | Address of FORTRAN-OTS work area |
| 6 | Address of overlay run time system work area |
| 10 | Address of the vector extension area |

Like the Directive Status Word, these parameters occupy the low address end of the task stack.

### 6.1.4 Stack

A default stack of 256(decimal) words is allocated for each task. The STACK= option may be used to override this allocation. A STACK=0 specification is useful in building shareable global areas which do not require a stack.

### 6.1.5 Read/Write Task Code (and Data)

The R/W p-sections of a task are concatenated after the end of the stack. The memory allocation is rounded up to a 32(decimal) word boundary by the addition of dead space.

### 6.1.6 Task Extension

Task extension is the extension to the task requested when the task is built (with the EXTTSK option), or installed or run (with the /INCREMENT or /INC qualifier).

### 6.1.7 Resident Overlays

Each resident overlay segment, whether read/write or read-only, begins on a 4K virtual address boundary. Parallel branches of the overlay tree are allocated the same virtual addresses. The addresses allocated are placed high in the task's address space, in order to leave sufficient room for task extension.

### 6.1.8 Read-Only Task Code (and Data)

If there are p-sections in the task which have the read-only attribute (read/write is the default), the Task Builder concatenates them, and allocates them separately from R/W p-sections. Memory for read-only p-sections is allocated at the highest possible virtual address (starting at a 4K boundary). This allows as much room as possible for the task to be extended.

Note that ODT cannot be used to modify read-only parts of a task. This also means that breakpoints cannot be set in such code. The PDS qualifier /READ_WRITE (MCR switch /RW) can be used when debugging such tasks to inhibit the generation of read-only code.

### 6.1.9 Program Sections (P-sections)

A program section (p-section) is the basic unit of memory allocation for the task. A source language program is translated into an object module consisting of p-sections. For example, the object module produced by compiling a typical FORTRAN program consists of a p-section containing the code generated by the compiler, a p-section for each common block defined in the FORTRAN program, and a set of p-sections required by the FORTRAN Object Time System.

A name and a set of attributes are associated with each p-section. The p-section attributes are given in Figure 6–1.

The scope-code and type-code are only meaningful when an overlay structure is defined for the task. The scope-code is described in connection with the resolution of p-sections in Chapter 7, Section "Resolution of Global Symbols in a Multi-segment Task". The type-code is described in connection with the generation of autoload vectors in Chapter 8, Section 8.1.3. The memory-code is not used by the Task Builder.

The access-code and alloc-code are used by the Task Builder to determine the placement and the size of the p-section in task memory.

The Task Builder divides storage into read/write and read-only memory and places the p-sections in the appropriate area according to access-code.

The alloc-code is used to determine the starting address and length of p-sections with the same name. If the alloc-code indicates that p-sections with the same name are to be overlaid, the Task Builder places each reference at the same position in task memory and determines the total allocation from the length of the longest reference. If the alloc-code indicates that p-sections with the same name are to be concatenated, the Task Builder places each reference one after another in task memory and determines the total allocation from the sum of the lengths of each reference.

When a p-section has the concatenate attribute, all references to that p-section are placed one after another in task memory. If any of these references ends on a byte boundary, the next reference to that p-section is not word-aligned.

Table 6–1 P-section Attributes

| Attribute | Value | Meaning |
|---|---|---|
| access-code | RW* | (read/write). Data can be read from and written into the p-section. |
| | RO | (read-only). Data can be read from, but cannot be written into the p-section. This attribute is overridden if the task is built with the /READ-WRITE PDS qualifier (MCR switch /RW). |
| type-code** | D | (data). The p-section contains data. |
| | I* | (instruction). The p-section contains instructions. |
| scope-code | GBL | (global). The p-section name is considered to cross segment boundaries. The Task Builder allocates storage for the p-section from references outside the defining segment. |
| | LCL* | (local). The p-section name is considered only within the defining segment. The Task Builder allocates storage for the p-section from references within the defining segment only. |
| alloc-code | CON* | (concatenate). P-sections with the same name are concatenated. The total allocation is the sum of the individual allocations. |
| | OVR | (overlay). P-sections with the same name overlay each other. The total allocation is the length of the longest individual allocation. |
| reloc-code | REL* | (relocatable). Storage in the p-section is allocated relative to the start of the p-section. |
| | ABS | (absolute). Storage in the p-section is always allocated relative to the program's virtual zero. |
| memory-code*** | HIGH | (high). The p-section is to be loaded into high speed memory. |
| | LOW* | (low). The p-section is to be loaded into core. |

* —Indicates the default attribute
** —Not to be confused with the I and D space hardware on the PDP 11/44, 11/45, 11/55 and 11/70.
*** —Not implemented

## 6.1.10 Allocation of P-sections

Suppose you enter the following command:

```
PDS> LINK/TASK:IMG1/MAP:MP1
FILE? IN1,IN2,IN3,LBR1/LB
```

**or**

```
MCR>TKB IMG1,MP1=IN1,IN2,IN3,LBR1/LB
```

You are directing the Task Builder to build a task image file, IMG1.TSK, and a memory allocation file, MP1.MAP, from the input files IN1.OBJ, IN2.OBJ, and IN3.OBJ, and to search the library file LBR1.OLB for any undefined global references. Suppose the input files are composed of p-sections with the following access-codes, alloc-codes, and sizes:

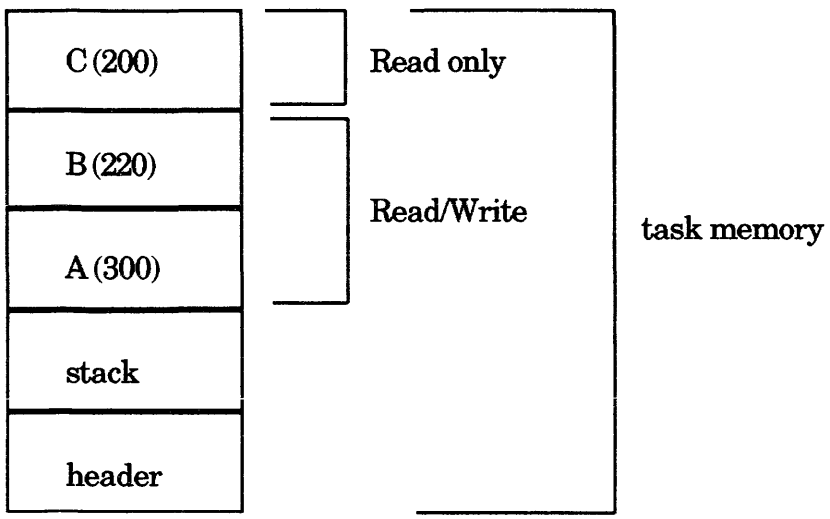
| File-name | P-section Name | Access Code | Alloc Code | Size (octal) |
|---|---|---|---|---|
| IN1 | B | RW | CON | 100 |
| | A | RW | OVR | 300 |
| | C | RO | CON | 150 |
| IN2 | A | RW | OVR | 250 |
| | B | RW | CON | 120 |
| IN3 | C | RO | CON | 50 |

First, the Task Builder collects all p-sections with the same name to determine the allocation for each uniquely named p-section.

In this example, there are two occurrences of the p-section named B with attributes RW and CON. The total allocation for B is the sum of the lengths of each reference; that is, 100 + 120 = 220. There are two occurrences of the p-section named A with attribute OVR; therefore the allocation for A is equal to the larger of the two references, that is, the 300 required for p-section A of file IN1 is used. The allocation for each uniquely named p-section then is:

| P-section Name | Total Allocation |
|---|---|
| B | 220 |
| A | 300 |
| C | 200 |

The Task Builder then re-organizes the p-sections alphabetically and places them in memory according to their access-code, as follows:




### Sequential Allocation of P-sections

The /SEQUENTIAL PDS qualifier (/SQ MCR switch) affects only the placement of p-sections in task memory. P-sections with the same name and attributes are collected as described; then

uniquely named p-sections are placed in memory in the order of input sequence according to the access-code.

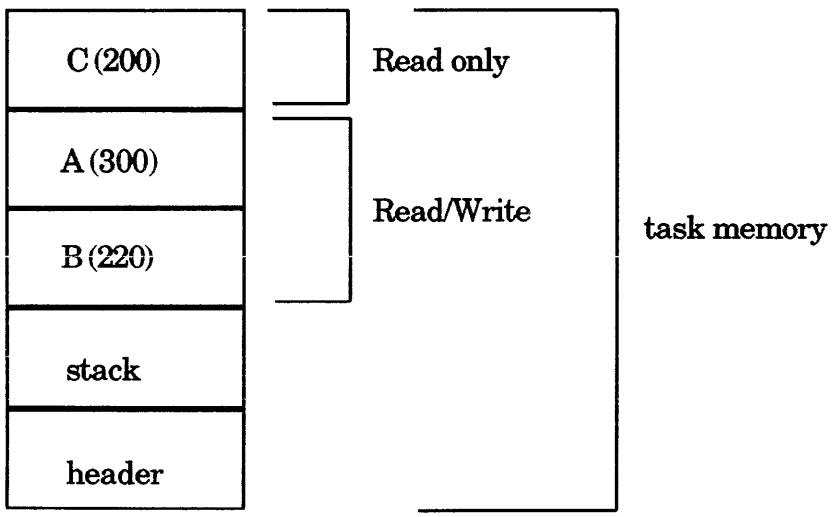
Suppose you add the /SEQUENTIAL PDS qualifier (/SQ MCR switch) to the previous example:

```
PDS> LINK/TASK:IMG1/SEQ/MAP:MP1
FILE? IN1,IN2,IN3,LIBR1/LIB
```

or:

```
MCR>TKB IMG1/SQ,MP1/-SP,MP1=IN1,IN2,IN3,LIBR1/LB
```

The Task Builder collects the p-sections and places them in memory in the input sequence, as follows:




The Task Builder concatenates or overlays the storage requirements of a .PSECT into one allocatable piece of storage. This allocatable piece is currently word aligned by default. This feature was purposely built into the Task Builder so that alignment could be supported at a later date. The user has three alternatives:

1 Allocate all byte aligned data in a separate concatenated .PSECT.

2 Put all data at the front of a program so that the assembler can flag any misalignment.

3 Use .EVEN statements when appropriate.

**Note: The SEQUENTIAL PDS qualifier (/SQ MCR switch) is intended primarily for use with programs written for other systems, such as RT11, that normally allocate tasks in this way. Newly written tasks should not use this facility. If a particular ordering is required, it should be obtained via the alphabetical ordering feature (see Section 6.1.10). Some system components, in particular the FORTRAN OTS, will not operate correctly if they are built into a task linked with the /SEQUENTIAL qualifier. If a task references a position-independent shareable global area, both the task and the SGA must be built with /SEQUENTIAL (or /SQ) specified.**

### 6.1.11 The Resolution of Global Symbols

When creating the task image file, the Task Builder resolves global references. Suppose the global symbols are defined and referenced in the p-sections in the following way:

| File Name | P-section Name | Global Definition | Global Reference |
|---|---|---|---|
| IN1 | B | B1 | A1 |
| | A | B2 | L1 |
| | C | | C1 |
| | | | XXX |
| IN2 | A | A1 | B2 |
| | B | B1 | |
| IN3 | C | | B1 |

In processing the first file, IN1, the Task Builder finds definitions for B1 and B2 and references to A1, L1, C1 and XXX. Since no definition exists for these references, the Task Builder defers the resolution of these global symbols. In processing the next file, IN2, the Task Builder finds a definition for A1, which resolves the previous reference, and a reference to B2, which can be immediately resolved.

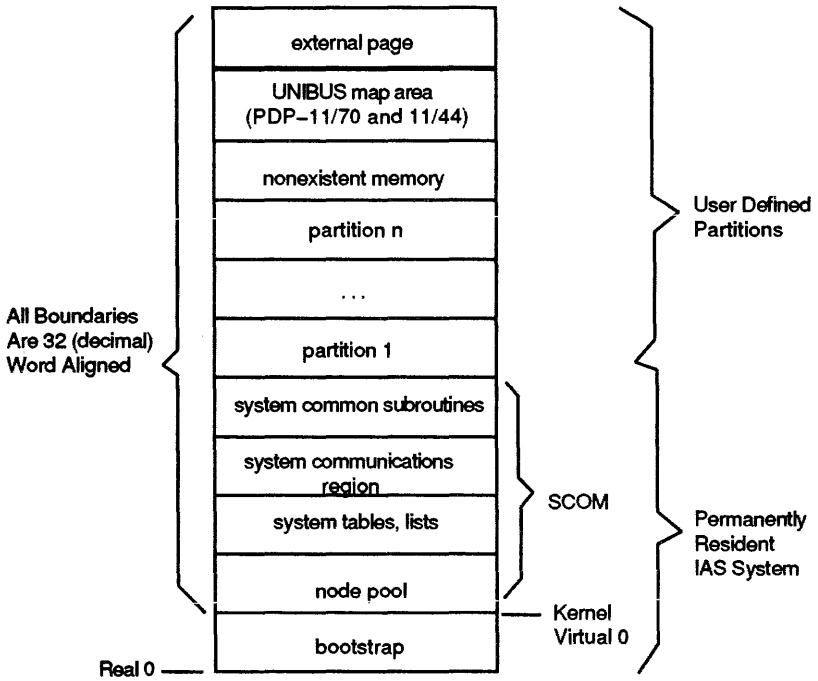
When all the input object files have been processed, the Task Builder has three unresolved global references, namely: C1, L1, and XXX. A search of the library file LBR1 resolves L1 and the Task Builder includes the defining module in the task image. A search of the default library resolves XXX. The default library is LB0:[1,1]SYSLIB.OLB. The global symbol C1 remains unresolved and is, therefore, listed as an undefined global symbol.

The relocatable global symbol B1 is defined twice and is listed as a multiply-defined global symbol on the terminal. The first definition of a multiply-defined symbol is used by the Task Builder. An absolute global symbol can be defined more than once without being listed as multiply defined as long as each occurrence of the symbol has the same value. The results of these resolutions are shown in Figure 6-2.

## 6.2 System Memory

In IAS, system memory consists of the resident executive and a set of named contiguous areas which are defined at system generation time. These named areas are partitions, each of which has parameters of base and length.

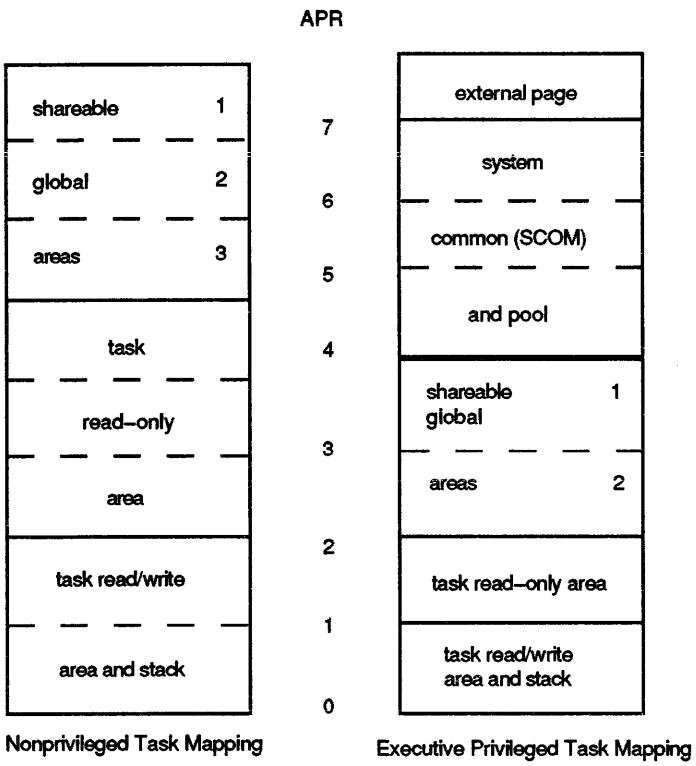
A typical system memory layout can be represented by the following diagram:




## 6.2.1 Executive Privileged Tasks

An executive privileged task has special memory access rights. A task which is not executive privileged can access only its own partition and any referenced shareable global areas, but a privileged task can also access SCOM and the external page.

The following diagram illustrates typical privileged and non-privileged tasks. Note that APR boundaries are aligned at 4K virtual addresses and 32 (decimal) word real addresses when in memory.




## 6.3 Task Image File

In addition to the task memory, or core image, the task image file contains a label block (occupying one, two or three disk blocks). The label block contains data that is used when the task is installed (explicitly or by the RUN timesharing command) to create an entry for the task in the system task directory. The label block and task image structure is described in detail in Appendix C.

## 6.4 Memory Allocation File

The memory allocation file lists information about the allocation of task memory and the resolution of global symbols. A global cross-reference list can be appended to the file by means of the /CROSS_REFERENCE PDS command qualifier (/CR MCR switch).

**Example 6–1 Memory Allocation File for IMG1.TSK;1**

```
IMG1.TSK;1   MEMORY ALLOCATION MAP  TKB D28          PAGE 1
                      3-JUL-78   10:49


IDENTIFICATION : 03
STACK    LIMITS: 000000 000777 001000 00512.
PRG XFR ADDRESS: 001570
TOTAL ATTACHMENT DESCRIPTORS: 3.
TASK  IMAGE  SIZE  : 480. WORDS
TASK HEADER SIZE: 160. WORDS
R-O REGION SIZE: 96. WORDS
TASK ADDRESS LIMITS: 000000 001777
R-W DISK BLK LIMITS: 000003 000005 000003 00003.
R-O DISK BLK LIMITS: 000006 000006 000001 00001.

*** ROOT SEGMENT: IN1


R/W MEM  LIMITS: 000000 001777 002000 01024.
R-O MEM  LIMITS: 140000 140277 000300 00192.
DISK BLK LIMITS: 000003 000004 000002 00002.


MEMORY ALLOCATION SYNOPSIS:

SECTION                                          TITLE  IDENT  FILE
-------                                          -----  -----  ----
. BLK.:(RW,I,LCL,REL,CON) 001000 000020 00016.
                          001000 000020 00016. LIB1          LER1.OLB;1
A     :(RW,I,LCL,REL,OVR) 001020 000250 00168.
                          001020 000100 00064. IN1           IN1.OBJ;1
                          001020 000250 00168. IN2           IN2.OBJ;1
B     :(RW,I,LCL,REL,CON) 001270 000420 00272.
                          001270 000300 00192. IN1           IN1.OBJ;1
                          001570 000120 00080. IN2           IN2.OBJ;1
C     :(RO,I,LCL,REL,CON) 140000 000220 00144.
                          140000 000150 00104. IN1           IN1.OBJ;1
                          140150 000050 00040. IN3           IN3.OBJ;1
$$AUTO:(RW,I,LCL,REL,CON) 160000 000130 00088.
$$LOAD:(RW,I,LCL,REL,CON) 160130 000170 00120.
$$MRKS:(RW,I,LCL,REL,OVR) 160320 000166 00118.
$$OVRS:(RW,I,LCL,ABS,CON) 000000 000000 00000.
$$RDSG:(RW,I,LCL,REL,OVR) 160506 000312 00202.
$$RESL:(RW,I,LCL,REL,CON) 161020 016216 07310.
$$RESM:(RW,I,LCL,REL,CON) 001710 000070 00056.


GLOBAL SYMBOLS:

A1     001020-R  B1     001570-R  B2     001270-R  L1     000022


FILE: IN1.OBJ;1  TITLE: IN1     IDENT:
    <. ABS.>: 000000 000000 000000 00000.
 >>>>>>>>>>>> UNDEFINED REFERENCE: C1
    <A     >: 001020 001117 000100 00064.
```

**Example 6-1 Memory Allocation File for IMG1.TSK;1 (continued)**

```
IMG1.TSK;1    MEMORY ALLOCATION MAP  TKB D28          PAGE 2
IN1                    3-JUL-78   10:49


    <B      >: 001270 001567 000300 00192.
        B1      001570-R  B2      001270-R
    <C      >: 140000 140147 000150 00104.


FILE: IN2.OBJ;1  TITLE: IN2      IDENT:
    <A      >: 001020 001267 000250 00168.
        A1      001020-R
    <B      >: 001570 001707 000120 00080.
        B1      001570-R


FILE: IN3.OBJ;1  TITLE: IN3      IDENT:
    <C      >: 140150 140217 000050 00040.


FILE: LBR1.OLB;1  TITLE: LIB1     IDENT:
    <. ABS.>: 000000 000000 000000 00000.
        L1      000022
    <. BLK.>: 001000 001017 000020 00016.



************

UNDEFINED REFERENCES:

    C1



*** TASK BUILDER STATISTICS:

    TOTAL WORK FILE REFERENCES: 5448.
    WORK  FILE  READS: 0.
    WORK  FILE WRITES: 0.
    SIZE OF CORE POOL: 16010. WORDS (62. PAGES)
    SIZE OF WORK FILE: 1536. WORDS (6. PAGES)

    ELAPSED TIME:00:00:05
```

**Example 6-1 Memory Allocation File for IMG1.TSK;1 (continued)**

```
MP0        CREATED BY  TKB    ON  3-JUL-78 AT 10:49     PAGE 1

GLOBAL CROSS REFERENCE                                  CREF   V02

SYMBOL  VALUE      REFERENCES...

A1      001020-R     IN1     # IN2
B1      001570-R   # IN1     # IN2       IN3
B2      001270-R   # IN1       IN2
C1      000000       IN1
L1      000022       IN1     # LIB1
$MUL    001710       IN1
```

### 6.4.1 Contents of the Memory Allocation File

The memory allocation file consists of the following items:

1 Page Header

2 Task Attributes

3 Overlay Description (if applicable)

4 Segment Description

5 Memory Allocation Synopsis

6 Global Symbols

7 File Contents

8 Summary of Undefined Global Symbols

9 Task Builder Statistics

If the /CROSS_REFERENCE PDS command qualifier (/CR MCR switch) is used to request a global cross-reference, then the following items are also included:

1 Cross-Reference Page Header

2 Global Cross-Reference

3 Segment Cross-Reference

A sample of the memory allocation file produced by the command

```
PDS> LINK/MAP:(MP0/NARROW/FILES)/TASK:IMG1/CROSS_REFERENCE-
IN1, IN2, IN3, LIBRI/LIBRARY
     or
MCR>TKB IMG1,MP0/-SP/-WI/-SH/CR=IN1,IN2,IN3,LIBR/LB
```

is shown in Figure 6-2, where each item is identified. The overlay description does not apply to this task, and is therefore not shown.

These items are described in the following paragraphs.

1 The page header shows the name of the task image file and the overlay segment name, along with the date, time, and version of the Task Builder that was used.

2 The task attribute section contains the following information. Each item is printed only if a non-default value has been specified.

- Task name
- Task partition
- Identification (task version)
- Task UIC
- Task priority
- Stack limits—consisting of the low and high addresses, followed by the length in octal and decimal bytes
- ODT transfer address—starting address of the debugging aid

- Program transfer address
- Task attributes—shown only if they differ from the defaults. One or more of the following may be displayed:

| | |
|---|---|
| -AB | Task cannot be aborted |
| -CP | Task is not checkpointable |
| DA | Task contains debugging aid |
| -DS | Task cannot be disabled |
| FP | Task uses floating-point processor |
| -FR | Task will not have its receive queues flushed |
| -FX | Task cannot be fixed |
| PI | Task contains position-independent code and data |
| PR | Task is executive privileged |
| -SE | Task cannot have data sent to it |
| TR | Task initial PS has T-bit set |

- Number of ADBS—number of attachment descriptors allocated in the task header, including those requested by the ATRG option and those allocated automatically by the Task Builder
- Mapped array area—amount of space allocated for VSECTs
- Task extension—the increment of physical memory allocated through the EXTTSK keyword
- Task image—the amount of memory required to contain task code, including the header
- Task header size (in words)
- Size of read/write resident overlay region, including VSECTs
- Size of read-only region, including task pure area and read-only resident overlays
- Task address limits—the lowest and highest virtual addresses allocated to the task
- Read/write disk block limits, for all segments
- Read-only disk block limits

3 The overlay description shows the address limits, length, and name of each overlay segment. Indenting is used to illustrate the overlay structure. The overlay description is printed only when a multi-segment task is created. An example of overlay description output is shown in Figure 6-2.

4 The segment description gives the name of the segment together with the segment address and disk space limits. A read-only resident segment does not have its disk block limits displayed. See Appendix C, Section C.1.1.

5 The memory allocation synopsis gives information about the p-sections that make up the memory allocated to each overlay segment. The information shown consists of the p-section name, attributes, starting address, and length in bytes (octal and decimal values), followed by a list of modules that contributed storage to the section. The entry for each module shows the starting address and length of the allocation, the module name, module identification, and file name.

6 If the /SEQUENTIAL PDS command qualfier (/SQ MCR switch) is applied, the p-sections are listed in the order of input; otherwise they appear in alphabetical order.

7 The following p-section information is omitted:

   a. The absolute section, . ABS. is not shown because it appears in every module and always has a length of 0.

   b. The unnamed relocatable section, shown as . BLK., is not displayed if its length is 0, because it appears in every module.

8 Global symbols that are defined in the segment are listed along with their octal values. A "-R" is appended to the value if the symbol is relocatable. The list is alphabetized in columns.

9 The file contents section lists the module name, file name, p-sections, and global definitions occurring in the module. Any undefined global references made by the module are also displayed. This section only appears if the /FILES or /FU map file specification qualifier was used.

10 A summary of undefined global references is printed after the listing of file contents.

11 The display of Task Builder statistics lists the following information, which may be used to evaluate Task Builder performance.

   a. Work File References—The number of times that the Task Builder accessed data stored in its work file.

   b. Work File Reads—The number of times that the work file device was accessed to read work file data.

   c. Work File Writes—The number of times that the work file device was accessed to write work file data.

   d. Size of Core Pool—The amount of memory that was available for work file data and table storage.

   e. Size of Work File—The amount of device storage that was required to contain the work file.

   f. Elapsed Time—The amount of wall-clock time required to construct the task image and produce the memory allocation file. Elapsed time is measured from the completion of option input to the completion of map output. This value excludes the time required to process the overlay description, parse the list of input file names, and create the cross-reference listing (if specified).

   Appendix F should be consulted for a more detailed discussion of the work file.

12 The cross-reference page header gives the name of the memory allocation file, the originating task (TKB), the date and time the memory allocation file was created, and the cross-reference page number, in the following format:

```
map file name  CREATED BY TKB ON date AT time PAGE n
GLOBAL CROSS REFERENCE            CREF Vn
SYMBOL          VALUE           REFERENCES...
```

13 The cross-reference list contains an alphabetic listing of each global symbol along with its value and the name of each referencing module. When a symbol is defined in several segments within an overlay structure, the last defined value is printed.

14 The suffix -R is appended to the value if the symbol is relocatable.

15 Prefix symbols accompanying each module name define the type of reference as follows:

| Prefix Symbol | Reference Type |
|---|---|
| blank | Module contains a reference that is resolved in the same segment or in a segment toward the root. |
| ^ | Module contains a reference that is resolved directly in a segment away from the root or in a co-tree. |
| @ | Module contains a reference that is resolved through an autoload vector. |
| # | Module contains a non-autoloadable definition. |
| * | Module contains an autoloadable definition. |

16 The segment cross-reference lists the name of each non-empty overlay segment and the modules that compose it.

## 6.4.2 Control of Memory Allocation File Contents and Format

By using the memory allocation and input file switches or qualifiers described below, you can eliminate non-essential information from the output, improve Task Builder throughput, and obtain output in a format that is more compatible with the hard copy device.

The amount of information presented in the memory allocation file is controlled by the /FILES and /FULL PDS qualifiers (/-SH and /MA MCR switches respectively). When the /FILES PDS qualifier (/-SH MCR switch) is included in the map file specification, the Task Builder includes the file contents section of the allocation listing. By default, this information is omitted as the most useful parts can be found from the memory allocation synopsis.

In general, the short format provides sufficient information for debugging, while reducing task-build time considerably. Listings that contain a full description of file contents can be obtained at less frequent intervals and kept for later reference. The contents of individual input files can be excluded from the listing by the /NOMAP PDS input file qualifier (/-MA MCR switch). Suppressing such output eliminates the following information from the allocation and cross-reference output for the excluded file:

1 P-section contributions as shown in the memory allocation synopsis.

2 Global symbol definitions.

3 File contents.

4 Global definitions or references, and module names as shown in the cross-reference listing.

To disable map output for individual files, you include the /NOMAP PDS qualifier (or MCR switch /-MA) in the input file specification. To include such output for the default system object module library and all memory-resident library files, you include PDS qualifier /FULL (or MCR switch /MA) in the memory allocation file specification.

The width of the listing is controlled by the /NARROW and /WIDE PDS qualifiers (/-WI and /WI MCR switches respectively). /NARROW (/-WI) indicates that the listing format can occupy 72 columns, suitable for output to a terminal. /WIDE (/WI) indicates that 132 columns can be used.

## 6.5 Examples: CALC;1 and CALC;2 Maps

The first run of CALC, described in Chapter 2, Section 2.3 produces the memory allocation file shown in Figure 6-3. This is the default memory allocation output, including all the parts described in this chapter except the file contents section.

The task attributes section lists the principal characteristics of interest, such as task size in words, and task address limits. Items such as task name and task attributes, that are not specified, or that do not differ from the default, have been omitted.

The segment description lists the memory and disk block limits for the root segment.

The Memory Allocation Synopsis displays the storage allocated to each p-section. The first line adjacent to the name, shows the total allocation. Subsequent lines show the contribution made by individual modules. The values displayed are: base address, and length in bytes (octal and decimal values).

Because the /FULL qualifier was not in force, all information about modules from the system library has been omitted.

The second example (see Figure 6-4) shows a map of CALC;1 with the /FULL and /FILES qualifiers.

In the example CALC.TSK;2 in Chapter 5, Section 5.7, the user added some code to RDIN.FIN, and entered two options during option input:

1 ACTFIL=1 - to eliminate the three active file buffers not needed by CALC.TSK,

2 PAR=GEN - to direct the Task Builder to use a larger partition for CALC.TSK. However, this has no affect on task building other than to set up a partition in which the task is to execute.

The memory allocation file shown in Figure 6-5 reflects these changes.

Because the ACTFIL keyword was used, the File Storage Region Buffer pool decreased from 4100 in CALC.TSK;1 to 1020 in CALC.TSK;2.

The use of the ACTFIL keyword saved 3060 bytes.

The remainder of this chapter contains Figure 6-3, Figure 6-4, and Figure 6-5 showing the memory allocation files described above.

**Example 6–2 Memory Allocation File for CALK.TSK;1 (Default Output Format)**

```
CALC.TSK;1   MEMORY ALLOCATION MAP  TKB D28          PAGE 1
                    3-JUL-78   10:49


IDENTIFICATION : FORV02
STACK    LIMITS: 000000 000777 001000 00512.
PRG XFR ADDRESS: 020246
TOTAL ATTACHMENT DESCRIPTORS: 3.
TASK  IMAGE  SIZE  : 6976. WORDS
TASK HEADER SIZE: 160. WORDS
TASK ADDRESS LIMITS: 000000 033247
R-W DISK BLK LIMITS: 000003 000035 000033 00027.

*** ROOT SEGMENT: RDIN


R/W MEM  LIMITS: 000000 033247 033250 13992.
DISK BLK LIMITS: 000003 000036 000034 00028.


MEMORY ALLOCATION SYNOPSIS:

SECTION                                          TITLE  IDENT  FILE
-------                                          -----  -----  ----
. BLK.:(RW,I,LCL,REL,CON) 001000 000002 00002.
DTA    :(RW,D,GBL,REL,OVR) 001002 001442 00802.
                           001002 001442 00802. .MAIN. FORV02 RDIN.OBJ;1
                           001002 001442 00802. PROC1  FORV02 PROC1.OBJ;1
                           001002 001442 00802. RPRT   FORV02 RPRT.OBJ;1
OTS$I  :(RW,I,LCL,REL,CON) 002444 015544 07012.
                           002444 000000 00000. .MAIN. FORV02 RDIN.OBJ;1
OTS$P  :(RW,D,GBL,REL,OVR) 020210 000036 00030.
$CODE  :(RW,I,LCL,REL,CON) 020246 000162 00114.
                           020246 000000 00000. .MAIN. FORV02 RDIN.OBJ;1
                           020246 000000 00000. .MAIN. FORV02 RDIN.OBJ;1
                           020246 000072 00058. .MAIN. FORV02 RDIN.OBJ;1
                           020340 000000 00000. PROC1  FORV02 PROC1.OBJ;1
                           020340 000000 00000. PROC1  FORV02 PROC1.OBJ;1
                           020340 000054 00044. PROC1  FORV02 PROC1.OBJ;1
                           020414 000000 00000. RPRT   FORV02 RPRT.OBJ;1
                           020414 000000 00000. RPRT   FORV02 RPRT.OBJ;1
                           020414 000014 00012. RPRT   FORV02 RPRT.OBJ;1
$DATA  :(RW,D,LCL,REL,CON) 020430 003722 02002.
                           020430 000000 00000. .MAIN. FORV02 RDIN.OBJ;1
                           020430 001750 01000. .MAIN. FORV02 RDIN.OBJ;1
                           022400 000000 00000. PROC1  FORV02 PROC1.OBJ;1
                           022400 000002 00002. PROC1  FORV02 PROC1.OBJ;1
                           022402 000000 00000. RPRT   FORV02 RPRT.OBJ;1
                           022402 001750 01000. RPRT   FORV02 RPRT.OBJ;1
$DATAP:(RW,D,LCL,REL,CON) 024352 000042 00034.
                           024352 000000 00000. .MAIN. FORV02 RDIN.OBJ;1
                           024352 000022 00018. .MAIN. FORV02 RDIN.OBJ;1
                           024374 000000 00000. PROC1  FORV02 PROC1.OBJ;1
                           024374 000010 00008. PROC1  FORV02 PROC1.OBJ;1
                           024404 000000 00000. RPRT   FORV02 RPRT.OBJ;1
                           024404 000010 00008. RPRT   FORV02 RPRT.OBJ;1
$$AOTS:(RW,D,LCL,REL,CON) 024414 000704 00452.
$$AUTO:(RW,I,LCL,REL,CON) 160000 000130 00088.
```

**Example 6-2 Memory Allocation File for CALC.TSK;1 (Default Output Format) (continued)**

```
CALC.TSK;1    MEMORY ALLOCATION MAP  TKB D28          PAGE 2
RDIN                    3-JUL-78   10:49


$$DEVT:(RW,D,LCL,REL,OVR) 025320 001210 00648.
$$FSR1:(RW,D,GBL,REL,OVR) 026530 004100 02112.
$$FSR2:(RW,D,GBL,REL,CON) 032630 000104 00068.
$$IOB1:(RW,D,LCL,REL,OVR) 032734 000204 00132.
$$IOB2:(RW,D,LCL,REL,OVR) 033140 000000 00000.
$$LOAD:(RW,I,LCL,REL,CON) 160130 000170 00120.
$$MRKS:(RW,I,LCL,REL,OVR) 160320 000166 00118.
$$OBF1:(RW,D,LCL,REL,CON) 033140 000110 00072.
$$OBF2:(RW,I,LCL,REL,CON) 033250 000000 00000.
$$OVRS:(RW,I,LCL,ABS,CON) 000000 000000 00000.
$$RDSG:(RW,I,LCL,REL,OVR) 160506 000312 00202.
$$RESL:(RW,I,LCL,REL,CON) 161020 016216 07310.
.$$$$.:(RW,D,GBL,REL,OVR) 033250 000000 00000.
                          033250 000000 00000. .MAIN. FORV02 RDIN.OBJ;1
                          033250 000000 00000. .MAIN. FORV02 RDIN.OBJ;1
                          033250 000000 00000. PROC1  FORV02 PROC1.OBJ;1
                          033250 000000 00000. PROC1  FORV02 PROC1.OBJ;1
                          033250 000000 00000. RPRT   FORV02 RPRT.OBJ;1
                          033250 000000 00000. RPRT   FORV02 RPRT.OBJ;1


GLOBAL SYMBOLS:

PROC1  020340-R  $RF2A1 000000-R  $$OTSI 002444-R
RPRT   020414-R  $$OTSC 020246-R


*** TASK BUILDER STATISTICS:

    TOTAL WORK FILE REFERENCES: 14413.
    WORK  FILE  READS: 0.
    WORK  FILE WRITES: 0.
    SIZE OF CORE POOL: 16010. WORDS (62. PAGES)
    SIZE OF WORK FILE: 3072. WORDS (12. PAGES)

    ELAPSED TIME:00:00:13
```

**Example 6–3 Memory Allocation File for CALC.TSK;1 (Part Printout/FULL/FILES)**

```
CALC.TSK;1    MEMORY ALLOCATION MAP  TKB D28          PAGE 1
                     3-JUL-78   10:49


IDENTIFICATION : FORV02
STACK    LIMITS: 000000 000777 001000 00512.
PRG XFR ADDRESS: 020246
TOTAL ATTACHMENT DESCRIPTORS: 3.
TASK  IMAGE  SIZE  : 6976. WORDS
TASK HEADER SIZE: 160. WORDS
TASK ADDRESS LIMITS: 000000 033247
R-W DISK BLK LIMITS: 000003 000035 000033 00027.

*** ROOT SEGMENT: RDIN


R/W MEM  LIMITS: 000000 033247 033250 13992.
DISK BLK LIMITS: 000003 000036 000034 00028.


MEMORY ALLOCATION SYNOPSIS:

SECTION                                          TITLE  IDENT  FILE
-------                                          -----  -----  ----
. BLK.:(RW,I,LCL,REL,CON)  001000 000002 00002.
                           001000 000002 00002.  $NVINI        SYSLIB.OLB;1
DTA    :(RW,D,GBL,REL,OVR)  001002 001442 00802.
                           001002 001442 00802.  .MAIN. FORV02 RDIN.OBJ;1
                           001002 001442 00802.  PROC1  FORV02 PROC1.OBJ;1
                           001002 001442 00802.  RPRT   FORV02 RPRT.OBJ;1
OTS$I  :(RW,I,LCL,REL,CON)  002444 015544 07012.
                           002444 000000 00000.  .MAIN. FORV02 RDIN.OBJ;1
                           002444 000134 00092.  $ADDM  F40002 SYSLIB.OLB;1
                           002600 000036 00030.  $CALL  F40002 SYSLIB.OLB;1
                           002636 000132 00090.  $EOL   F40002 SYSLIB.OLB;1
                           002770 001756 01006.  $CONVF F40002 SYSLIB.OLB;1
                           004746 000102 00066.  $IFR   F40002 SYSLIB.OLB;1
                           005050 000014 00012.  $ISNLS F40001 SYSLIB.OLB;1
                           005064 000102 00066.  $IMOVS F40002 SYSLIB.OLB;1
                           005166 000054 00044.  $ERAS  F40002 SYSLIB.OLB;1
                           005242 000050 00040.  $RETS  F40002 SYSLIB.OLB;1
                           005312 000044 00036.  $FVEC  F40002 SYSLIB.OLB;1
                           005356 000260 00176.  $TRARY F40002 SYSLIB.OLB;1
                           005636 000252 00170.  $OTI   F40002 SYSLIB.OLB;1
                           006110 000122 00082.  $SUER  F40002 SYSLIB.OLB;1
                           006232 000000 00000.  $OTV   F40001 SYSLIB.OLB;1
                           006232 000406 00262.  $CONVI F40002 SYSLIB.OLB;1
                           006640 000160 00112.  $SAVRE F40002 SYSLIB.OLB;1
                           007020 000036 00030.  $FCHNL F40001 SYSLIB.OLB;1
                           007056 001674 00956.  $FIO   F40002 SYSLIB.OLB;1
                           010752 000416 00270.  $OPEN  F40002 SYSLIB.OLB;1
                           011370 000076 00062.  $GETRE F40001 SYSLIB.OLB;1
                           011466 000066 00054.  $INITI F40002 SYSLIB.OLB;1
                           011554 000244 00164.  $STPPA F40001 SYSLIB.OLB;1
                           012020 001700 00960.  $ERRPT F40001 SYSLIB.OLB;1
                           013720 000204 00132.  $FPERR F40002 SYSLIB.OLB;1
                           014124 000274 00188.  $ERQIO F40001 SYSLIB.OLB;1
                           014420 000070 00056.  $CLOSE F40001 SYSLIB.OLB;1
```

**Example 6-3 Memory Allocation File for CALC.TSK;1 (Part Printout/FULL/FILES) (continued)**

```
CALC.TSK;1    MEMORY ALLOCATION MAP  TKB D28          PAGE 2
RDIN                    3-JUL-78   10:49

                              014510 003364 01780. $ERTXT F40004 SYSLIB.OLB;1
                              020074 000114 00076. $R50   F40001 SYSLIB.OLB;1
OTS$P :(RW,D,GBL,REL,OVR)     020210 000036 00030.
                              020210 000036 00030. $CONVF F40002 SYSLIB.OLB;1
                              020210 000036 00030. $CONVI F40002 SYSLIB.OLB;1
                              020210 000036 00030. $FIO   F40002 SYSLIB.OLB;1
$CODE :(RW,I,LCL,REL,CON)     020246 000162 00114.
                              020246 000000 00000. .MAIN. FORV02 RDIN.OBJ;1
                              020246 000000 00000. .MAIN. FORV02 RDIN.OBJ;1
                              020246 000072 00058. .MAIN. FORV02 RDIN.OBJ;1
                              020340 000000 00000. PROC1  FORV02 PROC1.OBJ;1
                              020340 000000 00000. PROC1  FORV02 PROC1.OBJ;1
                              020340 000054 00044. PROC1  FORV02 PROC1.OBJ;1
                              020414 000000 00000. RPRT   FORV02 RPRT.OBJ;1
                              020414 000000 00000. RPRT   FORV02 RPRT.OBJ;1
                              020414 000014 00012. RPRT   FORV02 RPRT.OBJ;1
$DATA :(RW,D,LCL,REL,CON)     020430 003722 02002.
                              020430 000000 00000. .MAIN. FORV02 RDIN.OBJ;1
                              020430 001750 01000. .MAIN. FORV02 RDIN.OBJ;1
                              022400 000000 00000. PROC1  FORV02 PROC1.OBJ;1
                              022400 000002 00002. PROC1  FORV02 PROC1.OBJ;1
                              022402 000000 00000. RPRT   FORV02 RPRT.OBJ;1
                              022402 001750 01000. RPRT   FORV02 RPRT.OBJ;1
$DATAP:(RW,D,LCL,REL,CON)     024352 000042 00034.
                              024352 000000 00000. .MAIN. FORV02 RDIN.OBJ;1
                              024352 000022 00018. .MAIN. FORV02 RDIN.OBJ;1
                              024374 000000 00000. PROC1  FORV02 PROC1.OBJ;1
                              024374 000010 00008. PROC1  FORV02 PROC1.OBJ;1
                              024404 000000 00000. RPRT   FORV02 RPRT.OBJ;1
                              024404 000010 00008. RPRT   FORV02 RPRT.OBJ;1
$$AOTS:(RW,D,LCL,REL,CON)     024414 000704 00452.
                              024414 000704 00452. $OTV   F40001 SYSLIB.OLB;1
$$AUTO:(RW,I,LCL,REL,CON)     160000 000130 00088.
                              160000 000130 00088. SYSRES 12     SYSRES.STB;1
$$DEVT:(RW,D,LCL,REL,OVR)     025320 001210 00648.
                              025320 000000 00000. $OTV   F40001 SYSLIB.OLB;1
$$FSR1:(RW,D,GBL,REL,OVR)     026530 004100 02112.
                              026530 000000 00000. $OTV   F40001 SYSLIB.OLB;1
                              026530 000000 00000. FCSFSR 0303MS SYSLIB.OLB;1
$$FSR2:(RW,D,GBL,REL,CON)     032630 000104 00068.
                              032630 000104 00068. FCSFSR 0303MS SYSLIB.OLB;1
$$IOB1:(RW,D,LCL,REL,OVR)     032734 000204 00132.
                              032734 000204 00132. $OTV   F40001 SYSLIB.OLB;1
$$IOB2:(RW,D,LCL,REL,OVR)     033140 000000 00000.
                              033140 000000 00000. $OTV   F40001 SYSLIB.OLB;1
$$LOAD:(RW,I,LCL,REL,CON)     160130 000170 00120.
                              160130 000170 00120. SYSRES 12     SYSRES.STB;1
$$MRKS:(RW,I,LCL,REL,OVR)     160320 000166 00118.
                              160320 000166 00118. SYSRES 12     SYSRES.STB;1
$$OBF1:(RW,D,LCL,REL,CON)     033140 000110 00072.
                              033140 000110 00072. $OTV   F40001 SYSLIB.OLB;1
$$OBF2:(RW,I,LCL,REL,CON)     033250 000000 00000.
                              033250 000000 00000. $OTV   F40001 SYSLIB.OLB;1
$$OVRS:(RW,I,LCL,ABS,CON)     000000 000000 00000.
                              000000 000000 00000. SYSRES 12     SYSRES.STB;1
```

**Example 6-3 Memory Allocation File for CALC.TSK;1 (Part Printout/FULL/FILES) (continued)**

```
CALC.TSK;1    MEMORY ALLOCATION MAP  TKB D28          PAGE 3
RDIN                  3-JUL-78   10:49


$$RDSG:(RW,I,LCL,REL,OVR) 160506 000312 00202.
                          160506 000312 00202. SYSRES 12     SYSRES.STB;1
$$RESL:(RW,I,LCL,REL,CON) 161020 016216 07310.
                          161020 016216 07310. SYSRES 12     SYSRES.STB;1
.$$$$.:(RW,D,GBL,REL,OVR) 033250 000000 00000.
                          033250 000000 00000. .MAIN. FORV02 RDIN.OBJ;1
                          033250 000000 00000. .MAIN. FORV02 RDIN.OBJ;1
                          033250 000000 00000. PROC1  FORV02 PROC1.OBJ;1
                          033250 000000 00000. PROC1  FORV02 PROC1.OBJ;1
                          033250 000000 00000. RPRT   FORV02 RPRT.OBJ;1
                          033250 000000 00000. RPPT   FORV02 RPRT.OBJ;1

GLOBAL SYMBOLS:

ADF$IM 002444-R  MOI$0S 005130-R  SVF$SM 005326-R  $NAMC  024466-R
ADF$MM 002476-R  MOI$1A 005160-R  TAD$   005406-R  $OPEN  010752-R
ADF$PM 002462-R  MOI$1M 005152-R  TAF$   005414-R  $OTI   005636-R
ADF$SM 002510-R  MOI$1S 005144-R  TAI$   005356-R  $OTIS  006110-R
BAH$   011650-R  MOL$IS 005100-R  TAL$   005364-R  $OTSV  000004
BEQ$   005212-R  MOL$SS 005064-R  TAP$   005400-R  $OTSVA 024464-R
BGE$   005222-R  NMI$1I 005200-R  TAQ$   005372-R  $PSE   011626-R
BGT$   005220-R  NMI$1M 005166-R  THRD$  007016-R  $PSES  011660-R
BLE$   005210-R  N.ALER 000010    V007A  000000    $RF2A1 000000-R
BLT$   005232-R  N.IOST 000004    $ALBP1 160016-R  $RLCB  176544-R
BNE$   005230-R  N.MRKS 000016    $ALBP2 160114-R  $RQCB  176646-R
BRA$   005224-R  N.OVLY 000000    $ATT   014124-R  $R50   020074-R
CAI$   002600-R  N.OVPT 000006    $AUTO  160000-R  $SAVRG 177004-R
CAL$   002606-R  N.RDSG 000014    $BINAS 013212-R  $SEQC  024464-R
DCO$   004024-R  N.STBL 000002    $CLOSE 014420-R  $SST   025300-R
ECO$   004016-R  N.SZSG 000012    $DET   014230-R  $SST0  012020-R
EOL$   002666-R  OCI$   006232-R  $ECI   006254-R  $SST1  012026-R
EXIT$  011732-R  OCO$   006434-R  $EOL   002664-R  $SST2  012040-R
FCO$   004012-R  O$VEF  000037    $ERRAA 012206-R  $SST3  012046-R
FOO$   011722-R  PROC1  020340-R  $ERRTB 013530-R  $SST4  012054-R
F.BFHD 000020    PSE$   011622-R  $ERRTE 013720-R  $SST5  012062-R
F.FDB  000154    RCI$   002770-R  $ERRWT 014314-R  $SST6  012146-R
GCO$   004004-R  REL$   005100-R  $ERRZA 013150-R  $SST7  012072-R
ICI$   006240-R  RET$   005256-R  $ERTXT 014510-R  $STP   011716-R
ICO$   006442-R  RET$F  005246-R  $ERXIT 012436-R  $STPS  011710-R
IFR$   004746-R  RET$I  005254-R  $EXIT  011732-R  $SVTK$ 013522-R
ISN$   005050-R  RET$L  005242-R  $EXIT$ 012124-R  $TAD   005406-R
LSN$   005056-R  RPRT   020414-R  $FCHNL 007020-R  $TAF   005414-R
MOI$IA 005110-R  SAF$IM 005312-R  $FILL  013266-R  $TAI   005356-R
MOI$IM 005104-R  SAF$MM 005346-R  $FIO   007614-R  $TAL   005364-R
MOI$IS 005100-R  SAF$SM 005314-R  $FLDEF 011276-R  $TAP   005400-R
MOI$MA 005124-R  SAVRG$ 006640-R  $FPERR 013720-R  $TAQ   005372-R
MOI$MM 005120-R  STP$   011716-R  $GETRE 011370-R  $VIRIN 001000-R
MOI$MS 005114-R  SUF$IM 002522-R  $IFR   004752-R  $$FIO  007620-R
MOI$SA 005074-R  SUF$MM 002554-R  $INITI 011466-R  $$IFR  004756-R
MOI$SM 005070-R  SUF$PM 002540-R  $IOEXI 002636-R  $$OTI  005640-R
MOI$SS 005064-R  SUF$SM 002566-R  $LOAD  160130-R  $$OTIS 006112-R
MOI$0A 005140-R  SVF$IM 005324-R  $MARKR 160320-R  $$OTSC 020246-R
MOI$0M 005134-R  SVF$MM 005352-R  $MARKS 160320-R  $$OTSI 002444-R
```

**Example 6-3 Memory Allocation File for CALC.TSK;1 (Part Printout/FULL/FILES) (continued)**

```
CALC.TSK;1    MEMORY ALLOCATION MAP  TKB D28          PAGE 4
RDIN                    3-JUL-78   10:49


.ASCPP 176426-R  .PPR50 176054-R  ..ENTR 172010-R  ..RBLK 173030-R
.ASLUN 170754-R  .PRSDI 174666-R  ..EXTD 172602-R  ..RDRN 167742-R
.CLOSE 161020-R  .PRSDV 175112-R  ..EXT1 172656-R  ..RFDB 167012-R
.CTRL  174340-R  .PRSFN 175306-R  ..FCSX 166520-R  ..RMOV 172016-R
.DCCVT 177034-R  .PUT   165066-R  ..FIND 172024-R  ..RTAD 172242-R
.FATAL 166532-R  .PUTSQ 165066-R  ..FINI 172130-R  ..RWAC 170016-R
.FINIT 172120-R  .SAVR1 166410-R  ..GTDI 170570-R  ..RWAT 167774-R
.FSRCB 032630-R  ..ALC1 172552-R  ..IDPB 166632-R  ..SEFB 167436-R
.FSRPT 000002    ..ALOC 172502-R  ..MKDL 172224-R  ..SGR5 174534-R
.GET   161666-R  ..ALUN 170760-R  ..MVR1 167534-R  ..STFN 172376-R
.GETSQ 161666-R  ..ANSP 166444-R  ..PARS 172450-R  ..WAEF 166742-R
.GTDID 162640-R  ..BDRC 171422-R  ..PDI  174110-R  ..WAIT 166722-R
.MBFCT 032730-R  ..BKRG 166500-R  ..PDID 170742-R  ..WAND 173006-R
.MOLUN 024472-R  ..CREA 171504-R  ..PGCR 167576-R  ..WAST 174016-R
.NLUNS 024470-R  ..CTRL 174344-R  ..PNT1 173136-R  ..WBLK 173036-R
.ODCVT 177030-R  ..DEL1 171750-R  ..PSDI 174700-R  ..WTWA 167146-R
.OPEN  162656-R  ..DID  174376-R  ..PSDV 175124-R  ..WTWD 167146-R
.OPFNB 162666-R  ..DIDF 174054-R  ..PSFN 175320-R  ..WTW1 167152-R
.PARSE 172436-R  ..DID1 174372-R  ..PSIT 173416-R  ..XQIO 166534-R
.POSIT 173372-R  ..DIRF 176314-R  ..PSRC 167726-R  ..XQI1 166552-R
.POSRC 167670-R  ..EFCK 167360-R  ..PSR1 167712-R
.PPASC 176140-R  ..EFC1 167366-R  ..QIOW 166672-R

FILE: SYSRES.STB;1  TITLE: SYSRES  IDENT: 12
    <$$AUTO>: 160000 160127 000130 00088.
        $ALBP1 160016-R  $ALBP2 160114-R  $AUTO  160000-R
    <$$LOAD>: 160130 160317 000170 00120.
        $LOAD  160130-R
    <$$MRKS>: 160320 160505 000166 00118.
        $MARKR 160320-R  $MARKS 160320-R
    <$$OVRS>: 000000 000000 000000 00000.
        N.ALER 000010    N.IOST 000004    N.MRKS 000016    N.OVLY 000000
        N.RDSG 000014    N.STBL 000002    N.SZSG 000012
    <$$RDSG>: 160506 161017 000312 00202.
    <$$RESL>: 161020 177235 016216 07310.
        $RLCB  176544-R  $RQCB  176646-R  $SAVRG 177004-R  .ASCPP 176426-R
        .ASLUN 170754-R  .CLOSE 161020-R  .CTRL  174340-R  .DCCVT 177034-R
        .FATAL 166532-R  .FINIT 172120-R  .GET   161666-R  .GETSQ 161666-R
        .GTDID 162640-R  .ODCVT 177030-R  .OPEN  162656-R  .OPFNB 162666-R
        .PARSE 172436-R  .POSIT 173372-R  .POSRC 167670-R  .PPASC 176140-R
        .PPR50 176054-R  .PRSDI 174666-R  .PRSDV 175112-R  .PRSFN 175306-R
        .PUT   165066-R  .PUTSQ 165066-R  .SAVR1 166410-R  ..ALC1 172552-R
        ..ALOC 172502-R  ..ALUN 170760-R  ..ANSP 166444-R  ..BDRC 171422-R
        ..BKRG 166500-R  ..CREA 171504-R  ..CTRL 174344-R  ..DEL1 171750-R
        ..DID  174376-R  ..DIDF 174054-R  ..DID1 174372-R  ..DIRF 176314-R
        ..EFCK 167360-R  ..EFC1 167366-R  ..ENTR 172010-R  ..EXTD 172602-R
        ..EXT1 172656-R  ..FCSX 166520-R  ..FIND 172024-R  ..FINI 172130-R
        ..GTDI 170570-R  ..IDPB 166632-R  ..MKDL 172224-R  ..MVR1 167534-R
        ..PARS 172450-R  ..PDI  174110-R  ..PDID 170742-R  ..PGCR 167576-R
        ..PNT1 173136-R  ..PSDI 174700-R  ..PSDV 175124-R  ..PSFN 175320-R
        ..PSIT 173416-R  ..PSRC 167726-R  ..PSR1 167712-R  ..QIOW 166672-R
        ..RBLK 173030-R  ..RDRN 167742-R  ..RFDB 167012-R  ..RMOV 172016-R
        ..RTAD 172242-R  ..RWAC 170016-R  ..RWAT 167774-R  ..SEFB 167436-R
        ..SGR5 174534-R  ..STFN 172376-R  ..WAEF 166742-R  ..WAIT 166722-R
```

**Example 6-3 Memory Allocation File for CALC.TSK;1 (Part Printout/FULL/FILES) (continued)**

```
CALC.TSK;1   MEMORY ALLOCATION MAP  TKB D28          PAGE 5
RDIN                  3-JUL-78   10:49

        ..WAND 173006-R  ..WAST 174016-R  ..WBLK 173036-R  ..WTWA 167146-R
        ..WTWD 167146-R  ..WTW1 167152-R  ..XQIO 166534-R  ..XQI1 166552-R
    <. ABS.>: 000000 000000 000000 00000.
        N.OVPT 000006    O$VEF  000037


FILE: RDIN.OBJ;1  TITLE: .MAIN.  IDENT: FORV02
    <$CODE >: 020246 020246 000000 00000.
    <$DATAP>: 024352 024352 000000 00000.
    <$DATA >: 020430 020430 000000 00000.
    <.$$$$.>: 033250 033250 000000 00000.
        $RF2A1 000000-R
    <OTS$I >: 002444 002444 000000 00000.
        $$OTSI 002444-R
    <$CODE >: 020246 020246 000000 00000.
        $$OTSC 020246-R
    <.$$$$.>: 033250 033250 000000 00000.
    <DTA   >: 001002 002443 001442 00802.
    <$DATAP>: 024352 024373 000022 00018.
    <$CODE >: 020246 020337 000072 00058.
    <$DATA >: 020430 022377 001750 01000.


FILE: PROC1.OBJ;1  TITLE: PROC1   IDENT: FORV02
    <$CODE >: 020340 020340 000000 00000.
    <$DATAP>: 024374 024374 000000 00000.
    <$DATA >: 022400 022400 000000 00000.
    <.$$$$.>: 033250 033250 000000 00000.
    <$CODE >: 020340 020340 000000 00000.
        PROC1  020340-R
    <.$$$$.>: 033250 033250 000000 00000.
    <DTA   >: 001002 002443 001442 00802.
    <$DATAP>: 024374 024403 000010 00008.
    <$CODE >: 020340 020413 000054 00044.
    <$DATA >: 022400 022401 000002 00002.


FILE: RPRT.OBJ;1  TITLE: RPRT    IDENT: FORV02
    <$CODE >: 020414 020414 000000 00000.
    <$DATAP>: 024404 024404 000000 00000.
    <$DATA >: 022402 022402 000000 00000.
    <.$$$$.>: 033250 033250 000000 00000.
    <$CODE >: 020414 020414 000000 00000.
        RPRT   020414-R
    <.$$$$.>: 033250 033250 000000 00000.
    <DTA   >: 001002 002443 001442 00802.
    <$DATAP>: 024404 024413 000010 00008.
    <$CODE >: 020414 020427 000014 00012.
    <$DATA >: 022402 024351 001750 01000.


FILE: SYSLIB.OLB;1  TITLE: $ADDM   IDENT: F40002
    <OTS$I >: 002444 002577 000134 00092.
        ADF$IM 002444-R  ADF$MM 002476-R  ADF$PM 002462-R  ADF$SM 002510-R
        SUF$IM 002522-R  SUF$MM 002554-R  SUF$PM 002540-R  SUF$SM 002566-R
```

**Example 6-3 Memory Allocation File for CALC.TSK;1 (Part Printout/FULL/FILES) (continued)**

```
CALC.TSK;1    MEMORY ALLOCATION MAP  TKB D28          PAGE 6
RDIN                  3-JUL-78   10:49


FILE: SYSLIB.OLB;1  TITLE: $CALL    IDENT: F40002
    <OTS$I >: 002600 002635 000036 00030.
        CAI$    002600-R  CAL$    002606-R


FILE: SYSLIB.OLB;1  TITLE: $EOL     IDENT: F40002
    <OTS$I >: 002636 002767 000132 00090.
        EOL$    002666-R  $EOL    002664-R  $IOEXI 002636-R


FILE: SYSLIB.OLB;1  TITLE: $CONVF   IDENT: F40002
    <OTS$I >: 002770 004745 001756 01006.
        DCO$    004024-R  ECO$    004016-R  FCO$    004012-R  GCO$    004004-R
        RCI$    002770-R
    <OTS$P >: 020210 020245 000036 00030.


FILE: SYSLIB.OLB;1  TITLE: $IFR     IDENT: F40002
    <OTS$I >: 004746 005047 000102 00066.
        IFR$    004746-R  $IFR    004752-R  $$IFR   004756-R


FILE: SYSLIB.OLB;1  TITLE: $ISNLS   IDENT: F40001
    <OTS$I >: 005050 005063 000014 00012.
        ISN$    005050-R  LSN$    005056-R


FILE: SYSLIB.OLB;1  TITLE: $IMOVS   IDENT: F40002
    <OTS$I >: 005064 005165 000102 00066.
        MOI$IA 005110-R  MOI$IM 005104-R  MOI$IS 005100-R  MOI$MA 005124-R
        MOI$MM 005120-R  MOI$MS 005114-R  MOI$SA 005074-R  MOI$SM 005070-R
        MOI$SS 005064-R  MOI$0A 005140-R  MOI$0M 005134-R  MOI$0S 005130-R
        MOI$1A 005160-R  MOI$1M 005152-R  MOI$1S 005144-R  MOL$IS 005100-R
        MOL$SS 005064-R  REL$   005100-R


FILE: SYSLIB.OLB;1  TITLE: $BRAS    IDENT: F40002
    <OTS$I >: 005166 005241 000054 00044.
        BEQ$    005212-R  BGE$    005222-R  BGT$    005220-R  BLE$    005210-R
        BLT$    005232-R  BNE$    005230-R  BRA$    005224-R  NMI$1I 005200-R
        NMI$1M 005166-R


FILE: SYSLIB.OLB;1  TITLE: $RETS    IDENT: F40002
    <OTS$I >: 005242 005311 000050 00040.
        RET$    005256-R  RET$F   005246-R  RET$I   005254-R  RET$L   005242-R


FILE: SYSLIB.OLB;1  TITLE: $FVEC    IDENT: F40002
    <OTS$I >: 005312 005355 000044 00036.
        SAF$IM 005312-R  SAF$MM 005346-R  SAF$SM 005314-R  SVF$IM 005324-R
        SVF$MM 005352-R  SVF$SM 005326-R


FILE: SYSLIB.OLB;1  TITLE: $TRARY   IDENT: F40002
```

**Example 6-3 Memory Allocation File for CALC.TSK;1 (Part Printout/FULL/FILES) (continued)**

```
CALC.TSK;1   MEMORY ALLOCATION MAP  TKB D28          PAGE 7
RDIN                  3-JUL-78   10:49

    <OTS$I >: 005356 005635 000260 00176.
        TAD$   005406-R  TAF$   005414-R  TAI$   005356-R  TAL$   005364-R
        TAP$   005400-R  TAQ$   005372-R  $TAD   005406-R  $TAF   005414-R
        $TAI   005356-R  $TAL   005364-R  $TAP   005400-R  $TAQ   005372-R


FILE: SYSLIB.OLB;1  TITLE: $OTI     IDENT: F40002
    <. ABS.>: 000000 000000 000000 00000.
        V007A  000000
    <OTS$I >: 005636 006107 000252 00170.


FILE: SYSLIB.OLB;1  TITLE: $SUBR    IDENT: F40002
    <OTS$I >: 006110 006231 000122 00082.
        $OTIS  006110-R  $$OTIS 006112-R


FILE: SYSLIB.OLB;1  TITLE: $OTV     IDENT: F40001
    <. ABS.>: 000000 000000 000000 00000.
        F.BFHD 000020    F.FDB  000154
    <OTS$I >: 006232 006232 000000 00000.
    <$$AOTS>: 024414 025317 000704 00452.
        $NAMC  024466-R  $OTSVA 024464-R  $SEQC  024464-R  $SST   025300-R
        .MOLUN 024472-R  .NLUNS 024470-R
    <$$DEVT>: 025320 025320 000000 00000.
    <$$FSR1>: 026530 026530 000000 00000.
    <$$IOB1>: 032734 033137 000204 00132.
    <$$IOB2>: 033140 033140 000000 00000.
    <$$OBF1>: 033140 033247 000110 00072.
    <$$OBF2>: 033250 033250 000000 00000.


FILE: SYSLIB.OLB;1  TITLE: $OTSV    IDENT: V0101A
    <. ABS.>: 000000 000000 000000 00000.
        $OTSV  000004


FILE: SYSLIB.OLB;1  TITLE: $CONVI   IDENT: F40002
    <OTS$I >: 006232 006637 000406 00262.
        ICI$   006240-R  ICO$   006442-R  OCI$   006232-R  OCO$   006434-R
        $ECI   006254-R
    <OTS$P >: 020210 020245 000036 00030.


FILE: SYSLIB.OLB;1  TITLE: $SAVRE   IDENT: F40002
    <OTS$I >: 006640 007017 000160 00112.
        SAVRC$ 006640-R  THRD$  007016-R


FILE: SYSLIB.OLB;1  TITLE: $FCHNL   IDENT: F40001
    <OTS$I >: 007020 007055 000036 00030.
        $FCHNL 007020-R


FILE: SYSLIB.OLB;1  TITLE: $FIO     IDENT: F40002
```

**Example 6-3 Memory Allocation File for CALC.TSK;1 (Part Printout/FULL/FILES) (continued)**

```
CALC.TSK;1   MEMORY ALLOCATION MAP  TKB D28          PAGE 8
RDIN                  3-JUL-78   10:49

    <OTS$I >: 007056 010751 001674 00956.
        $FIO   007614-R  $$FIO  007620-R
    <OTS$P >: 020210 020245 000036 00030.


FILE: SYSLIB.OLB;1  TITLE: $OPEN   IDENT: F40002
    <OTS$I >: 010752 011367 000416 00270.
        $FLDEF 011276-R  $OPEN  010752-R


FILE: SYSLIB.OLB;1  TITLE: $GETRE  IDENT: F40001
    <OTS$I >: 011370 011465 000076 00062.
        $GETRE 011370-R


FILE: SYSLIB.OLB;1  TITLE: $INITI  IDENT: F40002
    <OTS$I >: 011466 011553 000066 00054.
        $INITI 011466-R


FILE: SYSLIB.OLB;1  TITLE: $STPPA  IDENT: F40001
    <OTS$I >: 011554 012017 000244 00164.
        BAH$   011650-R  EXIT$  011732-R  FOO$   011722-R  PSE$   011622-R
        STP$   011716-R  $EXIT  011732-R  $PSE   011626-R  $PSES  011660-R
        $STP   011716-R  $STPS  011710-R


FILE: SYSLIB.OLB;1  TITLE: $ERRPT  IDENT: F40001
    <OTS$I >: 012020 013717 001700 00960.
        $BINAS 013212-R  $ERRAA 012206-R  $ERRTB 013530-R  $ERRTE 013720-R
        $ERRZA 013150-R  $ERXIT 012436-R  $EXIT$ 012124-R  $FILL  013266-R
        $SST0  012020-R  $SST1  012026-R  $SST2  012040-R  $SST3  012046-R
        $SST4  012054-R  $SST5  012062-R  $SST6  012146-R  $SST7  012072-R
        $SVTK$ 013522-R


FILE: SYSLIB.OLB;1  TITLE: $FPERR  IDENT: F40002
    <OTS$I >: 013720 014123 000204 00132.
        $FPERR 013720-R


FILE: SYSLIB.OLB;1  TITLE: $NVINI  IDENT:
    <. BLK.>: 001000 001001 000002 00002.
        $VIRIN 001000-R


FILE: SYSLIB.OLB;1  TITLE: FCSFSR  IDENT: 0303MS
    <. ABS.>: 000000 000000 000000 00000.
        .FSRPT 000002
    <$$FSR1>: 026530 026530 000000 00000.
    <$$FSR2>: 032630 032733 000104 00068.
        .FSRCB 032630-R  .MBFCT 032730-R


FILE: SYSLIB.OLB;1  TITLE: $ERQIO  IDENT: F40001
```

**Example 6-3 Memory Allocation File for CALC.TSK;1 (Part Printout/FULL/FILES) (continued)**

```
CALC.TSK;1    MEMORY ALLOCATION MAP  TKB D28          PAGE 9
RDIN                    3-JUL-78   10:49


    <OTS$I >: 014124 014417 000274 00188.
        $ATT   014124-R  $DET   014230-R  $ERRWT 014314-R


FILE: SYSLIB.OLB;1  TITLE: $CLOSE  IDENT: F40001
    <OTS$I >: 014420 014507 000070 00056.
        $CLOSE 014420-R


FILE: SYSLIB.OLB;1  TITLE: $ERTXT  IDENT: F40004
    <OTS$I >: 014510 020073 003364 01780.
        $ERTXT 014510-R


FILE: SYSLIB.OLB;1  TITLE: $R50    IDENT: F40001
    <OTS$I >: 020074 020207 000114 00076.
        $R50   020074-R



*** TASK BUILDER STATISTICS:

    TOTAL WORK FILE REFERENCES: 19016.
    WORK  FILE  READS: 0.
    WORK  FILE WRITES: 0.
    SIZE OF CORE POOL: 16010. WORDS (62. PAGES)
    SIZE OF WORK FILE: 3072. WORDS (12. PAGES)

    ELAPSED TIME:00:00:17
```

**Example 6–4 Memory Allocation for CALC.TSK;2**

```
CALC.TSK;2   MEMORY ALLOCATION MAP  TKB D28          PAGE 1
                    3-JUL-78   10:50


PARTITION NAME : GEN
IDENTIFICATION : FORV02
STACK    LIMITS: 000000 000777 001000 00512.
PRG XFR ADDRESS: 020246
TOTAL ATTACHMENT DESCRIPTORS: 3.
TASK  IMAGE  SIZE  : 6176. WORDS
TASK HEADER SIZE: 160. WORDS
TASK ADDRESS LIMITS: 000000 030167
R-W DISK BLK LIMITS: 000003 000032 000030 00024.

*** ROOT SEGMENT: RDIN


R/W MEM  LIMITS: 000000 030167 030170 12408.
DISK BLK LIMITS: 000003 000033 000031 00025.


MEMORY ALLOCATION SYNOPSIS:

SECTION                                          TITLE  IDENT  FILE
-------                                          -----  -----  ----
. BLK.:(RW,I,LCL,REL,CON) 001000 000002 00002.
DTA    :(RW,D,GBL,REL,OVR) 001002 001442 00802.
                          001002 001442 00802. .MAIN. FORV02 RDIN.OBJ;1
                          001002 001442 00802. PROC1  FORV02 PROC1.OBJ;1
                          001002 001442 00802. RPRT   FORV02 RPRT.OBJ;1
OTS$I  :(RW,I,LCL,REL,CON) 002444 015544 07012.
                          002444 000000 00000. .MAIN. FORV02 RDIN.OBJ;1
OTS$P  :(RW,D,GBL,REL,OVR) 020210 000036 00030.
$CODE  :(RW,I,LCL,REL,CON) 020246 000162 00114.
                          020246 000000 00000. .MAIN. FORV02 RDIN.OBJ;1
                          020246 000000 00000. .MAIN. FORV02 RDIN.OBJ;1
                          020246 000072 00058. .MAIN. FORV02 RDIN.OBJ;1
                          020340 000000 00000. PROC1  FORV02 PROC1.OBJ;1
                          020340 000000 00000. PROC1  FORV02 PROC1.OBJ;1
                          020340 000054 00044. PROC1  FORV02 PROC1.OBJ;1
                          020414 000000 00000. RPRT   FORV02 RPRT.OBJ;1
                          020414 000000 00000. RPRT   FORV02 RPRT.OBJ;1
                          020414 000014 00012. RPRT   FORV02 RPRT.OBJ;1
$DATA  :(RW,D,LCL,REL,CON) 020430 003722 02002.
                          020430 000000 00000. .MAIN. FORV02 RDIN.OBJ;1
                          020430 001750 01000. .MAIN. FORV02 RDIN.OBJ;1
                          022400 000000 00000. PROC1  FORV02 PROC1.OBJ;1
                          022400 000002 00002. PROC1  FORV02 PROC1.OBJ;1
                          022402 000000 00000. RPRT   FORV02 RPRT.OBJ;1
                          022402 001750 01000. RPRT   FORV02 RPRT.OBJ;1
$DATAP:(RW,D,LCL,REL,CON) 024352 000042 00034.
                          024352 000000 00000. .MAIN. FORV02 RDIN.OBJ;1
                          024352 000022 00018. .MAIN. FORV02 RDIN.OBJ;1
                          024374 000000 00000. PROC1  FORV02 PROC1.OBJ;1
                          024374 000010 00008. PROC1  FORV02 PROC1.OBJ;1
                          024404 000000 00000. RPRT   FORV02 RPRT.OBJ;1
                          024404 000010 00008. RPRT   FORV02 RPRT.OBJ;1
$$AOTS:(RW,D,LCL,REL,CON) 024414 000704 00452.
```

**Example 6-4 Memory Allocation File for CALC.TSK;2 (continued)**

```
CALC.TSK;2    MEMORY ALLOCATION MAP  TKB D28          PAGE 2
RDIN                    3-JUL-78   10:50


$$AUTO:(RW,I,LCL,REL,CON) 160000 000130 00088.
$$DEVT:(RW,D,LCL,REL,OVR) 025320 001210 00648.
$$FSR1:(RW,D,GBL,REL,OVR) 026530 001020 00528.
$$FSR2:(RW,D,GBL,REL,CON) 027550 000104 00068.
$$IOB1:(RW,D,LCL,REL,OVR) 027654 000204 00132.
$$IOB2:(RW,D,LCL,REL,OVR) 030060 000000 00000.
$$LOAD:(RW,I,LCL,REL,CON) 160130 000170 00120.
$$MRKS:(RW,I,LCL,REL,OVR) 160320 000166 00118.
$$OBF1:(RW,D,LCL,REL,CON) 030060 000110 00072.
$$OBF2:(RW,I,LCL,REL,CON) 030170 000000 00000.
$$OVRS:(RW,I,LCL,ABS,CON) 000000 000000 00000.
$$RDSG:(RW,I,LCL,REL,OVR) 160506 000312 00202.
$$RESL:(RW,I,LCL,REL,CON) 161020 016216 07310.
.$$$$.:(RW,D,GBL,REL,OVR) 030170 000000 00000.
                          030170 000000 00000. .MAIN. FORV02 RDIN.OBJ;1
                          030170 000000 00000. .MAIN. FORV02 RDIN.OBJ;1
                          030170 000000 00000. PROC1  FORV02 PROC1.OBJ;1
                          030170 000000 00000. PROC1  FORV02 PROC1.OBJ;1
                          030170 000000 00000. RPRT   FORV02 RPRT.OBJ;1
                          030170 000000 00000. RPRT   FORV02 RPRT.OBJ;1


GLOBAL SYMBOLS:

PROC1  020340-R  $RF2A1 000000-R  $$OTSI 002444-R
RPRT   020414-R  $$OTSC 020246-R


*** TASK BUILDER STATISTICS:

    TOTAL WORK FILE REFERENCES: 14413.
    WORK  FILE  READS: 0.
    WORK  FILE WRITES: 0.
    SIZE OF CORE POOL: 16010. WORDS (62. PAGES)
    SIZE OF WORK FILE: 3072. WORDS (12. PAGES)

    ELAPSED TIME:00:00:13
```

# 7 OVERLAY CAPABILITY

The Task Builder provides you with a means of reducing the memory and/or virtual address space requirements of a task by means of overlay structures created with the aid of the Overlay Description Language (ODL). Two kinds of overlay segments can be specified: those that reside on disk, and those that reside permanently in memory.

## 7.1 Overlay Description

To create an overlay structure, you divide a task into a series of segments:

1 A single root segment, which is always in memory.

2 Any number of overlay segments, which either reside on disk and share virtual address space and memory with one another; or which reside in memory but share virtual address space.

A segment consists of a set of modules and p-sections. Segments that overlay each other must be logically independent; that is, the components of one segment cannot reference the components of a segment with which it shares virtual address space. In addition to the logical independence of the overlay segments, you must consider the general flow of control within the task.

You must also consider what kind of overlay segment is most suitable, and how it will be constructed. Dividing a task into disk-resident overlays saves physical space, but introduces the overhead activity of loading these segments each time they are needed, but not present in memory. Memory-resident overlays, on the other hand, are loaded from disk only the first time they are referenced. Thereafter, they remain in memory and are referenced by re-mapping.

There are several large classes of tasks that can be handled effectively by an overlay structure. For example, a task that moves sequentially through a set of modules is well-suited to the use of an overlay structure. A task that selects one of a set of modules according to the value of an item of input data is also well-suited to an overlay structure.

### 7.1.1 Disk-Resident Overlay Structure

Disk-resident overlays conserve memory by sharing it. Segments that are logically independent need not be present in memory at the same time. They can therefore occupy a common physical area in memory whenever either needs to be used.

Consider a task, TK1, which consists of four input files. Each input file consists of a single module of the same name as the file. The task is built by the following command:

```
PDS> LINK/TASK:TK1
FILE? CNTRL,A,B,C
```

Suppose you know that the modules A, B, and C are logically independent. In this example:

A does not call B or C and does not use the data of B or C
B does not call A or C and does not use the data of A or C
C does not call A or B and does not use the data of A or B.

You can define an overlay structure in which A, B, and C are overlay segments that occupy the same storage. Suppose further that the flow of control for the task is as follows:

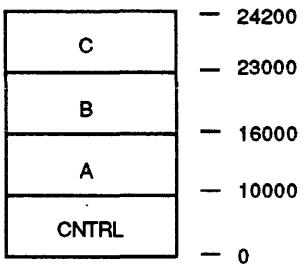
CNTRL calls A and A returns to CNTRL
CNTRL calls B and B returns to CNTRL
CNTRL calls C and C returns to CNTRL
CNTRL calls A and A returns to CNTRL

The loading of overlays occurs only four times during the execution of the task. Therefore, the user can reduce the memory requirements of the task without unduly increasing the overhead activity.

Consider the effect of introducing an overlay structure on the allocation of memory for the task. Suppose the lengths of the modules are as follows:

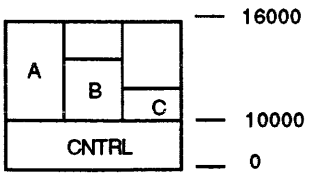
| | |
|---|---|
| CNTRL | 10,000 bytes |
| A | 6,000 bytes |
| B | 5,000 bytes |
| C | 1,200 bytes |

The memory allocation produced as a result of building the task as a single segment on a system with memory mapping hardware is as follows:




The memory allocation for a single-segment task requires 24200 bytes.

The memory allocation produced as a result of using the overlay capability and building a multi-segment task is as follows:

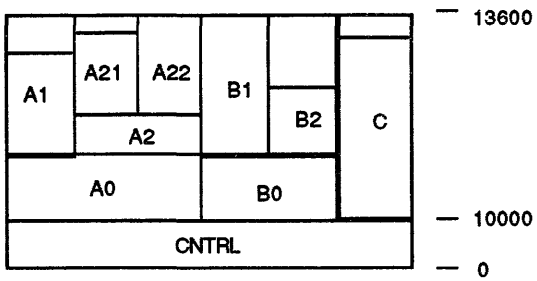



The multi-segment task requires 16,000 bytes. In addition to the module storage, additional storage is required for overhead connected with handling the overlay structure. This overhead is described later and illustrated in the example CALC.

The amount of storage required for the task is determined by the length of the root segment and the length of the longest overlay segment. Overlay segments A and B in this representation are much longer than overlay segment C. If the user can divide A and B into sets of logically independent modules, further reduction can be made in the storage requirements for the task.

Segment A is divided into a control program, A0, and two overlays, A1 and A2. Then, A2 is further divided into a main part ,A2, and two overlays ,A21 and A22. Similarly, segment B is divided into a control module, B0, and two overlays, B1 and B2.

The memory allocation for the task produced by the additional overlays defined for A and B is given in the following diagram:




As a single-segment task, TK1 required 24,200 bytes of storage. The first overlay structure reduced the requirement by 6,200 bytes. The second overlay structure further reduced the storage requirement by 2,200 bytes.

A vertical line drawn through the memory diagram indicates a state of memory at some point in time during the execution of the task. In the diagram given here, the leftmost such line gives memory when CNTRL, A0, and A1 are loaded: the next such line gives memory when CNTRL, A0, A2, and A21 are loaded: and so on.

A horizontal line can be drawn through the memory diagram to indicate segments that share the same storage. In the given diagram, the uppermost such line indicates A1, A21, A22, B1, B2 and C, all of which can use the same memory; the next such line gives A1, A2, B1, B2, and C; and so on.

## 7.1.2 Memory-Resident Overlay Structure

The Task Builder provides for the creation of overlay segments that are loaded from disk only the first time they are referenced. Thereafter, they are permanently resident in memory, sharing virtual address space in the same way as disk-resident overlays. Unlike disk-resident overlays, however, memory-resident overlays do not share physical memory. Instead, they reside in separate areas of memory, each one aligned on a 32-word boundary. Memory-resident overlays save time for a running task because they do not need to be copied from a secondary storage device each time they are to overlay other segments.

"Loading" a memory-resident overlay reduces to mapping a set of shared virtual addresses to the unique permanent physical area of memory containing the overlaying segment. This process is shown in Figure 7–1.

It is important that you exercise discretion in choosing whether to have memory-resident overlays in a structure. Indiscriminate use of these segments can result in inefficient allocation of virtual memory. This is because virtual memory is allocated in blocks that are 4K words long. Consequently, the length of each overlay segment should approach that limit if you are to minimize waste. (A segment that was one word longer than 4K, for example, would be allocated 8K of virtual memory. All but one word of the second 4K would be unusable.)

You should also maintain control over the contents of each segment in order to prevent wasted physical memory. The inclusion of a module in several memory-resident segments that overlay one another, causes storage to be reserved for each extra copy of the module. Common modules, including those from the system object module library, should be placed in a segment that can be accessed from all referencing segments.

The criterion for choosing to have memory-resident overlays is the need to save virtual address space when one of the following conditions exists:

1 Disk-resident overlays are undesirable

(because they would slow down the system to a point that is unacceptable),

or

2 Disk-resident overlays are impossible

(because the segments are portions of a shareable global area or other shared region, and must be permanently resident in memory).

Large systems can be utilized to better advantage because of the ability to save time when a large amount of physical memory is available. Shareable global areas can benefit especially, from the virtual-memory-saving attribute, by being divided into memory-resident segments.

If all the code in a resident overlay is contained in read-only p-sections, the overlay segment is marked as read-only. Such a segment is shared between all active versions of the task, in the same way as the task pure area.

Where there are commonly several versions of a task active, use of read-only resident overlays can save a significant amount of physical memory.

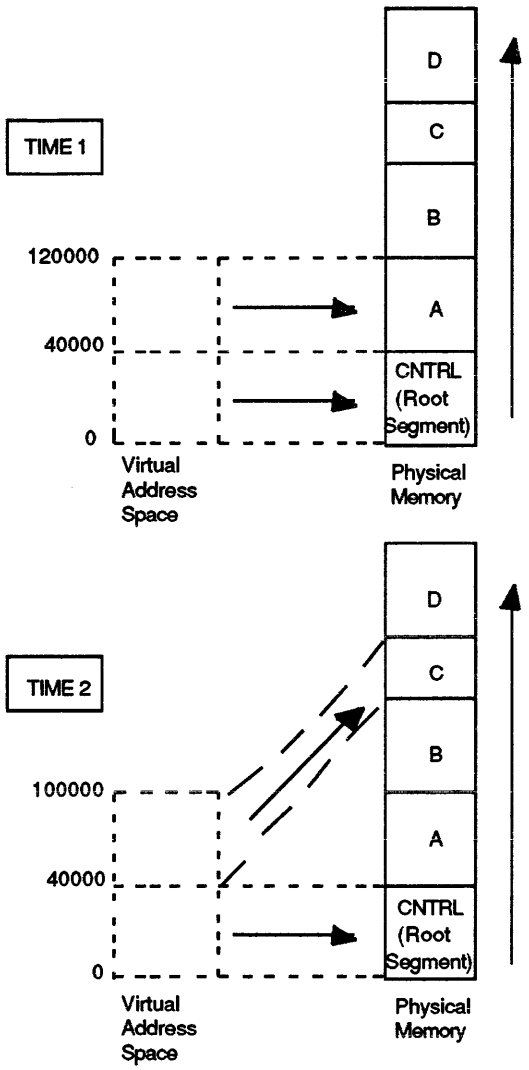
When a task has resident overlays, the physical memory for all of them is allocated when the task is loaded, and normally remains allocated until the task exits. The resident overlays can be removed from memory (by checkpointing or swapping) only when the task itself is so removed. This also applies to overlays which are not currently mapped.

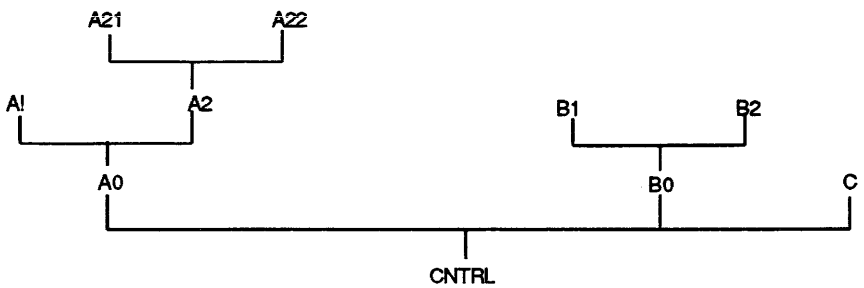
## 7.1.3 Overlay Tree

The Task Builder provides a language for representing an overlay structure consisting of one or more trees.

The memory allocation for the previous example can be represented by the single overlay tree shown below:

**Figure 7–1 Mapping Memory-Resident Overlays**



The tree has a root, CNTRL, and three main branches, A0, B0, and C. The tree has six leaves, A1, A21, A22, B1, B2, and C.

The tree has as many paths as it has leaves. The path down is defined from the leaf to the root, for example:

```
A21-A2-A0-CNTRL
```

The path up is defined from the root to the leaf, for example:

```
CNTRL-B0-B1.
```

Understanding the tree and its paths is important to the understanding of the overlay loading mechanism and the resolution of global symbols.

### Loading Mechanism

Modules can call other modules that exist on the same path. The module CNTRL is common to every path of the tree and, therefore, can call and be called by every module in the tree. The module A2 can call the modules A21, A22, A0, and CNTRL; but A2 can not call A1, B1, B2, B0 or C.

When a module calls a module in another overlay segment, the overlay segment must be in memory or must be brought into memory. The methods for loading overlays are described in Chapter 8, Section 8.1 and Section 8.2.

### Resolution of Global Symbols in a Multi-segment Task

The Task Builder performs the same activities in resolving global symbols for a multi-segment task as it does for a single segment task. The rules defined in Chapter 6, Section 6.1.11 for the resolution of global symbols in a single segment task still apply, but the scope of the global symbols is altered by the overlay structure.

In a single segment task, any global definition can be referenced by any module. In a multi-segment task, a module can only reference a global symbol that is defined on a path that passes through the segment to which the module belongs.

In a single segment task, if two global symbols with the same name are defined, the symbols are multiply-defined and (if the values differ) an error message is produced.

In a multi-segment task:

1 Two global symbols with the same name can be defined if they are on separate paths, and not referenced from a segment that is common to both.

2 If a global symbol is defined more than once on separate paths, but referenced from a segment that is common to both, the symbol is ambiguously defined.

3 If a global symbol is defined more than once on a single path, it is multiply-defined.

The procedure for resolving global symbols can be summarized as follows:

1 The Task Builder selects an overlay segment for processing.

2 Each module in the segment is scanned for global definitions and references.

3 If the symbol is a definition, the Task Builder searches all segments on paths that pass through the segment being processed, and looks for references that must be resolved.

4 If the symbol is a reference, the Task Builder performs the tree search as described in step 3, looking for an existing definition.

5 If the symbol is new, it is entered in a list of global symbols associated with the segment.

Overlay segments are selected for processing in an order corresponding to their distance from the root. That is, the Task Builder considers a segment farther away from the root, before processing an adjoining segment.

When a segment is being processed, the search for global symbols proceeds in the following order:

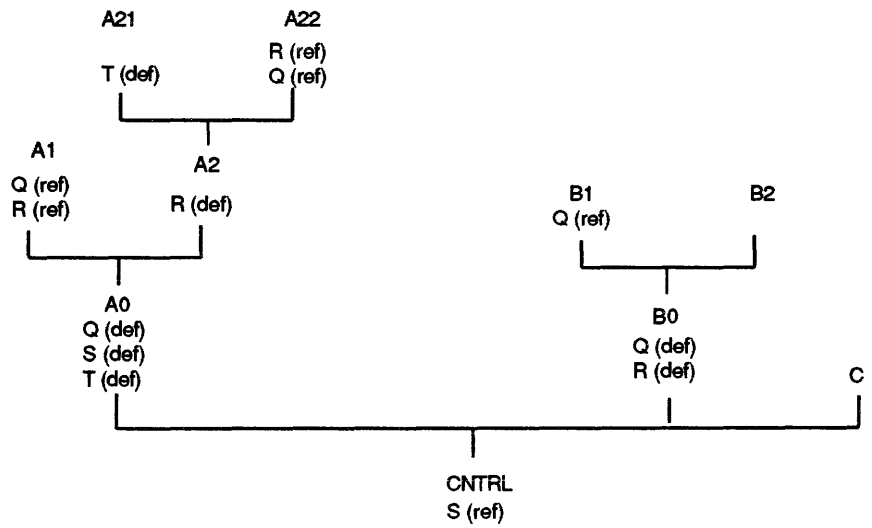
1 The segment being processed.

2 All segments toward the root.

3 All segments away from the root.

4 All co-trees (see Section "Resolution of P-sections in a Multi-segment Task").

Consider the task TK1 and the global symbols Q, R, S, and T.




The symbols shown in the diagram are described below:

Q The global symbol Q is defined in the segment A0 and in the segment B0. The reference to Q in segment A22 and the reference to Q in segment A1 are resolved to the definition in A0. The reference to Q in B1 is resolved to the definition in B0. The two definitions of Q are distinct in all respects and occupy different memory allocations.

R The global symbol R is defined in the segment A2. The reference to R in A22 is resolved to the definition in A2 because there is a path to the reference from the definition (CNTRL-A0-A2-A22). The reference to R in A1, however, is undefined because there is no definition for R on a path through A1.

S The global symbol S is defined in A0 and B0. References to S from A1, A21, or A22 are resolved to the definition in A0, and references to S in B1 and B2 are resolved to the definition in B0. However, the reference to S in CNTRL cannot be resolved because there are two definitions of S on separate paths through CNTRL. S is ambiguously defined.

T The global symbol T is defined in A21 and A0. Since there is a single path through the two definitions (CNTRL-A0-A2-A21), the global symbol T is multiply-defined.

### Resolution of Global Symbols from the Default Library

The process of resolving global symbols may require two passes over the tree structure. The global symbols described in the previous section are included in user-specified input modules that are scanned by the Task Builder in the first pass. If any undefined symbols remain, the Task Builder initiates a second pass over the structure in an attempt to resolve such symbols by searching the default object module library (normally LB0:[1,1]SYSLIB.OLB). Any undefined symbols remaining after the second pass are reported to you.

When multiple tree structures (co-trees) are defined, any resolution of global symbols across tree structures during a second pass can result in multiple or ambiguous definitions. In addition, such references can cause overlay segments to be inadvertently displaced from memory by the overlay loading routines, thereby causing run-time failures to occur. To eliminate these conditions, the tree search on the second pass is restricted to:

1 The segment in which the undefined reference has occurred

2 All segments in the current tree that are on a path through the segment

3 The root segment

When the current segment is the main root, the tree search is extended to all segments. You can unconditionally extend the tree search to all segments by including the /FULL_SEARCH PDS qualifier (/FU MCR switch).

### Resolution of P-sections in a Multi-segment Task

Each p-section has an attribute that indicates whether the p-section is local (LCL) to the segment in which it is defined or of global (GBL) extent.

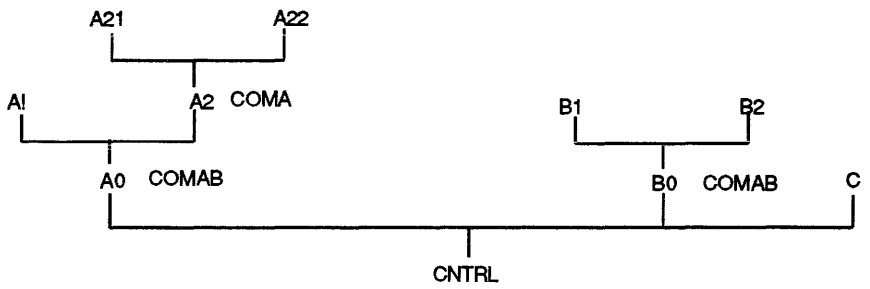
Local p-sections with the same name can appear in any number of segments. Storage is allocated for each local p-section in the segment in which it is declared. Global p-sections of the same name, however, must be resolved by the Task Builder.

When a global p-section is defined in several overlay segments along a common path, the Task Builder allocates all storage for the p-section in the overlay segment closest to the root.

FORTRAN common blocks are translated into global p-sections with the overlay (OVR) attribute. Suppose that in the task TK1, the common block COMA is defined in modules A2 and A21. The Task Builder allocates the storage for COMA in A2 because that segment is closer to the root than the segment which contains A21.

However, if the segments A0 and B0 use a common block COMAB, the Task Builder allocates the storage for COMAB in both the segment which contains A0 and the segment which contains B0. A0 and B0 can not communicate through COMAB. When the overlay segment containing B0 is loaded, any data stored in COMAB by A0 is lost.

The tree for the task TK1 including the allocation of the common blocks COMA and COMAB is:




You can specify the allocation of p-sections. If A0 and B0 need to share the contents of COMAB, you can force the allocation of this p-section into the root segment by the use of the .PSECT directive, described in Section ".PSECT Directive".

Misalignment between a global tag within the .PSECT and the resulting task image in a multi-segment task can occur if you reference a global .PSECT that is also defined in a module in the default library. This condition can be corrected by:

1 Explicitly specifying the default library as the last module in the ODL, or

2 Including the referenced library modules directly in the ODL specification.

## 7.1.4 Overlay Description Language (ODL)

The Task Builder provides a language that allows you to describe the overlay structure. The overlay description language (ODL) contains five directives by which you can describe the overlay structure of a task.

An overlay description consists of a series of ODL directives. There must be one .ROOT directive and one .END directive. The .ROOT directive tells the Task Builder where to start building the tree and the .END directive tells the Task Builder where the input ends.

### .ROOT and .END Directives

The arguments of the .ROOT directive make use of two operators to express concatenation and overlaying. A pair of parentheses delimits a group of segments that start at the same location in memory. The number of nested parentheses cannot exceed 16.

1 The operator dash (-) indicates the concatenation of storage. For example, X-Y means that the memory allocation must contain X and Y simultaneously. So X and Y are allocated in sequence.

2 The operator comma (,) appearing within parentheses indicates the overlaying of storage. For example, Y,Z means that memory can contain either Y or Z. Therefore Y and Z can share storage.

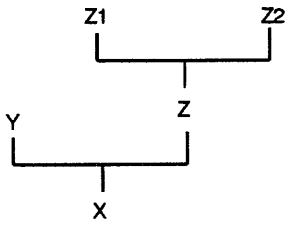
   This operator is also used to define multiple tree structures, as described in Section 7.1.5.

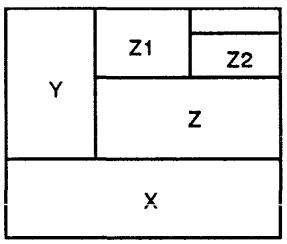
3 The operator exclamation mark (!) indicates memory residency of overlays. See Section 7.1.2.

The following ODL directives:

```
.ROOT X-(Y,Z-(Z1,Z2))
.END
```

describe the following tree and its corresponding memory diagram:




To create the overlay description for the task TK1 described earlier in this chapter, the user creates a file TFIL.ODL that contains the directives:

```
.ROOT CNTRL-(A0-(A1,A2-(A21,A22)),B0-(B1,B2),C)
.END
```

To build the task with that overlay structure, the user types:

```
PDS> LINK/TASK:TK1/OVERLAY:TFIL
```

or

```
MCR>TKB TK1=TFIL/MP
```

The OVERLAY qualifier tells the Task Builder that the file TFIL.ODL contains an overlay description for the task.

The qualifiers on input file specifications in the ODL files are always given in the MCR form.

### .FCTR Directive

The tree that represents the overlay structure can be complicated. The ODL includes another directive, .FCTR, which allows you to build large trees and represent them systematically.

The .FCTR directive allows you to extend the tree description beyond a single line. Since there can be only one .ROOT directive, the .FCTR directive must be used if the tree definition exceeds one line. The .FCTR directive, however, can also be used to introduce clarity in the representation of the tree.

The maximum number of nested .FCTR levels is 32.

To simplify the tree given in the file TFIL the .FCTR directive is introduced into the ODL as follows:

```
          .ROOT CNTRL-(AFCTR,BFCTR,C)
AFCTR:    .FCTR A0-(A1,A2-(A21,A22))
BFCTR:    .FCTR B0-(B1,B2)
          .END
```

The label BFCTR, is used in the .ROOT directive to designate the argument of the .FCTR directive, B0-(B1,B2). The resulting overlay description is easier to interpret than the original description. The tree consists of a root, CNTRL, and three main branches. Two of the main branches have sub-branches.

The .FCTR directive can be nested. You can modify file TFIL.ODL as follows:

```
            .ROOT CNTRL-(AFCTR,BFCTR,C)
AFCTR:      .FCTR A0-(A1,A2FCTR)
A2FCTR:     .FCTR A2-(A21,A22)
BFCTR:      .FCTR B0-(B1,B2)
            .END
```

The decision to use the .FCTR directive is based on considerations of space, style, and readability of a complex ODL file.

### Exclamation Point Operator

The exclamation point operator allows you to specify overlay segments that will permanently reside in memory rather than on disk. Memory residency is specified by placing an exclamation point (!) immediately before the left parenthesis enclosing the segments to be affected:

```
.ROOT A-!(B,C)
```

In the example above, segments B and C are declared resident in separate areas of memory. The single starting virtual address for both B and C is determined by the Task Builder, by rounding the octal length of segment A up to the next 4K boundary. The physical memory allocated to segments B and C is determined by rounding the actual length of each segment to the next 32-word boundary and adding this value to the total memory required by the task.

The exclamation point operator applies only to segments at the first level inside a pair of parentheses; segments in parentheses nested within that level are not affected. It is therefore possible to define an overlay structure that combines the space-saving attributes of disk-resident overlays, with the speed of memory-resident overlays.

The following example shows this:

```
.ROOT A-!(B1-(B2,B3),C)
.END
```

In this example above B1 and C are declared memory-resident by the exclamation point operator.

B2 and B3 are declared disk-resident because no exclamation point operator is present before the parentheses enclosing them.

While it is perfectly valid for a memory-resident overlay to call a disk-resident overlay, it is illegal to build the following type of structure; that is, an exclamation point cannot be used for segments emanating from a disk-resident segment (in this case, B1):

```
.ROOT A-(B1-!(B2,B3),C
.END
```

The exclamation point operator will be ignored unless the task has been built using the /RESIDENT_OVERLAYS PDS qualifier (/RO MCR switch)

### .NAME Directive

The .NAME directive allows you to specify a name for a segment, and in so doing, to attach attributes to the segment. The name must be unique with respect to filenames, p-section names, .FCTR labels and other segment names that are used in the overlay description.

The chief uses of this directive are:

1 To name uniquely a segment that is to be loaded through the manual load facility, and

2 To permit a segment, that does not contain executable code, to be loaded through the autoload mechanism.

(Loading mechanisms are described in Chapter 8.)

The format of the .NAME directive is

```
.NAME segname [,attr][,attr]
```

where:

- segname - is a 1- to 6-character name composed from the character set A-Z, 0-9 and $.
- ,attr - denotes an optional attribute.

attr is one of the following:

| | |
|---|---|
| GBL | The name is entered in the segment's global symbol table.<br>GBL makes it possible to load non-executable overlay segments by means of the autoload mechanism (see Chapter 8, Section 8.1). |
| NODSK | No disk space is allocated to the named segment.<br>If a data overlay segment has no initial values, but will have its contents established by the running task, no space for the task image on disk need be reserved. If NODSK has been specified, an attempt to initialize the segment with data at task-build time results in a fatal error. |
| NOGBL | The name is not entered in the segment's global symbol table. If GBL is not present NOGBL is assumed. |
| DSK | Disk storage is allocated to the named segment. If NODSK is not present, DSK is assumed. |
| NOPHY | No memory is allocated to the segment. Addresses are allocated in the segment starting at relative zero, but the segment cannot be loaded by the overlay run-time system.<br>NOPHY allows data other than the task itself to be included in the task image file, for example, error messages. The IO.LOV QIO function code can be used to load all or part of the segment into a specified buffer, as described in Chapter 8, Section 8.1. |

The attributes described are not attached to a segment until the name is used in a .ROOT or .FCTR statement that defines an overlay segment. When multiple segment names are applied to a segment, the attributes of the last name given take effect.

In the following modified tree for TK1, you give names to the three main branches, A0, B0 and C, by specifying them in the .NAME directive, and using them in the .ROOT directive. The default attributes NOGBL and DSK are in effect for BRNCH1 and BRNCH3. BRNCH2 has the complement attributes GBL and NODSK which will cause the name BRNCH2 to be entered into its segment's global symbol table, and the allocation of disk space for the segment to be suppressed. B0, B1 and B2 can contain either data or executable code; the other two branches must contain executable code.

```
                .NAME BRNCH1
                .NAME BRNCH2,GBL,NODSK
                .NAME BRNCH3
                .ROOT CNTRL-(BRNCH1-AFCTR,BRNCH2-BFCTR,BRNCH3-C)
AFCTR:          .FCTR A0-(A1,A2-(A21,A22))
BFCTR:          .FCTR B0-*(B1,B2)
                .END
```

(* is the autoload indicator, see Chapter 8, Section 8.1.)

The data overlay segment BRNCH2 is loaded by including the following statement in the user's program.

```
CALL BRNCH2
```

This action is immediately followed by an automatic return to the next instruction in the program.

Segment names are also used in making patches with the options ABSPAT, GBLPAT and SYMPAT (see Section 5.5 onwards).

If no segment name is specified the Task Builder establishes a segment name, using the first .PSECT or module name occurring in the segment.

### .PSECT Directive

The .PSECT directive allows the placement of a global p-section to be specified directly. The name of the p-section and its attributes are given in the .PSECT directive. The name can then be used explicitly in the definition of the tree to indicate the segment in which the p-section is to be allocated. It can also be used to force a p-section to be shared (see Section "Resolution of P-sections in a Multi-segment Task").

A problem could be encountered in communication resulting from the overlay description for TK1 if you were careful about the logical independence of the modules in the overlay segment, but failed to take into account the logical independence requirement of multiple executions of the same overlay segment.

The flow of the task TK1, as described earlier in this chapter, is summarized in the following way. CNTRL calls each of the overlay segments and the overlay segment returns to CNTRL in the following order: A,B,C,A. The module A is executed twice. The overlay segment containing A must be reloaded for the second execution of A.

The module A uses the common block DATA3 and the Task Builder allocates DATA3 in the overlay segment containing A. The first execution of A stores some results in DATA3. The second execution of A requires these values. In the present overlay description, however, the values calculated by the first execution of A are overlaid. When the segment containing A is read in for the second execution, the common block is in its initial state.

The use of a .PSECT directive forces the allocation of DATA3 into the root segment to permit the two executions of A to communicate. File TFIL.ODL is modified as follows:

```
            .PSECT DATA3,RW,D,GBL,REL,OVR
            .ROOT CNTRL-DATA3-(AFCTR,BFCTR,C)
AFCTR:      .FCTR A0-(A1,A2-(A21,A22))
BFCTR:      .FCTR B0-(B1,B2)
            .END
```

The attributes RW,D,GBL,REL and OVR are described in Table 6–1.

## 7.1.5 Multiple Tree Structures

The Task Builder allows the specification of more than one tree within the overlay structure. A structure containing multiple trees has the following properties:

1 Storage is not shared among trees. The total storage required is the sum of the longest path on each tree.

2 Each path in a tree is common to all paths on every other tree.

These properties allow modules, that would otherwise have to reside in the root segment, to be contained in an overlay tree.

Such overlay trees within the structure consist of a main tree and one or more co-trees. The root segment of the main tree is loaded by the Executive when the task is made active, while segments within each co-tree are loaded through calls to the overlay runtime system. Except for this distinction, all overlay trees have identical characteristics. That is, each tree must have a root segment and possibly one or more overlay segments.

The following sections describe the procedure for specifying multiple trees in the overlay description language and illustrate the use of co-trees to reduce the memory required by a task.

### Defining a Multiple Tree Structure

Multiple tree structures are specified within the ODL by extending the function of the comma (,) operator. As previously described, this operator, when included within parentheses, defines a pair of segments that share storage. The inclusion of the comma operator outside all parentheses delimits overlay trees. The first overlay tree thus defined is the main tree. Subsequent trees are co-trees.

Consider the following:

```
        .ROOT   X,Y
X:      .FCTR   X0-(X1,X2,X3)
Y:      .FCTR   Y0-(Y1,Y2)
        .END
```

Two overlay trees are specified. A main tree containing the root segment X0 and three overlay segments and a co-tree consisting of root segment Y0 and two overlay segments. The Executive loads segment X0 into memory when the task is activated. The task then loads the remaining segments through calls to the overlay runtime system.

A co-tree must have a root segment to establish linkages to the overlay segments within the co-tree. Logically, these root segments need not contain code or data. (Such modules can be resident in the main root). A segment of this type termed a 'null segment', may be created by means of the .NAME directive. The previous example is modified as shown below to include a null segment.

```
        .ROOT   X,Y
X:      .FCTR   X0-Y0-(X1,X2,X3)
        .NAME   YNUL
Y:      .FCTR   YNUL-(Y1,Y2)
        .END
```

The null segment 'YNUL' is created, using the .NAME directive, and replaces the co-tree root that formerly contained Y0.OBJ. Y0 now resides in the main root.

### Multiple Tree Example

The following example illustrates the use of multiple trees to reduce the size of the task.

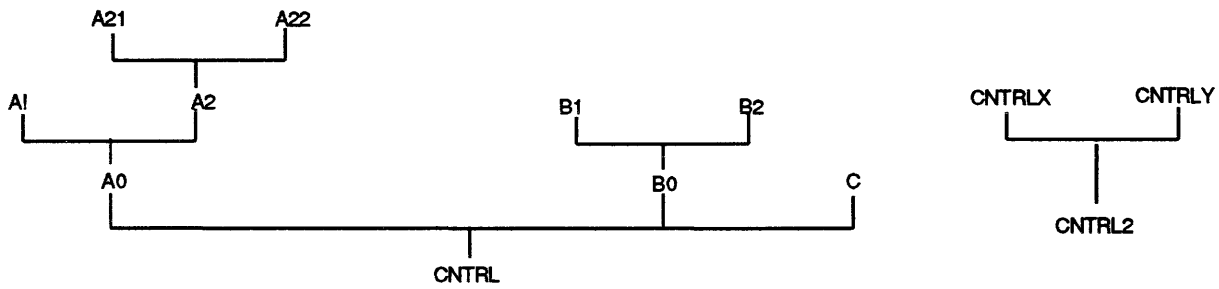
Suppose that in the task TK1 the root segment CNTRL consists of a small dispatching routine and two long modules, CNTRLX and CNTRLY. CNTRLX and CNTRLY are logically independent of each other, are approximately equal in length, and must access modules on all the paths of the main tree.

You can define a co-tree for CNTRLX and CNTRLY and effect a saving in the storage required for the task. You modify the overlay description in file TFIL.ODL as follows:

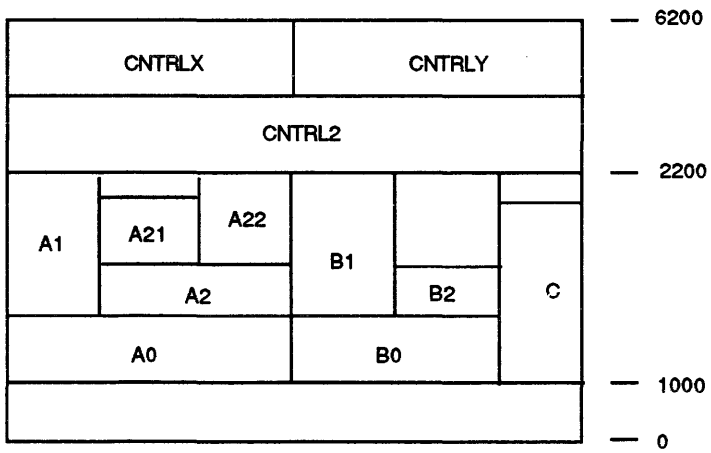
```
.NAME CNTRL2
.ROOT CNTRL-(AFCTR,BFCTR,C),CNTRL2-(CNTRLX,CNTRLY)
...
.END
```

The co-tree is defined at zero parenthesis level in the .ROOT directive. A co-tree must have a root segment, to establish links to the overlay segments within the co-tree. When no code or data logically belong in the root, the .NAME directive can be used to create a null root segment.

The tree for the task TK1 now is:




The corresponding memory diagram is:




The specification of the co-tree decreases the storage allocation by 4,000 bytes. CNTRLX and CNTRLY can still access modules on all the paths of the main tree. The only requirement imposed by the introduction of the co-tree is the logical independence of CNTRLX and CNTRLY.

Any number of co-trees can be defined. Additional co-trees can access all the modules in the main tree and in the other co-trees.

## 7.1.6 Overlay Core Image

The contents of the core image for a task with an overlay structure are described briefly in the following paragraphs.

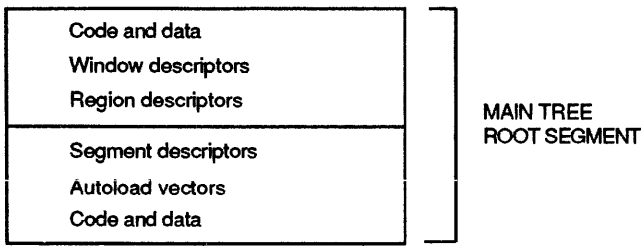
The root segment of the main tree contains modules that are resident in memory throughout the execution of the task, along with the following data required by the overlay loading routines.

1 Segment tables

2 Autoload vectors

3 Window descriptors

4 Region descriptors



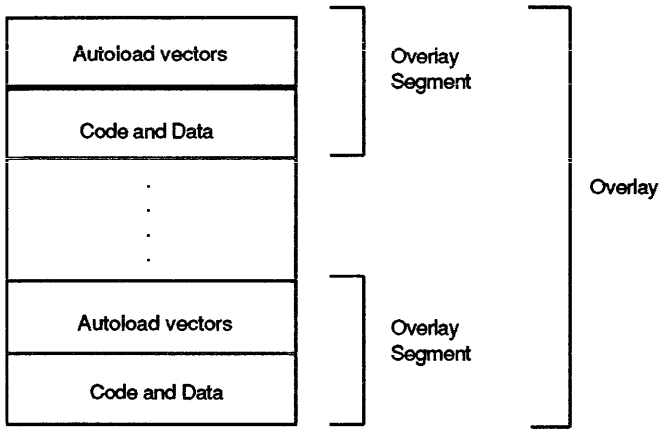

The segment table contains a segment descriptor for every segment in the task. The descriptor contains information about the load address, the length of the segment, and the tree linkage.

Autoload vectors appear in every segment that calls modules in another segment located farther away from the root of the tree.

Window descriptors are allocated whenever a memory-resident overlay structure is defined for the task. The descriptor contains information required by the Create Address Window system directive (CRAWS). One descriptor is allocated for each memory-resident overlay segment. For further information on IAS System directives, see the IAS System Directives Reference Manual.

Region descriptors are allocated whenever a task is linked to a shared region containing memory-resident overlays. The descriptor contains information required by the Attach Region system directive (ATRG$).

The main tree overlay region consists of memory allocated for the overlay segments of the main tree. The overlays are read into this area of memory as they are needed.




The co-tree overlay region consists of memory allocated for co-tree overlay segments.

The co-tree root segment contains modules that, once loaded, must remain resident in memory.

### 7.1.7 Overlaying Programs Written in a High-level Language

Programs written in a higher-level language usually require a large number of library routines in order to execute. Unless care is taken when overlaying such programs, the following problems can occur:

1 Task Builder speed can be drastically reduced because of the number of library references in each overlay segment.

2 Library references from the default object module library, that are resolved across tree boundaries, can result in unintentional displacement of segments from memory at run-time.

3 Attempts to task build such programs can result in multiple and ambiguous symbol definitions when a co-tree structure is defined.

The following procedure is effective in solving these problems:

1 Task Builder speed can be increased by linking commonly used library routines into the main root segment.

2 Ambiguous and multiple definitions and cross-tree references can be eliminated by using the /NOFULL_SEARCH PDS qualifier (/-FU MCR switch) to restrict the scope of the default library search.

With the default PDS qualifier /NOFULL_SEARCH (MCR switch/-FU), when a reference to a symbol is found in a co-tree, only the root segment of the main tree and of other co-trees is searched. The full search forces the Task Builder to search all segments of all trees to resolve global symbol references.

If sufficient mapping registers are available, the object time system can, in effect, be placed in the root segment by building a memory-resident library as described in Section 6.1.2. This also reduces total system memory requirements if other tasks are currently using the library.

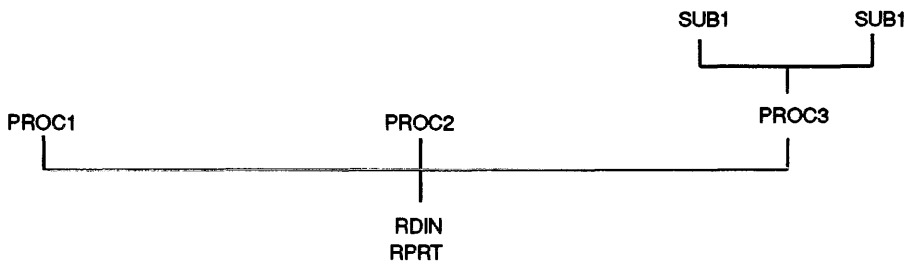
If a memory-resident library cannot be built, the user can force library modules into the root by preparing a list of the appropriate global references, and linking an object module derived from the list into the root segment.

For other ways to reduce task size consult the appropriate language user's guide.

## 7.2 EXAMPLE: CALC.TSK;3

The version of CALC introduced earlier is now ready for the addition of two more data processing routines, PROC2.OBJ and PROC3.OBJ. These new algorithms are logically independent of each other and of PROC1.OBJ. The third algorithm, PROC3.OBJ, contains two independent routines SUB1.OBJ and SUB2.OBJ.

You define an overlay structure for CALC as follows:




### 7.2.1 Creating the ODL File

You construct a file, CALTR.ODL, of ODL directives to represent the tree for CALC, as follows:

```
PDS> EDIT
FILE? CALTR.ODL
[EDI -- CREATING NEW FILE]
INPUT
                .ROOT RDIN-RPRT-*(PROC1,PROC2,P3FCTR)
        P3FCTR: .FCTR PROC3-(SUB1,SUB2)
                .END

 *EX
```

The "*" in the ODL description is the autoload indicator and is described in Chapter 8, Section 8.1.1.

### 7.2.2 Building the Task

You build the task with the same options as in the example of Chapter 5, Section 5.7.2. The names of the input files are replaced by a single filename that designates the file containing the overlay description:

```
PDS> LINK/TASK:CALC.TSK;3/MAP:(/SHORT)/OPTIONS-
/OVERLAY_DESCRIPTION:CALTR
OPTIONS? PAR=GEN
OPTIONS? ACTFIL=1
OPTIONS? /
```

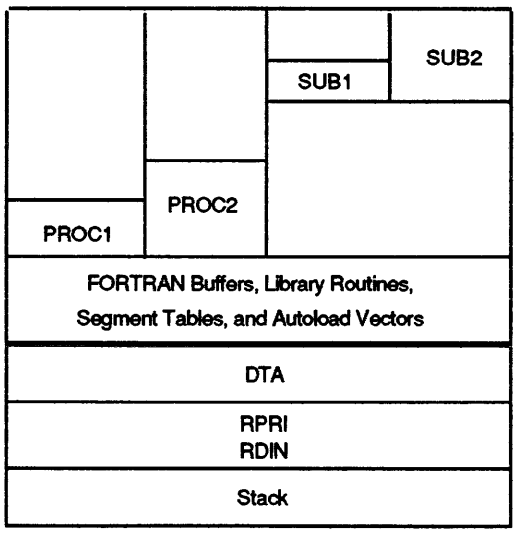
or

```
TKB>CALC;3,CALC=CALTR/MP
ENTER OPTIONS:
TKB>PAR = GEN
TKB>ACTFIL =1
TKB>/
```

## 7.2.3 Memory Allocation File for CALC.TSK;3

The short memory allocation file for this multi-segment task consists of one page per segment. For convenience the pages are compressed in this manual. See Figure 7-2.

The memory diagram for CALC.TSK;3 is:



**Example 7–1 Memory Allocation File for CALC.TSK;3**

```
CALC.TSK;3    MEMORY ALLOCATION MAP  TKB D28          PAGE 1
                      3-JUL-78   10:50


IDENTIFICATION : FORV02
STACK    LIMITS: 000000 000777 001000 00512.
PRG XFR ADDRESS: 017772
TOTAL ATTACHMENT DESCRIPTORS: 3.
TASK  IMAGE  SIZE  : 8544. WORDS
TASK HEADER SIZE: 160. WORDS
TASK ADDRESS LIMITS: 000000 041347
R-W DISK BLK LIMITS: 000003 000056 000054 00044.


CALC.TSK;3 OVERLAY DESCRIPTION:

BASE    TOP         LENGTH
----    ---         ------
000000  033143  033144  13924.     ROOT
033144  035767  002624  01428.       PROC1
033144  037727  004564  02420.       PROC2
033144  040307  005144  02660.       PROC3
040310  041157  000650  00424.         SUB1
040310  041347  001040  00544.         SUB2
```

**Example 7-1 Memory Allocation File for CALC.TSK;3 (continued)**

```
CALC.TSK;3   MEMORY ALLOCATION MAP  TKB D28          PAGE 2
RDIN                  3-JUL-78   10:50



*** ROOT SEGMENT: RDIN


R/W MEM  LIMITS: 000000 033143 033144 13924.
DISK BLK LIMITS: 000003 000036 000034 00028.


MEMORY ALLOCATION SYNOPSIS:

SECTION                                          TITLE  IDENT  FILE
-------                                          -----  -----  ----
. BLK.:(RW,I,LCL,REL,CON)  001000 000002 00002.
DTA   :(RW,D,GBL,REL,OVR)  001002 001442 00802.
                           001002 001442 00802. .MAIN. FORV02 RDIN.OBJ;2
                           001002 001442 00802. RPRT   FORV02 RPRT.OBJ;1
OTS$I :(RW,I,LCL,REL,CON)  002444 015270 06840.
                           002444 000000 00000. .MAIN. FORV02 RDIN.OBJ;2
OTS$P :(RW,D,GBL,REL,OVR)  017734 000036 00030.
$CODE :(RW,I,LCL,REL,CON)  017772 000132 00090.
                           017772 000000 00000. .MAIN. FORV02 RDIN.OBJ;2
                           017772 000000 00000. .MAIN. FORV02 RDIN.OBJ;2
                           017772 000116 00078. .MAIN. FORV02 RDIN.OBJ;2
                           020110 000000 00000. RPRT   FORV02 RPRT.OBJ;1
                           020110 000000 00000. RPRT   FORV02 RPRT.OBJ;1
                           020110 000014 00012. RPRT   FORV02 RPRT.OBJ;1
$DATA :(RW,D,LCL,REL,CON)  020124 003720 02000.
                           020124 000000 00000. .MAIN. FORV02 RDIN.OBJ;2
                           020124 001750 01000. .MAIN. FORV02 RDIN.OBJ;2
                           022074 000000 00000. RPRT   FORV02 RPRT.OBJ;1
                           022074 001750 01000. RPRT   FORV02 RPRT.OBJ;1
$DATAP:(RW,D,LCL,REL,CON)  024044 000032 00026.
                           024044 000000 00000. .MAIN. FORV02 RDIN.OBJ;2
                           024044 000022 00018. .MAIN. FORV02 RDIN.OBJ;2
                           024066 000000 00000. RPRT   FORV02 RPRT.OBJ;1
                           024066 000010 00008. RPRT   FORV02 RPRT.OBJ;1
$$ALER:(RW,I,LCL,REL,CON)  024076 000024 00020.
$$ALVC:(RW,D,LCL,REL,CON)  024122 000030 00024.
$$AOTS:(RW,D,LCL,REL,CON)  024152 000704 00452.
$$AUTO:(RW,I,LCL,REL,CON)  160000 000130 00088.
$$DEVT:(RW,D,LCL,REL,OVR)  025056 001210 00648.
$$FSR1:(RW,D,GBL,REL,OVR)  026266 004100 02112.
$$FSR2:(RW,D,GBL,REL,CON)  032366 000104 00068.
$$IOB1:(RW,D,LCL,REL,OVR)  032472 000204 00132.
$$IOB2:(RW,D,LCL,REL,OVR)  032676 000000 00000.
$$LOAD:(RW,I,LCL,REL,CON)  160130 000170 00120.
$$MRKS:(RW,I,LCL,REL,OVR)  160320 000166 00118.
$$OBF1:(RW,D,LCL,REL,CON)  032676 000110 00072.
$$OBF2:(RW,I,LCL,REL,CON)  033006 000000 00000.
$$OVDT:(RW,D,LCL,REL,OVR)  033006 000020 00016.
$$OVRS:(RW,I,LCL,ABS,CON)  000000 000000 00000.
$$RDSG:(RW,I,LCL,REL,OVR)  160506 000312 00202.
$$RESL:(RW,I,LCL,REL,CON)  161020 016216 07310.
$$RGDS:(RW,D,LCL,REL,CON)  033026 000000 00000.
$$RTS :(RW,I,GBL,REL,OVR)  033026 000002 00002.
```

**Example 7-1 Memory Allocation File for CALC.TSK;3 (continued)**

```
CALC.TSK;3   MEMORY ALLOCATION MAP  TKB D28          PAGE 3
RDIN                  3-JUL-78   10:50


$$SGD0:(RW,D,LCL,REL,OVR) 033030 000000 00000.
$$SGD1:(RW,D,LCL,REL,CON) 033030 000110 00072.
$$SGD2:(RW,D,LCL,REL,OVR) 033140 000002 00002.
$$WNDS:(RW,D,LCL,REL,CON) 033142 000000 00000.
.$$$$.:(RW,D,GBL,REL,OVR) 033142 000000 00000.
                          033142 000000 00000. .MAIN. FORV02 RDIN.OBJ;2
                          033142 000000 00000. .MAIN. FORV02 RDIN.OBJ;2
                          033142 000000 00000. RPRT   FORV02 RPRT.OBJ;1
                          033142 000000 00000. RPRT   FORV02 RPRT.OBJ;1


GLOBAL SYMBOLS:

PROC1  024122-R  PROC3  024142-R  $RF2A1 000000-R  $$OTSI 002444-R
PROC2  024132-R  RPRT   020110-R  $$OTSC 017772-R


CALC.TSK;3   MEMORY ALLOCATION MAP  TKB D28          PAGE 4
PROC1                 3-JUL-78   10:50


*** SEGMENT: PROC1


R/W MEM  LIMITS: 033144 035767 002624 01428.
DISK BLK LIMITS: 000037 000041 000003 00003.


MEMORY ALLOCATION SYNOPSIS:

SECTION                                        TITLE  IDENT  FILE
-------                                        -----  -----  ----
. BLK.:(RW,I,LCL,REL,CON) 033144 000000 00000.
ADTA  :(RW,D,GBL,REL,OVR) 033144 002260 01200.
                          033144 002260 01200. PROC1  FORV02 PROC1.OBJ;2
DTA   :(RW,D,GBL,REL,OVR) 001002 001442 00802.
                          001002 001442 00802. PROC1  FORV02 PROC1.OBJ;2
OTSSI :(RW,I,LCL,REL,CON) 035424 000254 00172.
$CODE :(RW,I,LCL,REL,CON) 035700 000054 00044.
                          035700 000000 00000. PROC1  FORV02 PROC1.OBJ;2
                          035700 000000 00000. PROC1  FORV02 PROC1.OBJ;2
                          035700 000054 00044. PROC1  FORV02 PROC1.OBJ;2
$DATA :(RW,D,LCL,REL,CON) 035754 000002 00002.
                          035754 000000 00000. PROC1  FORV02 PROC1.OBJ;2
                          035754 000002 00002. PROC1  FORV02 PROC1.OBJ;2
$DATAP:(RW,D,LCL,REL,CON) 035756 000010 00008.
                          035756 000000 00000. PROC1  FORV02 PROC1.OBJ;2
                          035756 000010 00008. PROC1  FORV02 PROC1.OBJ;2
$$ALVC:(RW,D,LCL,REL,CON) 035766 000000 00000.
.$$$$.:(RW,D,GBL,REL,OVR) 033142 000000 00000.
                          033142 000000 00000. PROC1  FORV02 PROC1.OBJ;2
                          033142 000000 00000. PROC1  FORV02 PROC1.OBJ;2

GLOBAL SYMBOLS:

PROC1  035700-R
```

**Example 7-1 Memory Allocation File for CALC.TSK;3 (continued)**

```
CALC.TSK;3    MEMORY ALLOCATION MAP  TKB D28          PAGE 5
PROC2                 3-JUL-78   10:50


*** SEGMENT: PROC2


R/W MEM  LIMITS: 033144 037727 004564 02420.
DISK BLK LIMITS: 000042 000046 000005 00005.


MEMORY ALLOCATION SYNOPSIS:

SECTION                                          TITLE  IDENT  FILE
-------                                          -----  -----  ----
ADTA  :(RW,D,GBL,REL,OVR) 033144 002260 01200.
                          033144 002260 01200.  PROC2  FORV02 PROC2.OBJ;1
$CODE :(RW,I,LCL,REL,CON) 035424 000014 00012.
                          035424 000000 00000.  PROC2  FORV02 PROC2.OBJ;1
                          035424 000000 00000.  PROC2  FORV02 PROC2.OBJ;1
                          035424 000014 00012.  PROC2  FORV02 PROC2.OBJ;1
$DATA :(RW,D,LCL,REL,CON) 035440 002260 01200.
                          035440 000000 00000.  PROC2  FORV02 PROC2.OBJ;1
                          035440 002260 01200.  PROC2  FORV02 PROC2.OBJ;1
$DATAP:(RW,D,LCL,REL,CON) 037720 000010 00008.
                          037720 000000 00000.  PROC2  FORV02 PROC2.OBJ;1
                          037720 000010 00008.  PROC2  FORV02 PROC2.OBJ;1
$$ALVC:(RW,D,LCL,REL,CON) 037730 000000 00000.
.$$$$.:(RW,D,GBL,REL,OVR) 033142 000000 00000.
                          033142 000000 00000.  PROC2  FORV02 PROC2.OBJ;1
                          033142 000000 00000.  PROC2  FORV02 PROC2.OBJ;1


GLOBAL SYMBOLS:

PROC2  035424-R
```

**Example 7-1 Memory Allocation File for CALC.TSK;3 (continued)**

```
CALC.TSK;3    MEMORY ALLOCATION MAP  TKB D28          PAGE 6
PROC3                  3-JUL-78   10:50


*** SEGMENT: PROC3


R/W MEM  LIMITS: 033144 040307 005144 02660.
DISK BLK LIMITS: 000047 000054 000006 00006.


MEMORY ALLOCATION SYNOPSIS:

SECTION                                          TITLE  IDENT  FILE
-------                                          -----  -----  ----
ADTA  :(RW,D,GBL,REL,OVR) 033144 002260 01200.
                          033144 002260 01200. PROC3  FORV02 PROC3.OBJ;1
DTA   :(RW,D,GBL,REL,OVR) 001002 001442 00802.
                          001002 001442 00802. PROC3  FORV02 PROC3.OBJ;1
$CODE :(RW,I,LCL,REL,CON) 035424 000044 00036.
                          035424 000000 00000. PROC3  FORV02 PROC3.OBJ;1
                          035424 000000 00000. PROC3  FORV02 PROC3.OBJ;1
                          035424 000044 00036. PROC3  FORV02 PROC3.OBJ;1
$DATA :(RW,D,LCL,REL,CON) 035470 002570 01400.
                          035470 000000 00000. PROC3  FORV02 PROC3.OBJ;1
                          035470 002570 01400. PROC3  FORV02 PROC3.OBJ;1
$DATAP:(RW,D,LCL,REL,CON) 040260 000010 00008.
                          040260 000000 00000. PROC3  FORV02 PROC3.OBJ;1
                          040260 000010 00008. PROC3  FORV02 PROC3.OBJ;1
$$ALVC:(RW,D,LCL,REL,CON) 040270 000020 00016.
.$$$$.:(RW,D,GBL,REL,OVR) 033142 000000 00000.
                          033142 000000 00000. PROC3  FORV02 PROC3.OBJ;1
                          033142 000000 00000. PROC3  FORV02 PROC3.OBJ;1


GLOBAL SYMBOLS:

PROC3  035424-R  SUB1   040300-R  SUB2   040270-R
```

**Example 7-1 Memory Allocation File for CALC.TSK;3 (continued)**

```
CALC.TSK;3    MEMORY ALLOCATION MAP  TKB D28          PAGE 7
SUB1                    3-JUL-78   10:50


*** SEGMENT: SUB1


R/W MEM  LIMITS: 040310 041157 000650 00424.
DISK BLK LIMITS: 000055 000055 000001 00001.


MEMORY ALLOCATION SYNOPSIS:

SECTION                                          TITLE  IDENT  FILE
-------                                          -----  -----  ----
ADTA  :(RW,D,GBL,REL,OVR)  033144 002260 01200.
                           033144 002260 01200.  SUB2   FORV02 SUB1.OBJ;1
$CODE :(RW,I,LCL,REL,CON)  040310 000014 00012.
                           040310 000000 00000.  SUB2   FORV02 SUB1.OBJ;1
                           040310 000000 00000.  SUB2   FORV02 SUB1.OBJ;1
                           040310 000014 00012.  SUB2   FORV02 SUB1.OBJ;1
$DATA :(RW,D,LCL,REL,CON)  040324 000624 00404.
                           040324 000000 00000.  SUB2   FORV02 SUB1.OBJ;1
                           040324 000624 00404.  SUB2   FORV02 SUB1.OBJ;1
$DATAP:(RW,D,LCL,REL,CON)  041150 000010 00008.
                           041150 000000 00000.  SUB2   FORV02 SUB1.OBJ;1
                           041150 000010 00008.  SUB2   FORV02 SUB1.OBJ;1
$$ALVC:(RW,D,LCL,REL,CON)  041160 000000 00000.
.$$$$.:(RW,D,GBL,REL,OVR)  033142 000000 00000.
                           033142 000000 00000.  SUB2   FORV02 SUB1.OBJ;1
                           033142 000000 00000.  SUB2   FORV02 SUB1.OBJ;1


GLOBAL SYMBOLS:

SUB2   040310-R
```

**Example 7-1 Memory Allocation File for CALC.TSK;3 (continued)**

```
CALC.TSK;3   MEMORY ALLOCATION MAP  TKB D28          PAGE 8
SUB2                  3-JUL-78   10:50


*** SEGMENT: SUB2


R/W MEM  LIMITS: 040310 041347 001040 00544.
DISK BLK LIMITS: 000056 000057 000002 00002.


MEMORY ALLOCATION SYNOPSIS:

SECTION                                          TITLE  IDENT  FILE
-------                                          -----  -----  ----
. BLK.:(RW,I,LCL,REL,CON) 040310 000000 00000.
ADTA  :(RW,D,GBL,REL,OVR) 033144 002260 01200.
                          033144 002260 01200. SUB1   FORV02 SUB2.OBJ;1
DTA   :(RW,D,GBL,REL,OVR) 001002 001442 00802.
                          001002 001442 00802. SUB1   FORV02 SUB2.OBJ;1
OTS$I :(RW,I,LCL,REL,CON) 040310 000154 00108.
$CODE :(RW,I,LCL,REL,CON) 040464 000032 00026.
                          040464 000000 00000. SUB1   FORV02 SUB2.OBJ;1
                          040464 000000 00000. SUB1   FORV02 SUB2.OBJ;1
                          040464 000032 00026. SUB1   FORV02 SUB2.OBJ;1
$DATA :(RW,D,LCL,REL,CON) 040516 000622 00402.
                          040516 000000 00000. SUB1   FORV02 SUB2.OBJ;1
                          040516 000622 00402. SUB1   FORV02 SUB2.OBJ;1
$DATAP:(RW,D,LCL,REL,CON) 041340 000010 00008.
                          041340 000000 00000. SUB1   FORV02 SUB2.OBJ;1
                          041340 000010 00008. SUB1   FORV02 SUB2.OBJ;1
$$ALVC:(RW,D,LCL,REL,CON) 041350 000000 00000.
.$$$$.:(RW,D,GBL,REL.OVR) 033142 000000 00000.
                          033142 000000 00000. SUB1   FORV02 SUB2.OBJ;1
                          033142 000000 00000. SUB1   FORV02 SUB2.OBJ;1


GLOBAL SYMBOLS:

SUB1   040464-R


*** TASK BUILDER STATISTICS:

    TOTAL WORK FILE REFERENCES: 21731.
    WORK  FILE  READS: 0.
    WORK  FILE WRITES: 0.
    SIZE OF CORE POOL: 16010. WORDS (62. PAGES)
    SIZE OF WORK FILE: 7680. WORDS (30. PAGES)

    ELAPSED TIME:00:00:21
```

## 7.3 EXAMPLE CALC.TSK;4

After examining the memory allocation file for CALC.TSK;3, you observe that the Task Builder has allocated ADTA in the overlay segments PROC1.OBJ, PROC2.OBJ, and PROC3.OBJ, since all of these segments are equidistant from the root.

These segments need to communicate with each other through ADTA. In the existing allocation, any values placed in ADTA by PROC1.OBJ are lost when PROC2.OBJ is loaded. Similarly, any values stored in ADTA by PROC2.OBJ are lost when PROC3.OBJ is loaded.

A .PSECT directive is added to the overlay description to force ADTA into the root segment. PROC1.OBJ, PROC2.OBJ, and PROC3.OBJ can then communicate with each other. CALTR.ODL needs to be modified as follows:

```
            .ROOT RDIN-RPRT-ADTA-*(PROC1,PROC2,P3FCTR)
P3FCTR:     .FCTR PROC3-(SUB1,SUB2)
            .PSECT ADTA,RW,D,GBL,REL,OVR
            .END
```

The task is built as in CALC.TSK;3.

**Example 7–2 Memory Allocation File for CALC.TSK;4**

```
CALC.TSK;4    MEMORY ALLOCATION MAP  TKB D28          PAGE 1
                      3-JUL-78   10:51


IDENTIFICATION : FORV02
STACK    LIMITS: 000000 000777 001000 00512.
PRG XFR ADDRESS: 022252
TOTAL ATTACHMENT DESCRIPTORS: 3.
TASK  IMAGE  SIZE  : 8544. WORDS
TASK HEADER SIZE: 160. WORDS
TASK ADDRESS LIMITS: 000000 041347
R-W DISK BLK LIMITS: 000003 000052 000050 00040.


CALC.TSK;4 OVERLAY DESCRIPTION:

BASE    TOP         LENGTH
----    ---         ------
000000  035423  035424  15124.    RDIN
035424  035767  000344  00228.      PROC1
035424  037727  002304  01220.      PROC2
035424  040307  002664  01460.      PROC3
040310  041157  000650  00424.        SUB1
040310  041347  001040  00544.        SUB2
```

**Example 7-2 Memory Allocation File for CALC.TSK;4 (continued)**

```
CALC.TSK;4    MEMORY ALLOCATION MAP  TKB D28          PAGE 2
RDIN                    3-JUL-78   10:51



*** ROOT SEGMENT: RDIN


R/W MEM  LIMITS: 000000 035423 035424 15124.
DISK BLK LIMITS: 000003 000040 000036 00030.


MEMORY ALLOCATION SYNOPSIS:

SECTION                                          TITLE  IDENT  FILE
-------                                          -----  -----  ----
. BLK.:(RW,I,LCL,REL,CON) 001000 000002 00002.
ADTA  :(RW,D,GBL,REL,OVR) 001002 002260 01200.
DTA   :(RW,D,GBL,REL,OVR) 003262 001442 00802.
                          003262 001442 00802. .MAIN. FORV02 RDIN.OBJ;2
                          003262 001442 00802. RPRT   FORV02 RPRT.OBJ;1
OTS$I :(RW,I,LCL,REL,CON) 004724 015270 06840.
                          004724 000000 00000. .MAIN. FORV02 RDIN.OBJ;2
OTS$P :(RW,D,GBL,REL,OVR) 022214 000036 00030.
$CODE :(RW,I,LCL,REL,CON) 022252 000132 00090.
                          022252 000000 00000. .MAIN. FORV02 RDIN.OBJ;2
                          022252 000000 00000. .MAIN. FORV02 RDIN.OBJ;2
                          022252 000116 00078. .MAIN. FORV02 RDIN.OBJ;2
                          022370 000000 00000. RPRT   FORV02 RPRT.OBJ;1
                          022370 000000 00000. RPRT   FORV02 RPRT.OBJ;1
                          022370 000014 00012. RPRT   FORV02 RPRT.OBJ;1
$DATA :(RW,D,LCL,REL,CON) 022404 003720 02000.
                          022404 000000 00000. .MAIN. FORV02 RDIN.OBJ;2
                          022404 001750 01000. .MAIN. FORV02 RDIN.OBJ;2
                          024354 000000 00000. RPRT   FORV02 RPRT.OBJ;1
                          024354 001750 01000. RPRT   FORV02 RPRT.OBJ;1
$DATAP:(RW,D,LCL,REL,CON) 026324 000032 00026.
                          026324 000000 00000. .MAIN. FORV02 RDIN.OBJ;2
                          026324 000022 00018. .MAIN. FORV02 RDIN.OBJ;2
                          026346 000000 00000. RPRT   FORV02 RPRT.OBJ;1
                          026346 000010 00008. RPRT   FORV02 RPRT.OBJ;1
$$ALER:(RW,I,LCL,REL,CON) 026356 000024 00020.
$$ALVC:(RW,D,LCL,REL,CON) 026402 000030 00024.
$$AOTS:(RW,D,LCL,REL,CON) 026432 000704 00452.
$$AUTO:(RW,I,LCL,REL,CON) 160000 000130 00088.
$$DEVT:(RW,D,LCL,REL,OVR) 027336 001210 00648.
$$FSR1:(RW,D,GBL,REL,OVR) 030546 004100 02112.
$$FSR2:(RW,D,GBL,REL,CON) 034646 000104 00068.
$$IOB1:(RW,D,LCL,REL,OVR) 034752 000204 00132.
$$IOB2:(RW,D,LCL,REL,OVR) 035156 000000 00000.
$$LOAD:(RW,I,LCL,REL,CON) 160130 000170 00120.
$$MRKS:(RW,I,LCL,REL,OVR) 160320 000166 00118.
$$OBF1:(RW,D,LCL,REL,CON) 035156 000110 00072.
$$OBF2:(RW,I,LCL,REL,CON) 035266 000000 00000.
$$OVDT:(RW,D,LCL,REL,OVR) 035266 000020 00016.
$$OVRS:(RW,I,LCL,ABS,CON) 000000 000000 00000.
$$RDSG:(RW,I,LCL,REL,OVR) 160506 000312 00202.
$$RESL:(RW,I,LCL,REL,CON) 161020 016216 07310.
$$RGDS:(RW,D,LCL,REL,CON) 035306 000000 00000.
```

**Example 7-2 Memory Allocation File for CALC.TSK;4 (continued)**

```
CALC.TSK;4   MEMORY ALLOCATION MAP  TKB D28          PAGE 3
RDIN                  3-JUL-78   10:51


$$RTS :(RW,I,GBL,REL,OVR) 035306 000002 00002.
$$SGD0:(RW,D,LCL,REL,OVR) 035310 000000 00000.
$$SGD1:(RW,D,LCL,REL,CON) 035310 000110 00072.
$$SGD2:(RW,D,LCL,REL,OVR) 035420 000002 00002.
$$WNDS:(RW,D,LCL,REL,CON) 035422 000000 00000.
.$$$$.:(RW,D,GBL,REL,OVR) 035422 000000 00000.
                          035422 000000 00000. .MAIN. FORV02 RDIN.OBJ;2
                          035422 000000 00000. .MAIN. FORV02 RDIN.OBJ;2
                          035422 000000 00000. RPRT   FORV02 RPRT.OBJ;1
                          035422 000000 00000. RPRT   FORV02 RPRT.OBJ;1


GLOBAL SYMBOLS:

PROC1  026402-R  PROC3  026422-R  $RF2A1 000000-R  $$OTSI 004724-R
PROC2  026412-R  RPRT   022370-R  $$OTSC 022252-R

CALC.TSK;4   MEMORY ALLOCATION MAP  TKB D28          PAGE 4
PROC1                 3-JUL-78   10:51


*** SEGMENT: PROC1


R/W MEM  LIMITS: 035424 035767 000344 00228.
DISK BLK LIMITS: 000042 000042 000001 00001.


MEMORY ALLOCATION SYNOPSIS:
SECTION                                          TITLE  IDENT  FILE
-------                                          -----  -----  ----
. BLK.:(RW,I,LCL,REL,CON) 035424 000000 00000.
ADTA  :(RW,D,GBL,REL,OVR) 001002 002260 01200.
                          001002 002260 01200. PROC1  FORV02 PROC1.OBJ;2
DTA   :(RW,D,GBL,REL,OVR) 003262 001442 00802.
                          003262 001442 00802. PROC1  FORV02 PROC1.OBJ;2
OTS$I :(RW,I,LCL,REL,CON) 035424 000254 00172.
$CODE :(RW,I,LCL,REL,CON) 035700 000054 00044.
                          035700 000000 00000. PROC1  FORV02 PROC1.OBJ;2
                          035700 000000 00000. PROC1  FORV02 PROC1.OBJ;2
                          035700 000054 00044. PROC1  FORV02 PROC1.OBJ;2
$DATA :(RW,D,LCL,REL,CON) 035754 000002 00002.
                          035754 000000 00000. PROC1  FORV02 PROC1.OBJ;2
                          035754 000002 00002. PROC1  FORV02 PROC1.OBJ;2
$DATAP:(RW,D,LCL,REL,CON) 035756 000010 00008.
                          035756 000000 00000. PROC1  FORV02 PROC1.OBJ;2
                          035756 000010 00008. PROC1  FORV02 PROC1.OBJ;2
$$ALVC:(RW,D,LCL,REL,CON) 035766 000000 00000.
.$$$$.:(RW,D,GBL,REL,OVR) 035422 000000 00000.
                          035422 000000 00000. PROC1  FORV02 PROC1.OBJ;2
                          035422 000000 00000. PROC1  FORV02 PROC1.OBJ;2


GLOBAL SYMBOLS:

PROC1  035700-R
```

**Example 7-2 Memory Allocation File for CALC.TSK;4 (continued)**

```
CALC.TSK;4   MEMORY ALLOCATION MAP  TKB D28          PAGE 5
PROC2                3-JUL-78   10:51


*** SEGMENT: PROC2


R/W MEM  LIMITS: 035424 037727 002304 01220.
DISK BLK LIMITS: 000043 000045 000003 00003.


MEMORY ALLOCATION SYNOPSIS:

SECTION                                          TITLE  IDENT  FILE
-------                                          -----  -----  ----
ADTA  :(RW,D,GBL,REL,OVR) 001002 002260 01200.
                          001002 002260 01200. PROC2  FORV02 PROC2.OBJ;1
$CODE :(RW,I,LCL,REL,CON) 035424 000014 00012.
                          035424 000000 00000. PROC2  FORV02 PROC2.OBJ;1
                          035424 000000 00000. PROC2  FORV02 PROC2.OBJ;1
                          035424 000014 00012. PROC2  FORV02 PROC2.OBJ;1
$DATA :(RW,D,LCL,REL,CON) 035440 002260 01200.
                          035440 000000 00000. PROC2  FORV02 PROC2.OBJ;1
                          035440 002260 01200. PROC2  FORV02 PROC2.OBJ;1
$DATAP:(RW,D,LCL,REL,CON) 037720 000010 00008.
                          037720 000000 00000. PROC2  FORV02 PROC2.OBJ;1
                          037720 000010 00008. PROC2  FORV02 PROC2.OBJ;1
$$ALVC:(RW,D,LCL,REL,CON) 037730 000000 00000.
.$$$$.:(RW,D,GBL,REL,OVR) 035422 000000 00000.
                          035422 000000 00000. PROC2  FORV02 PROC2.OBJ;1
                          035422 000000 00000. PROC2  FORV02 PROC2.OBJ;1


GLOBAL SYMBOLS:

PROC2  035424-R
```

**Example 7-2 Memory Allocation File for CALC.TSK;4 (continued)**

```
CALC.TSK;4    MEMORY ALLOCATION MAP  TKB D28          PAGE 6
PROC3                  3-JUL-78   10:51


*** SEGMENT: PROC3


R/W MEM  LIMITS: 035424 040307 002664 01460.
DISK BLK LIMITS: 000046 000050 000003 00003.


MEMORY ALLOCATION SYNOPSIS:

SECTION                                          TITLE  IDENT  FILE
-------                                          -----  -----  ----
ADTA  :(RW,D,GBL,REL,OVR) 001002 002260 01200.
                          001002 002260 01200.  PROC3  FORV02 PROC3.OBJ;1
DTA   :(RW,D,GBL,REL,OVR) 003262 001442 00802.
                          003262 001442 00802.  PROC3  FORV02 PROC3.OBJ;1
$CODE :(RW,I,LCL,REL,CON) 035424 000044 00036.
                          035424 000000 00000.  PROC3  FORV02 PROC3.OBJ;1
                          035424 000000 00000.  PROC3  FORV02 PROC3.OBJ;1
                          035424 000044 00036.  PROC3  FORV02 PROC3.OBJ;1
$DATA :(RW,D,LCL,REL,CON) 035470 002570 01400.
                          035470 000000 00000.  PROC3  FORV02 PROC3.OBJ;1
                          035470 002570 01400.  PROC3  FORV02 PROC3.OBJ;1
$DATAP:(RW,D,LCL,REL,CON) 040260 000010 00008.
                          040260 000000 00000.  PROC3  FORV02 PROC3.OBJ;1
                          040260 000010 00008.  PROC3  FORV02 PROC3.OBJ;1
$$ALVC:(RW,D,LCL,REL,CON) 040270 000020 00016.
.$$$$.:(RW,D,GBL,REL,OVR) 035422 000000 00000.
                          035422 000000 00000.  PROC3  FORV02 PROC3.OBJ;1
                          035422 000000 00000.  PROC3  FORV02 PROC3.OBJ;1


GLOBAL SYMBOLS:

PROC3  035424-R  SUB1   040300-R  SUB2   040270-R
```

**Example 7-2 Memory Allocation File for CALC.TSK;4 (continued)**

```
CALC.TSK;4    MEMORY ALLOCATION MAP  TKB D28          PAGE 7
SUB1                  3-JUL-78   10:51


*** SEGMENT: SUB1


R/W MEM  LIMITS: 040310 041157 000650 00424.
DISK BLK LIMITS: 000051 000051 000001 00001.


MEMORY ALLOCATION SYNOPSIS:

SECTION                                          TITLE  IDENT  FILE
-------                                          -----  -----  ----
ADTA  :(RW,D,GBL,REL,OVR) 001002 002260 01200.
                          001002 002260 01200. SUB2   FORV02 SUB1.OBJ;1
$CODE :(RW,I,LCL,REL,CON) 040310 000014 00012.
                          040310 000000 00000. SUB2   FORV02 SUB1.OBJ;1
                          040310 000000 00000. SUB2   FORV02 SUB1.OBJ;1
                          040310 000014 00012. SUB2   FORV02 SUB1.OBJ;1
$DATA :(RW,D,LCL,REL,CON) 040324 000624 00404.
                          040324 000000 00000. SUB2   FORV02 SUB1.OBJ;1
                          040324 000624 00404. SUB2   FORV02 SUB1.OBJ;1
$DATAP:(RW,D,LCL,REL,CON) 041150 000010 00008.
                          041150 000000 00000. SUB2   FORV02 SUB1.OBJ;1
                          041150 000010 00008. SUB2   FORV02 SUB1.OBJ;1
$$ALVC:(RW,D,LCL,REL,CON) 041160 000000 00000.
.$$$$.:(RW,D,GBL,REL,OVR) 035422 000000 00000.
                          035422 000000 00000. SUB2   FORV02 SUB1.OBJ;1
                          035422 000000 00000. SUB2   FORV02 SUB1.OBJ;1


GLOBAL SYMBOLS:

SUB2   040310-R
```

**Example 7-2 Memory Allocation File for CALC.TSK;4 (continued)**

```
CALC.TSK;4    MEMORY ALLOCATION MAP  TKB D28          PAGE 8
SUB2                    3-JUL-78   10:51


*** SEGMENT: SUB2


R/W MEM  LIMITS: 040310 041347 001040 00544.
DISK BLK LIMITS: 000052 000053 000002 00002.


MEMORY ALLOCATION SYNOPSIS:

SECTION                                          TITLE  IDENT  FILE
-------                                          -----  -----  ----
. BLK.:(RW,I,LCL,REL,CON) 040310 000000 00000.
ADTA  :(RW,D,GBL,REL,OVR) 001002 002260 01200.
                          001002 002260 01200. SUB1   FORV02 SUB2.OBJ;1
DTA   :(RW,D,GBL,REL,OVR) 003262 001442 00802.
                          003262 001442 00802. SUB1   FORV02 SUB2.OBJ;1
OTS$I :(RW,I,LCL,REL,CON) 040310 000154 00108.
$CODE :(RW,I,LCL,REL,CON) 040464 000032 00026.
                          040464 000000 00000. SUB1   FORV02 SUB2.OBJ;1
                          040464 000000 00000. SUB1   FORV02 SUB2.OBJ;1
                          040464 000032 00026. SUB1   FORV02 SUB2.OBJ;1
$DATA :(RW,D,LCL,REL,CON) 040516 000622 00402.
                          040516 000000 00000. SUB1   FORV02 SUB2.OBJ;1
                          040516 000622 00402. SUB1   FORV02 SUB2.OBJ;1
$DATAP:(RW,D,LCL,REL,CON) 041340 000010 00008.
                          041340 000000 00000. SUB1   FORV02 SUB2.OBJ;1
                          041340 000010 00008. SUB1   FORV02 SUB2.OBJ;1
$$ALVC:(RW,D,LCL,REL,CON) 041350 000000 00000.
.$$$$.:(RW,D,GBL,REL,OVR) 035422 000000 00000.
                          035422 000000 00000. SUB1   FORV02 SUB2.OBJ;1
                          035422 000000 00000. SUB1   FORV02 SUB2.OBJ;1


GLOBAL SYMBOLS:

SUB1    040464-R



*** TASK BUILDER STATISTICS:

    TOTAL WORK FILE REFERENCES: 22127.
    WORK  FILE  READS: 0.
    WORK  FILE WRITES: 0.
    SIZE OF CORE POOL: 16010. WORDS (62. PAGES)
    SIZE OF WORK FILE: 7680. WORDS (30. PAGES)

    ELAPSED TIME:00:00:20
```

## 7.4 Summary of the Overlay Description Language

1 An overlay structure consists of one or more trees. Each tree contains at least one segment. A segment is a set of modules and p-sections that can be loaded by a single disk access.

A tree can have only one root segment, but it can have any number of overlay segments.

2 The ODL provides five directives for specifying the tree representation of the overlay structure, namely:

```
.ROOT
.END
.PSECT
.FCTR
.NAME
```

These directives can appear in any order in the overlay description, subject to the following restrictions:

a. There can be only one .ROOT and one .END directive.

b. The .END directive must be the last directive, since it terminates input.

3 The tree structure is defined by the operators "-" (hyphen), "," (comma), "!" (exclamation mark) and by the use of parentheses.

- indicates that its arguments are to be concatenated and thus co-exist in memory.

, within parentheses, indicates that is arguments are to be overlaid and thus share memory. The parentheses group segments that begin at the same point in memory.

, not within parentheses, separates trees (main tree and each co-tree, see item 10 below).

! immediately before a left parenthesis indicates that the immediately enclosed segments are memory resident. Segments enclosed in further parentheses are not allowed.

For example,

```
.ROOT A-B-(C,D-(E,F))
```

defines an overlay structure with a root segment consisting of the modules A and B. In this structure, there are four overlay segments, C, D, E, and F. The outer parenthesis pair indicates that the overlay segments C and D start at the same location in memory.

4 The simplest overlay description consists of two directives, as follows:

```
.ROOT A-B-(C,D-(E,F))
.END
```

Any number of the optional directives (.FCTR, .PSECT, and .NAME) can be included.

5 The .ROOT directive defines the overlay structure. The arguments of the .ROOT directive are one or more of the following:

a. File specifications as described in Chapter 2, Section 2.4.1 (PDS) or Chapter 3, Section 3.3.1 (MCR)

b. Factor labels

c. Segment names

d. P-section names

6 The .END directive is required to terminate input.

7 The .FCTR directive provides a means for replacing text by a symbolic reference (the factor label). This replacement is useful for two reasons:

a. The .FCTR directive effectively extends the text of the .ROOT directive to more than one line and thus allows complex trees to be represented.

b. The .FCTR directive allows the overlay description to be written in a form that makes the structure of the tree more apparent.

For example:

```
.ROOT A-(B-(C,D),E-(F,G),H)
.END
```

can be expressed, using the .FCTR directive, as follows:

```
            .ROOT A-(F1,F2,H)
F1:         .FCTR B-(C,D)
F2:         .FCTR E-(F,G)
            .END
```

The second representation makes it clear that the tree has three main branches.

8 A .PSECT directive is required when a .ROOT or a .FCTR specifies the segment in which a p-section is placed.

The .PSECT directive gives the name of the p-section and its attributes. For example:

```
.PSECT ALPHA,CON,GBL,RW,I,REL
```

ALPHA is the p-section name and the remaining arguments are attributes. P-section attributes are described in Table 5-1. The p-section name must appear first on the .PSECT directive, but the attributes can appear in any order or can be omitted. If an attribute is omitted, a default assumption is made. For p-section attributes the default assumptions are:

```
RW,I,LCL,REL,CON
```

In the above example, therefore, it is only necessary to specify the attributes that do not correspond to the default assumption:

```
.PSECT ALPHA,GBL
```

9 The .NAME directive provides a means for defining a segment name for use in the overlay description and for specifying segment attributes. This directive is useful for creating a null segment or naming a segment that is to be loaded manually or naming a non-executable segment that is to be autoloadable. If the .NAME directive is not used, the name of the first file, or p-section in the segment is used to identify the segment.

The .NAME directive defines a name, as follows:

```
.NAME segname [,attr][,attr]
```

where:

- segname - is the defined name, composed from the character set A-Z, 0-9 and $.
- attr - is an optional attribute, taken from GBL, NODSK, NOGBL, DKS, NOPHY. Defaults: NOGBL, DSK.

The defined name must be unique with respect to the names of p-sections, segments, files, and factor labels.

10 A co-tree can be defined by specifying an additional tree structure in the .ROOT directive. The first overlay tree description in the .ROOT directive is the main tree. Subsequent overlay descriptions are co-trees. For example:

```
.ROOT A-B-(C,D-(E,F)),X-(Y,Z),Q-(R,S,T)
```

The main tree in this example has the root segment consisting of files A.OBJ and B.OBJ; two co-trees are defined; the first co-tree has the root segment X and the second co-tree has the root segment Q.

11 Qualifiers for file specifications in ODL files always use the MCR switch form. The list below indicates the form for each qualifier:

| PDS form | MCR form |
|---|---|
| /[NO]CONCATENATED | /[NO]CC |
| /LIBRARY | /LB |
| /LIBRARY:model[:...] | /LB:mod1[:...] |
| /[NO]MAP | /[NO]MAP |
| /SELECT | /SS |

12 Comments are prefixed by ";" (semicolon).

13 File specifications on a single ODL command line adopt the device and ufd specification defaults from the file specifications to their left, see the IAS MCR User's Guide.

# 8 LOADING MECHANISMS

There are two methods for loading both disk-resident and memory- resident overlays:

Autoload — in which the Overlay Runtime System is automatically called upon to load those segments that are marked by you, and

Manual Load — in which you include in the task explicit calls to the Overlay Runtime System.

You must decide which of these methods to use, because both cannot be used in the same task.

The loading process depends on the kind of overlay:

1 Disk-Resident - A segment is loaded from disk into a shared area of physical memory, writing over whatever was present.

2 Memory-Resident - A segment is made available by mapping a set of shared virtual addresses to a unique unshared area of physical memory, where the segment has been made permanently resident (after having been initially brought in from the disk).

The term "load", as used in this manual, refers to both processes.

In the autoload method, loading and error recovery are handled by the Overlay Runtime System. Overlays are automatically loaded by being referenced through a transfer-of-control instruction (CALL, JMP, or JSR). No explicit calls to the Overlay Runtime System are needed.

In the manual load method, the user handles loading and error recovery explicitly. Manual loading saves space and gives the user full control over the loading process, including the ability to specify whether loading is to be done synchronously or asynchronously.

Provision must be made for loading the overlay segments of the main tree and the root segments, as well as the overlay segments of the co-trees. Once loaded, the root segment of a co-tree remains in memory.

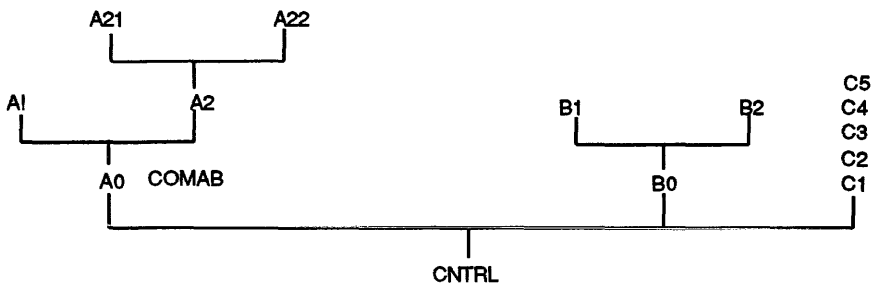
## 8.1 Autoload

When using the autoload method you place the autoload indicator "*" in the ODL description of the task at the points where loading must take place. The execution of a transfer of control instruction to an autoloadable segment up-tree automatically initiates the autoload process.

### 8.1.1 Autoload Indicator

The autoload indicator, "*", is placed in the overlay description at the points where autoloading is required. If the autoload indicator is inserted before parentheses (and before an exclamation point operator if used) then every name within the parentheses is marked autoloadable. Applying the autoload indicator at the outermost parentheses level of the ODL tree description marks every module in the overlay segments autoloadable.

Consider the example TK1 of Chapter 7, "Resolution of Global Symbols in a Multi-segment Task", and suppose further that segment C consists of a set of modules C1, C2, C3, C4 and C5. The tree diagram for TK1 then is:




If you introduce the autoload indicator at the outermost parentheses level, regardless of the flow of control within the task, a module is always properly loaded when it is called. The ODL description for the task with this provision then is:

```
            .ROOT CNTRL-*(AFCTR,BCTR,CFCTR)
AFCTR:      .FCTR A0-(A1,A2-(A21,A22))
BFCTR:      .FCTR B0-(B1,B2)
CFCTR:      .FCTR C1-C2-C3-C4-C5
            .END
```

To be assured that all modules of a co-tree are properly loaded, the user must mark the root segment as well as the outermost parentheses level of the co-tree, as follows:

```
.ROOT CNTRL-*(AFCTR,BFTCR,CFCTR),*CNTRL2-*(CNTRLX,CNTRLY)
. . .
```

The above example assumes that one or more modules containing executable code reside in CNTRL2.

The autoload indicator can be applied to the following constructs:

1 Filenames - to make all the components of the file autoloadable.

2 Parenthesized ODL tree descriptions - to make all the names within the parentheses autoloadable.

3 P-section names - to make the p-section autoloadable. The p-section must have the I (instruction) attribute.

4 Segment names introduced by the .NAME directive - to make all components of the segment to which the name applies autoloadable.

5 Factor label names - to make the first component of the factor autoloadable. If the entire factor is enclosed in parentheses, then all the components are made autoloadable.

Suppose you introduce two .PSECT directives and a .NAME directive into the ODL description for TK1 and then apply autoload indicators in the following way:

```
            .ROOT CNTRL-(*AFCTR,*BFCTR,*CFCTR)
AFCTR:      .FCTR A0-*ASUB1-ASUB2-*(A1,A2-(A21,A22))
BFCTR:      .FCTR (B0-(B1,B2))
CFCTR:      .FCTR CNAM-C1-C2-C3-C4-C5
            .NAME CNAM
            .PSECT ASUB1,I,GBL,OVR
            .PSECT ASUB2,I,GBL,OVR
            .END
```

The interpretation for each autoload indicator in the overlay description is as follows:

```
(*AFCTR,*BFCTR,*CFCTR)
```

The autoload indicator is applied to each factor name:

*AFCTR=*AO
*BFCTR=*(BO-(B1-B2))
*CFCTR=*CNAM

CNAM, however, is an element defined by a .NAME directive; therefore, all the components of the segment to which the name applies are made autoloadable; that is, C1, C2, C3, C4, and C5.

*ASUB1 - The autoload indicator is applied to the name of a p-section having the I attribute, so the p-section ASUB1 is made autoloadable, That is, all symbols defined in the p-section will be autoloadable.

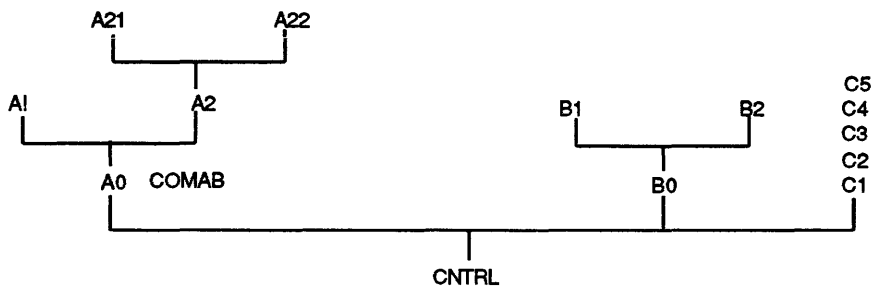
*(A1,A2-(A21,A22)) - The autoload indicator is applied to a portion of the ODL description enclosed in parentheses, so every element within the parentheses is made autoloadable, that is, files A1, A2, A21, and A22.

The effect of this ODL description is to make every element except p-section ASUB2 autoloadable.

## 8.1.2 Path-loading

Autoload uses the technique of path-loading. That is, whenever a segment is loaded all segments between it and the root are also loaded.

Consider again the example TK1 and the tree diagram:



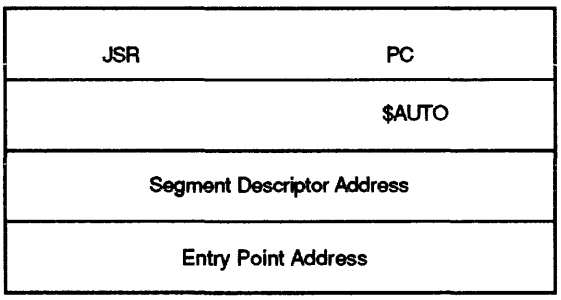
If CNTRL calls A2, then all the modules between the calling module CNTRL and the called module A2 are loaded. In this case modules A0 and A2 are loaded.

The Overlay Runtime System keeps track of the segments in memory and only issues load requests for those segments not in memory. If, in the above example, CNTRL called A1 and then called A2, A0 and A1 are loaded first and then A2 is loaded. A0 is not loaded when A2 is loaded because it is already in memory.

A reference from one segment to another segment down-tree (closer to the root) is resolved directly. For example, if A2 calls A0, then the reference is resolved directly because A0 is known to be in memory as a result of the path-loading that took place in the call to A2.

## 8.1.3 Autoload Vectors

When the Task Builder sees a reference from a segment to an autoloadable segment up-tree, it generates an autoload vector in the segment for the referenced global symbol. The definition of the symbol is changed to an autoload vector table entry. The autoload vector has the following format:




A "transfer of control" instruction to the referenced global symbol executes the call to the autoload routine $AUTO contained in the autoload vector.

An exception is made in the case of a p-section with the D (data) attribute. References from a segment to a global symbol up-tree in a p-section with the D attribute are resolved directly.

Since the Task Builder has no information about the flow of control within the task, it often generates more autoload vectors than are necessary. You can, however, apply your knowledge of the flow of control of your task and your knowledge of path-loading to determine the placement of autoload indicators. By placing the autoload indicators only at the points where loading is actually required, you can minimize the number of autoload vectors generated for the task.

If in TK1 all the calls to overlays originate in the root segment, (that is, no module in an overlay segment calls outside its overlay segment) and if the root segment CNTRL has the following contents:

```
PROGRAM CNTRL
CALL A1
CALL A21
CALL A2
CALL A0
CALL A22
CALL B0
CALL B1
```

```
CALL B2
CALL C1
CALL C2
CALL C3
CALL C4
CALL C5
END
```

If the autoload indicator is placed at the outermost parentheses level, 13 autoload vectors are generated for this task.

Since A2 and A0 are loaded by path loading to A21, the autoload vectors for A2 and A0 are unnecessary. The call to C1 loads the segment that contains C2, C3, C4 and C5; therefore autoload vectors for C2 through C5 are unnecessary.

You eliminate the unnecessary autoload vectors by placing the autoload indicator only at the points where loading is required, as follows:

```
            .ROOT CNTRL-(AFCTR,*BFCTR,CFCTR)
AFCTR:      .FCTR A0-(*A1,A2-*(A21,A22))
BFCTR:      .FCTR (B0-(B1,B2))
CFCTR:      .FCTR *C1-C2-C3-C4-C5
            .END
```

With this ODL description, the Task Builder generates only seven autoload vectors, namely those for A1, A21, A22, B0, B1, B2, and C1.

The autoload vectors for each segment are placed in the p-section $$ALVC, which is generated automatically by TKB. This p-section is read/write, so if a resident overlay segment which is otherwise read-only contains autoload references, it will become read/write and will not be shareable between multiple copies of the task.

This may be avoided by including a GBLREF option specifying each symbol for which there is an up-tree reference from the resident segment. This forces the autoload vector to be placed in the root segment.

## 8.1.4 Autoload Summary

1 Autoload is almost totally transparent to the user task. In particular, all registers are preserved across an autoloadable transfer of control. However, the condition code settings are not preserved across such a call.

2 Autoload can work only where the reference to another segment is explicit at task-build time. In particular, a segment can never be autoloaded as a result of a return from a subroutine.

3 Autoload should not be used in conjunction with the use of the .PSECT directive to move p-sections further from the root. When a global symbol is defined in a module, the symbol is associated with the segment containing the module. This is true even if the symbol is defined in a p-section which has been moved further away from the root. Thus if a segment closer to the root refers to the symbol, the segment which will be autoloaded will be the one containing the defining module, not the one containing the p-section. The effect of this is that the code labelled by the symbol will not be loaded before control is transferred.

## 8.2 Manual Load

If you decide to use the manual load method of loading segments, explicit calls to the $LOAD system routine must be included in the programs. These load requests give the name of the segment to be loaded and optionally give information necessary to perform asynchronous load requests and to handle unsuccessful load requests.

The $LOAD routine does not path-load. A call to $LOAD always results in the segment named in the load request being loaded and only that segment being loaded.

The MACRO-11 programmer calls the $LOAD routine directly. The FORTRAN programmer is provided with the subroutine "MNLOAD".

### 8.2.1 Manual Load Calling Sequence

The MACRO-11 programmer calls $LOAD, as follows:

```
        MOV     #PBLK,R0
        CALL    $LOAD
```

where PBLK labels a parameter block with the following format:

```
PBLK:   .BYTE   length,event-flag
        .RAD50  /seg-name/
        .WORD   I/O-status
        .WORD   AST-trp
```

You must specify the following parameters:

- length - the length of the parameter block (3-5 words).
- event-flag - the event flag number, used for asynchronous loading. If the event-flag number is zero, synchronous loading is performed.
- seg-name - the name of the segment to be loaded, a 1- to 6-character alphanumeric (Radix-50) name, occupying two words.

The following parameters are optional:

- I/O-status - the address of the I/O status block as described for the QIO directive in the IAS System Directive Reference Manual.
- AST-trp - the address of an AST routine to which control is transferred at the completion of the load request.

The condition code C is set or cleared on return, as follows:

If C = 0, the load request was successfully executed.
If C = 1, the load request was unsuccessful.

For a synchronous load request, the return of the condition code 0 means that the desired segment has been loaded and is ready to be executed. For an asynchronous load request, the return of the code 0 means that the load request has been successfully queued to the device, but the segment is not necessarily in memory. You must ensure that loading has been completed by waiting for the specified event flag before calling any routines or accessing any data in the segment.

## 8.2.2 FORTRAN Subroutine for Manual Load Request

To use manual load in a FORTRAN program, the program makes explicit reference to the $LOAD routine by means of the "MNLOAD" subroutine. The subroutine call has the following form:

```
CALL MNLOAD (seg-name,event-flag,I/O-status,ast-trp,ld-ind)
```

where:

- seg-name - is a 2-word real variable containing the segment name in alphanumeric (Radix-50) format.
- event-flag - is an optional integer event flag number, to be used for an asynchronous load request. If the event flag number is zero, the load request is considered synchronous.
- I/O-status - is an optional 2-word integer array to contain the I/O status doubleword, as described for the QIO directive in the IAS System Directives Reference Manual.
- ast-trp - is an optional asynchronous trap subroutine to be entered at the completion of a request. MNLOAD requires that all pending traps specify the same subroutine.
- ld-ind - is an optional integer variable to contain the results of the subroutine call. One of the following values is returned:

  +1 request was successfully executed.
  -1 request had bad parameters or was not executed successfully.

Optional arguments can be omitted. The following calls are all legal:

| Call | Effect |
|---|---|
| CALL MNLOAD (SEGA1) | Load the segment named in SEGA1 synchronously. |
| CALL MNLOAD (SEGA1,0,,,LDIND) | Load the segment named in SEGA1 synchronously. and return success indicator to LDIND. |
| CALL MNLOAD (SEGA1,1,IOSTAT, ASTSUB,LDIND) | Load the segment named in SEGA1 asynchronously, transferring control to ASTSUB upon completion of the load request, storing the I/O-status doubleword in IOSTAT and the success indicator in LDIND. |

Consider the program CNTRL described in connection with the autoload method, and suppose that between the calls to the overlay segments there is sufficient processing to make asynchronous loading effective. The user removes the autoload indicators from the ODL description and recompiles the FORTRAN programs with explicit calls to the MNLOAD subroutine, as follows:

```
PROGRAM CNTRL
EXTERNAL ASTSUB
INTEGER IOSTAT(2)
COMMON /IOSTAT/ IOSTAT
DATA SEGA1 /6RA1    /
DATA SEGA21 /6RA21  /
...
CALL MNLOAD (SEGA1,1,IOSTAT,ASTSUB,LDIND)
...
CALL A1
...
CALL MNLOAD (SEGA21,1,IOSTAT,ASTSUB,LDIND)
...
CALL A21
...
...
END
```

The AST subroutine, ("ASTSUB" in the example), should normally be written in MACRO-11. It may access the I/O status block using the p-section IOSTAT, with attributes RW, OVR, GBL, D.

## 8.3 Error Handling

If the manual load method is selected, you must provide error handling routines which diagnose load errors and provide appropriate recovery.

If the autoload method is selected, a simple recovery procedure is provided, which checks the Directive Status Word (DSW) for the presence of an error indication. If the DSW indicates that no system dynamic storage is available, the routine issues a "wait for significant event" directive and tries again; if the problem is not dynamic storage, the recovery procedure generates a breakpoint synchronous trap. If the program is set to service the trap and returns without altering the state of the program, the request can be retried.

A more comprehensive user-written error recovery subroutine can be substituted for the system-provided routine if the following conventions are observed:

1 The error recovery routine must have the entry point name $ALERR.

2 The contents of all registers must be saved and restored.

On entry to $ALERR, R2 contains the address of the descriptor for the segment that could not be loaded. Before recovery action can be taken, the routine must determine the cause of the error by examining the following words in the sequence indicated:

1 $DSW - The Directive Status Word may contain an error status code, indicating that the I/O request to load the overlay segment was rejected by the Executive.

2 .NIOST - This is a 2-word I/O Status block containing the results of the load overlay request returned by the device handler. The status code occupies the low-order byte of word 0.

## 8.4 Example: CALC.TSK;5

Suppose the task CALC is now complete and error-free and you want to adjust the autoload vectors to minimize the amount of storage required. Your knowledge of the flow of control of the task determines that PROC3.OBJ is always in memory as a result of path-loading when it is called and therefore, the autoload vector for PROC3.OBJ can be eliminated.

The ODL description in CALTR.ODL is modified as follows:

```
                .ROOT RDIN-RPRT-ADTA-(*PROC1,*PROC2,P3FCTR)
P3FCTR:         .FCTR PROC3-*(SUB1,SUB2)
                .END
```

The task is built and the resulting memory allocation file in Figure 8-1 shows that the repositioning of the autoload indicator saved 10 bytes.

**Example 8–1 Memory Allocation File for CALC.TSK;5**

```
-CALC.TSK;5    MEMORY ALLOCATION MAP  TKB D28          PAGE 1
                      3-JUL-78   10:51


IDENTIFICATION : FORV02
STACK    LIMITS: 000000 000777 001000 00512.
PRG XFR ADDRESS: 022252
TOTAL ATTACHMENT DESCRIPTORS: 3.
TASK  IMAGE  SIZE  : 8544. WORDS
TASK HEADER SIZE: 160. WORDS
TASK ADDRESS LIMITS: 000000 041337
R-W DISK BLK LIMITS: 000003 000052 000050 00040.


CALC.TSK;5 OVERLAY DESCRIPTION:

BASE    TOP          LENGTH
----    ---          ------
000000  035413  035414  15116.    RDIN
035414  035757  000344  00228.      PROC1
035414  037717  002304  01220.      PROC2
035414  040277  002664  01460.      PROC3
040300  041147  000650  00424.        SUB1
040306  041337  001040  00544.        SUB2
```

**Example 8-1 Memory Allocation File for CALC.TSK;5 (continued)**

```
CALC.TSK;5    MEMORY ALLOCATION MAP  TKB D28          PAGE 2
RDIN                  3-JUL-78   10:51



*** ROOT SEGMENT: RDIN


R/W MEM  LIMITS: 000000 035413 035414 15116.
DISK BLK LIMITS: 000003 000040 000036 00030.


MEMORY ALLOCATION SYNOPSIS:

SECTION                                          TITLE  IDENT  FILE
-------                                          -----  -----  ----
. BLK.:(RW,I,LCL,REL,CON) 001000 000002 00002.
ADTA  :(RW,D,GBL,REL,OVR) 001002 002260 01200.
DTA   :(RW,D,GBL,REL,OVR) 003262 001442 00802.
                          003262 001442 00802. .MAIN. FORV02 RDIN.OBJ;2
                          003262 001442 00802. RPRT   FORV02 RPRT.OBJ;1
OTS$I :(RW,I,LCL,REL,CON) 004724 015270 06840.
                          004724 000000 00000. .MAIN. FORV02 RDIN.OBJ;2
OTS$P :(RW,D,CBL,REL,OVR) 022214 000036 00030.
$CODE :(RW,I,LCL,REL,CON) 022252 000132 00090.
                          022252 000000 00000. .MAIN. FORV02 RDIN.OBJ;2
                          022252 000000 00000. .MAIN. FORV02 RDIN.OBJ;2
                          022252 000116 00078. .MAIN. FORV02 RDIN.OBJ;2
                          022370 000000 00000. RPRT   FORV02 RPRT.OBJ;1
                          022370 000000 00000. RPRT   FORV02 RPRT.OBJ;1
                          022370 000014 00012. RPRT   FORV02 RPRT.OBJ;1
$DATA :(RW,D,LCL,REL,CON) 022404 003720 02000.
                          022404 000000 00000. .MAIN. FORV02 RDIN.OBJ;2
                          022404 001750 01000. .MAIN. FORV02 RDIN.OBJ;2
                          024354 000000 00000. RPRT   FORV02 RPRT.OBJ;1
                          024354 001750 01000. RPRT   FORV02 RPRT.OBJ;1
$DATAP:(RW,D,LCL,REL,CON) 026324 000032 00026.
                          026324 000000 00000. .MAIN. FORV02 RDIN.OBJ;2
                          026324 000022 00018. .MAIN. FORV02 RDIN.OBJ;2
                          026346 000000 00000. RPRT   FORV02 RPRT.OBJ;1
                          026346 000010 00008. RPRT   FORV02 RPRT.OBJ;1
$$ALER:(RW,I,LCL,REL,CON) 026356 000024 00020.
$$ALVC:(RW,D,LCL,REL,CON) 026402 000020 00016.
$$AOTS:(RW,D,LCL,REL,CON) 026422 000704 00452.
$$AUTO:(RW,I,LCL,REL,CON) 160000 000130 00088.
$$DEVT:(RW,D,LCL,REL,OVR) 027326 001210 00648.
$$FSR1:(RW,D,GBL,REL,OVR) 030536 004100 02112.
$$FSR2:(RW,D,GBL,REL,CON) 034636 000104 00068.
$$IOB1:(RW,D,LCL,REL,OVR) 034742 000204 00132.
$$IOB2:(RW,D,LCL,REL,OVR) 035146 000000 00000.
$$LOAD:(RW,I,LCL,REL,CON) 160130 000170 00120.
$$MRKS:(RW,I,LCL,REL,OVR) 160320 000166 00118.
$$OBF1:(RW,D,LCL,REL,CON) 035146 000110 00072.
$$OBF2:(RW,I,LCL,REL,CON) 035256 000000 00000.
$$OVDT:(RW,D,ICL,REL,OVR) 035256 000020 00016.
$$OVRS:(RW,I,LCL,ABS,CON) 000000 000000 00000.
$$RDSG:(RW,I,LCL,REL,OVR) 160506 000312 00202.
$$RESL:(RW,I,LCL,REL,CON) 161020 016216 07310.
$$RCDS:(RW,D,LCL,REL,CON) 035276 000000 00000.
```

**Example 8-1 Memory Allocation File for CALC.TSK;5 (continued)**

```
CALC.TSK;5    MEMORY ALLOCATION MAP  TKB D28          PAGE 3
RDIN                    3-JUL-78   10:51


$$RTS :(RW,I,GBL,REL,OVR) 035276 000002 00002.
$$SGD0:(RW,D,LCL,REL,OVR) 035300 000000 00000.
$$SGD1:(RW,D,LCL,REL,CON) 035300 000110 00072.
$$SGD2:(RW,D,LCL,REL,OVR) 035410 000002 00002.
$$WNDS:(RW,D,LCL,REL,CON) 035412 000000 00000.
.$$$$.:(RW,D,GBL,REL,OVR) 035412 000000 00000.
                          035412 000000 00000. .MAIN. FORV02 RDIN.OBJ;2
                          035412 000000 00000. .MAIN. FORV02 RDIN.OBJ;2
                          035412 000000 00000. RPRT   FORV02 RPRT.OBJ;1
                          035412 000000 00000. RPRT   FORV02 RPRT.OBJ;1


GLOBAL SYMBOLS:

PROC1  026402-R  RPRT    022370-R  $$OTSC 022252-R
PROC2  026412-R  $RF2A1  000000-R  $$OTSI 004724-R

CALC.TSK;5    MEMORY ALLOCATION MAP  TKB D28          PAGE 4
PROC1                   3-JUL-78   10:51


*** SEGMENT: PROC1


R/W MEM  LIMITS: 035414 035757 000344 00228.
DISK BLK LIMITS: 000042 000042 000001 00001.


MEMORY ALLOCATION SYNOPSIS:

SECTION                                          TITLE  IDENT  FILE
-------                                          -----  -----  ----
. BLK.:(RW,I,LCL,REL,CON) 035414 000000 00000.
ADTA  :(RW,D,GBL,REL,OVR) 001002 002260 01200.
                          001002 002260 01200. PROC1  FORV02 PROC1.OBJ;2
DTA   :(RW,D,GBL,REL,OVR) 003262 001442 00802.
                          003262 001442 00802. PROC1  FORV02 PROC1.OBJ;2
OTS$I :(RW,I,LCL,REL,CON) 035414 000254 00172.
$CODE :(RW,I,LCL,REL,CON) 035670 000054 00044.
                          035670 000000 00000. PROC1  FORV02 PROC1.OBJ;2
                          035670 000000 00000. PROC1  FORV02 PROC1.OBJ;2
                          035670 000054 00044. PROC1  FORV02 PROC1.OBJ;2
$DATA :(RW,D,LCL,REL,CON) 035744 000002 00002.
                          035744 000000 00000. PROC1  FORV02 PROC1.OBJ;2
                          035744 000002 00002. PROC1  FORV02 PROC1.OBJ;2
$DATAP:(RW,D,LCL,REL,CON) 035746 000010 00008.
                          035746 000000 00000. PROC1  FORV02 PROC1.OBJ;2
                          035746 000010 00008. PROC1  FORV02 PROC1.OBJ;2
$$ALVC:(RW,D,LCL,REL,CON) 035756 000000 00000.
.$$$$.:(RW,D,GBL,REL,OVR) 035412 000000 00000.
                          035412 000000 00000. PROC1  FORV02 PROC1.OBJ;2
                          035412 000000 00000. PROC1  FORV02 PROC1.OBJ;2


GLOBAL SYMBOLS:

PROC1  035670-R
```

**Example 8-1 Memory Allocation File for CALC.TSK;5 (continued)**

```
CALC.TSK;5    MEMORY ALLOCATION MAP  TKB D28          PAGE 5
PROC2                 3-JUL-78   10:51


*** SEGMENT: PROC2


R/W MEM  LIMITS: 035414 037717 002304 01220.
DISK BLK LIMITS: 000043 000045 000003 00003.


MEMORY ALLOCATION SYNOPSIS:

SECTION                                          TITLE  IDENT  FILE
-------                                          -----  -----  ----
ADTA  :(RW,D,GBL,REL,OVR) 001002 002260 01200.
                          001002 002260 01200.  PROC2  FOFV02 PROC2.OBJ;1
$CODE :(RW,I,LCL,REL,CON) 035414 000014 00012.
                          035414 000000 00000.  PROC2  FORV02 PROC2.OBJ;1
                          035414 000000 00000.  PROC2  FORV02 PROC2.OBJ;1
                          035414 000014 00012.  PROC2  FORV02 PROC2.OBJ;1
$DATA :(RW,D,LCL,REL,CON) 035430 002260 01200.
                          035430 000000 00000.  PROC2  FORV02 PROC2.OBJ;1
                          035430 002260 01200.  PROC2  FOPV02 PROC2.OBJ;1
$DATAP:(RW,D,LCL,REL,CON) 037710 000010 00008.
                          037710 000000 00000.  PROC2  FOFV02 PROC2.OBJ;1
                          037710 000010 00008.  PROC2  FOPV02 PROC2.OBJ;1
$$ALVC:(RW,D,LCL,REL,CON) 037720 000000 00000.
.$$$$.:(RW,D,GBL,REL,OVR) 035412 000000 00000.
                          035412 000000 00000.  PROC2  FOPV02 PROC2.OBJ;1
                          035412 000000 00000.  PROC2  FORV02 PROC2.OBJ;1


GLOBAL SYMBOLS:

PROC2  035414-R
```

**Example 8-1 Memory Allocation File for CALC.TSK;5 (continued)**

```
CALC.TSK;5   MEMORY ALLOCATION MAP  TKB D28          PAGE 6
PROC3                3-JUL-78   10:51


*** SEGMENT: PROC3


R/W MEM  LIMITS: 035414 040277 002664 01460.
DISK BLK LIMITS: 000046 000050 000003 00003.


MEMORY ALLOCATION SYNOPSIS:

SECTION                                          TITLE  IDENT  FILE
-------                                          -----  -----  ----
ADTA  :(RW,D,CBL,REL,OVR) 001002 002260 01200.
                          001002 002260 01200.  PROC3  FORV02 PPOC3.OBJ;1
DTA   :(RW,D,CBL,REL,OVR) 003262 001442 00802.
                          003262 001442 00802.  PROC3  FORV02 PROC3.OBJ;1
$CODE :(RW,I,LCL,REL,CON) 035414 000044 00036.
                          035414 000000 00000.  PROC3  FORV02 PROC3.OBJ;1
                          035414 000000 00000.  PROC3  FORV02 PROC3.OBJ;1
                          035414 000044 00036.  PROC3  FORV02 PROC3.OBJ;1
$DATA :(RW,D,LCL,REL,CON) 035460 002570 01400.
                          035460 000000 00000.  PROC3  FORV02 PROC3.OBJ;1
                          035460 002570 01400.  PROC3  FORV02 PROC3.OBJ;1
$DATAP:(RW,D,LCL,REL,CON) 040250 000010 00008.
                          040250 000000 00000.  PROC3  FORV02 PROC3.OBJ;1
                          040250 000010 00008.  PROC3  FORV02 PROC3.OBJ;1
$$ALVC:(RW,D,LCL,REL,CON) 040260 000020 00016.
.$$$$.:(RW,D,GBL,REL,OVR) 035412 000000 00000.
                          035412 000000 00000.  PROC3  FORV02 PROC3.OBJ;1
                          035412 000000 00000.  PROC3  FORV02 PROC3.OBJ;1


GLOBAL SYMBOLS:

PROC3  035414-R  SUB1   040270-R  SUB2   040260-R
```

**Example 8-1 Memory Allocation File for CALC.TSK;5 (continued)**

```
CALC.TSK;5   MEMORY ALLOCATION MAP  TKB D28          PAGE 7
SUB1                  3-JUL-78   10:51


*** SEGMENT: SUB1


R/W MEM  LIMITS: 040300 041147 000650 00424.
DISK BLK LIMITS: 000051 000051 000001 00001.


MEMORY ALLOCATION SYNOPSIS:

SECTION                                          TITLE  IDENT  FILE
-------                                          -----  -----  ----
ADTA  :(RW,D,GBL,REL,OVR) 001002 002260 01200.
                          001002 002260 01200. SUB2   FORV02 SUB1.OBJ;1
$CODE :(RW,I,LCL,REL,CON) 040300 000014 00012.
                          040300 000000 00000. SUB2   FORV02 SUB1.OBJ;1
                          040300 000000 00000. SUB2   FORV02 SUB1.OBJ;1
                          040300 000014 00012. SUB2   FORV02 SUB1.OBJ;1
$DATA :(RW,D,LCL,REL,CON) 040314 000624 00404.
                          040314 000000 00000. SUB2   FORV02 SUB1.OBJ;1
                          040314 000624 00404. SUB2   FORV02 SUB1.OBJ;1
$DATAP:(RW,D,LCL,REL,CON) 041140 000010 00008.
                          041140 000000 00000. SUB2   FORV02 SUB1.OBJ;1
                          041140 000010 00008. SUB2   FORV02 SUB1.OBJ;1
$$ALVC:(RW,D,LCL,REL,CON) 041150 000000 00000.
.$$$$.:(RW,D,GBL,REL,OVR) 035412 000000 00000.
                          035412 000000 00000. SUB2   FORV02 SUB1.OBJ;1
                          035412 000000 00000. SUB2   FORV02 SUB1.OBJ;1


GLOBAL SYMBOLS:

SUB2   040300-R
```

**Example 8-1 Memory Allocation File for CALC.TSK;5 (continued)**

```
CALC.TSK;5    MEMORY ALLOCATION MAP  TKB D28          PAGE 8
SUB2                    3-JUL-78   10:51


*** SEGMENT: SUB2


R/W MEM  LIMITS: 040300 041337 001040 00544.
DISK BLK LIMITS: 000052 000053 000002 00002.


MEMORY ALLOCATION SYNOPSIS:

SECTION                                          TITLE  IDENT  FILE
-------                                          -----  -----  ----
. BLK.:(RW,I,LCL,REL,CON) 040300 000000 00000.
ADTA   :(RW,D,GBL,REL,OVR) 001002 002260 01200.
                           001002 002260 01200. SUB1   FORV02 SUB2.OBJ;1
DTA    :(RW,D,GBL,REL,OVR) 003262 001442 00802.
                           003262 001442 00802. SUB1   FORV02 SUB2.OBJ;1
OTS$I  :(RW,I,LCL,REL,CON) 040300 000154 00108.
$CODE  :(RW,I,LCL,REL,CON) 040454 000032 00026.
                           040454 000000 00000. SUB1   FORV02 SUB2.OBJ;1
                           040454 000000 00000. SUB1   FORV02 SUB2.OBJ;1
                           040454 000032 00026. SUB1   FORV02 SUB2.OBJ;1
$DATA  :(RW,D,LCL,REL,CON) 040506 000622 00402.
                           040506 000000 00000. SUB1   FORV02 SUB2.OBJ;1
                           040506 000622 00402. SUB1   FORV02 SUB2.OBJ;1
$DATAP:(RW,D,LCL,REL,CON) 041330 000010 00008.
                           041330 000000 00000. SUB1   FORV02 SUB2.OBJ;1
                           041330 000010 00008. SUB1   FORV02 SUB2.OBJ;1
$$ALVC:(RW,D,LCL,REL,CON) 041340 000000 00000.
.$$$$.:(RW,D,GBL,REL,OVR) 035412 000000 00000.
                           035412 000000 00000. SUB1   FORV02 SUB2.OBJ;1
                           035412 000000 00000. SUB1   FORV02 SUB2.OBJ;1


GLOBAL SYMBOLS:

SUB1    040454-R



*** TASK BUILDER STATISTICS:

    TOTAL WORK FILE REFERENCES: 22122.
    WORK  FILE  READS: 0.
    WORK  FILE WRITES: 0.
    SIZE OF CORE POOL: 16010. WORDS (62. PAGES)
    SIZE OF WORK FILE: 7680. WORDS (30. PAGES)

    ELAPSED TIME:00:00:20
```

## 8.5 Using the QIO Directive to Load from the Task Image File

It is sometimes required to load part of the task image file into a location other than that allocated by Task Builder. This can be the case, for example, if a segment containing error messages has been generated and given the NOPHY attribute so that no task address space is allocated to it.

The QIO function code IO.LOV can be used to load one or more blocks from the task image file into a specified buffer. The format is:

```
QIO[$]      IO.LOV,lun,efn,pri,iosb,ast,<bufadr,buflen,,,block>
QIOW[$]
```

where:

- lun - is the LUN to be used for the function. This must be the LUN whose number is at location .NOVLY in the overlay control block, which is inserted automatically into an overlaid task.
- efn - is an optional event flag. The flag will be set when the function has been completed and either the overlay has been successfully completed or an error has occurred.
- pri - is the priority of the request. This parameter should normally be omitted, in which case the task's priority will be used.
- iosb - is the address of the 2-word I/O status block. If this parameter is supplied, the first word of the status block will contain the status of the request on completion. This will normally be one of:

  IS.SUC - request successfully completed
  IE.OVR - request issued on the wrong LUN or specified block number not in the task image file
  IE.VER - device parity error
  For other error statuses see the *IAS Device Handlers Reference Manual*, Chapter 4.
- ast - (optional) is the address of an AST routine to be executed when the request has been completed.
- bufadr - is the even address of the buffer which is to receive the data.
- buflen - is the even length in bytes of the buffer.
- ,, - the two additional commas are mandatory and indicate two null parameters.
- block - is the block number of the first block to be read. This is the relative block number in the task image file. The starting block of an overlay segment can be found from the segment table, whose location and format is described in Appendix C, Section C.7.

Example:

```
QIOW$S  #IO.LOV,.NOVLY,#EFN,,#IOSB,,<#BUFF,#512,,,R1>
```

which will read the block, specified in R1, of the image file of the issuing task into the buffer at BUFF.

# 9 SHAREABLE GLOBAL AREAS

IAS provides the facility for dynamic Shareable Global Areas (SGAs). This chapter describes the use and creation of SGAs in so far as they are related to task building.

## 9.1 Summary OF SGA Information

This summary lists all the important information about SGAs and provides references to further information about each item.

1 SGAs are created using the Task Builder. They do not have task headers or stacks (see Chapter 5, Sections "Header" (PDS) or "FX" (MCR) and "Stack").

2 An SGA must be installed before any task that uses it can be installed or run. See either the IAS MCR User's Guide or the IAS PDS User's Guide.

3 Access permissions are established when the SGA is installed. Read, write, extend and delete access to the SGA can be allowed or denied for tasks that are not owned by the owner of the SGA. See either the IAS MCR User's Guide or the IAS PDS User's Guide for further information.

4 SGAs occupy memory only when one or more referencing tasks are active. When all referencing tasks become inactive, the space occupied by shareable global areas is freed. The executive's treatment of the SGA at this point depends on whether it is a resident library, a common area or an installed region. See the IAS Executive Facilities Reference Manual for further details.

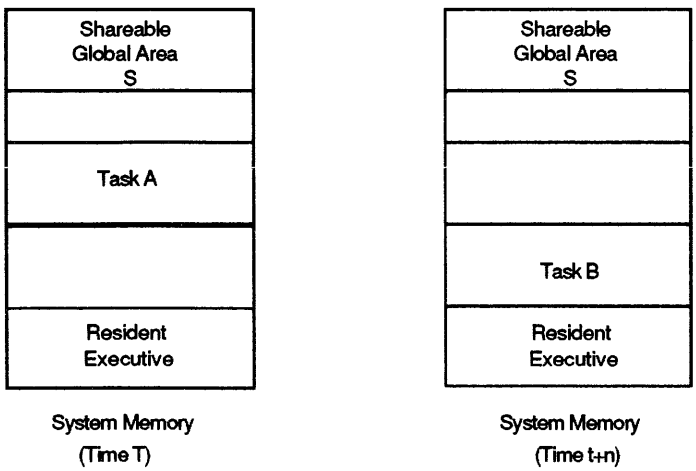
5 When a task which uses an SGA is built, the SGA must exist in the form of a task image and symbol table file (see Section 9.3).

6 When a task is built the SGAs it uses are named using the SGA or RESSGA option. In addition the access it requires to these SGAs is declared (see Chapter 5, Section 5.3). This access is always subject to that granted by the SGA and specified when installing the SGA (see 3. above).

### 9.1.1 Sharing Memory

Consider first the case in which two tasks, Task A and Task B, need to communicate a large amount of data. A convenient method of transmitting this data is using a read/write common area or installed region. Tasks can communicate independently of their time of execution. This case is illustrated in Figure 9–1.

Figure 9–1 SGA as a Common Data Area

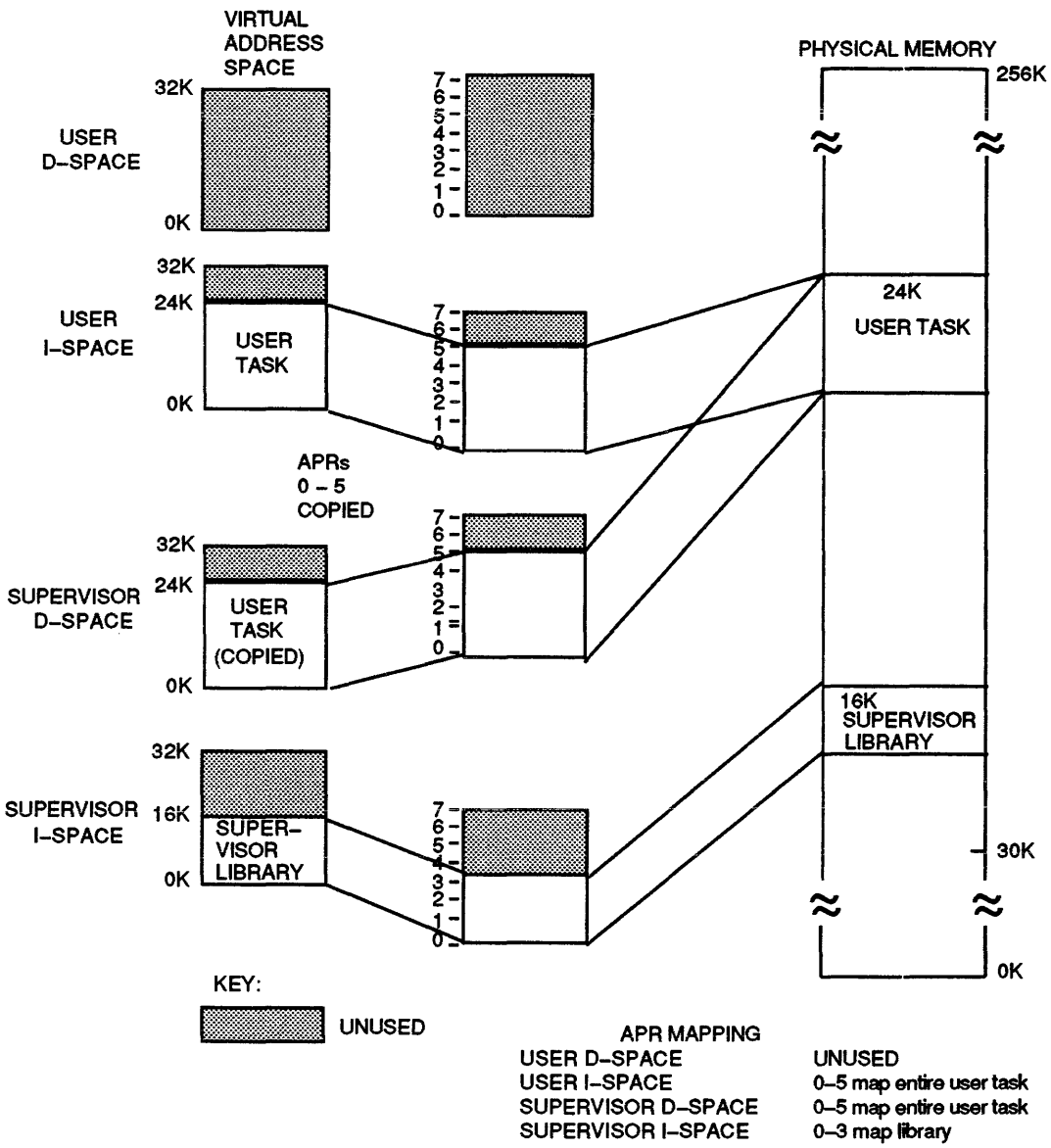



Task A and Task B communicate through the shareable global area, to which any number of tasks can be linked.

Changes to the SGA are retained throughout any swapping (or checkpointing) operations because the SGA is a common area or installed region and is written to disk at these times. The SGA is also written to disk if Task A exits before Tasks B begins.

Next, consider the case where tasks A and B use common code routines. The routines can be included in a read-only shareable global area so that a single copy is accessible to all tasks. Figure 9–2 illustrates this case.

**Figure 9–2 Tasks Using the Same Routines**




In this case, the SGA could be a resident library because the deletion of the memory version and reloading of task image file is of no consequence. The deletion and reloading are irrelevant because the SGA contains static, read-only information.

A task can link to a maximum of seven shareable global areas depending on the size of the task and the SGAs. If, however, the task is multi-user and has read-only sections in the root, this pure area of the root is considered as an SGA, and the number of external SGAs which can be linked to the task is reduced to six. Further, each SGA must begin in a separate APR.

A shareable global area has associated with it a task image file and a symbol definition file. When a task links to a shareable global area the Task Builder uses the symbol definition file of the shareable global area to establish the linkages between the task and the shareable global area.

### 9.1.2 Location of SGAs on Disk

SGAs for general (public) use are stored in LB0:[1,1]. They include the library SGA SYSRES.

SYSRES contains commonly used routines for the IAS file services, for automatic overlay loading and for data format conversion. SYSRES is linked to a task by default if no options are specified to the LINK command.

If a task requests access to an SGA via the SGA task builder option (see Chapter 5, Section "SGA") the SGA must be in LB0:[1,1].

SGAs can be stored by the user in other file areas as appropriate to a particular user or group of users. Such SGAs must be requested by the RESSGA option (see Chapter 5, Section "RESSGA").

### 9.1.3 SGAs and Library Files

A resident library SGA is not the same as a library file of object modules.

When routines are built into an SGA, an accessing task maps on to the SGA and only one copy is loaded into memory for all such tasks, as shown in Figure 9–2.

When routines are extracted from a library file, a copy of the necessary object modules is loaded for each task requiring the routines (see Figure 9–2).

Either method can be used depending on the time and memory requirements of the particular application. For example, the routines in the SGA SYSRES are also among those supplied in the library file LB0:[1,1]SYSLIB.OLB.

## 9.2 Using an Existing Shareable Global Area

The user can link a task to any of the system SGAs by specifying the SGA keyword option along with the name of the SGA and the type of access required.

If the user wants to link task IMG1 to a system SGA named JRNAL so that data can be examined but not overwritten, the SGA keyword can be used to specify the name JRNAL and the read-only attribute.

```
PDS> LINK/TASK:IMG1/MAP/OPTIONS
FILE? IN1,IN2,IN3
OPTIONS? SGA=JRNAL:RO
OPTIONS? /
```

or

```
TKB>IMG1,LP:= IN1,IN2,IN3
TKB>/
ENTER OPTIONS:
TKB>SGA=JRNAL:RO
TKB>//
```

A task can link to any SGA on the disk. However, before the task can be activated, all SGAs it uses must be installed.

## 9.3 Creating a Shareable Global Area

To create a shareable global area, the task image and symbol definition files must be built.

Runnable tasks were described in Chapter 6, Section 6.1.1. An SGA differs from a runnable task in that it does not have a header or a stack. Therefore, the user must specify that the header and stack are not to be produced for the task image file when an SGA is created. The task image and symbol table of an SGA must have the same filename and the (default) types .TSK and .STB. This set of conditions is necessary and sufficient to identify the entity as an SGA.

In summary, to create an SGA the following steps are taken:

1 The task image file is built, specifying also a symbol definition file.

2 The task image file has the /NOHEADER (/-HD) qualifier, indicating that no header is required.

3 The option STACK=0 is entered during option input to eliminate the stack.

4 Although it is not mandatory, the user can save disk space by setting UNITS=0.

Suppose the user wants to create a resident library, ZETA, from the files Z1, Z2, and Z3. Suppose that it is to be accessed via the task builder option SGA, and so must be held in LB0:[1,1]. The SGA is built as follows:

```
PDS> LINK/TASK:LB0:[1,1]ZETA/NOHEADER/MAP-
/SYMBOLS:LB0:[1,1]ZETA/OPTIONS
FILE? Z1,Z2,Z3
OPTIONS? STACK=0
OPTIONS? UNITS=0
OPTIONS? /
```

or

```
TKB>LB0:[1,1]ZETA/-HD,ZETA,LB0:[1,1]ZETA=Z1,Z2,Z3
TKB>/
ENTER OPTIONS:
TKB>STACK = 0
TKB>UNITS = 0
TKB>/
```

A task can now link to the SGA. However, before a task can be installed and activated, the SGA must be made known to the Executive via Install, defining the owner, non-owner access and the type of SGA. The following example illustrates a typical installation procedure for a library SGA. See the INSTALL command in either the IAS MCR User's Guide or PDS User's Guide.

```
PDS> INSTALL/LIBRARY:ZETA/UIC:[1,1]/ACCESS:RO [1,1]ZETA
```

or

```
MCR>INS [1,1]ZETA/LI/TASK=ZETA/ACC=RO/UIC=[1,1]
```

## 9.4 Position Independent and Absolute Shareable Global Areas

A shareable global area can be either position independent or absolute. Position independent SGAs can be placed anywhere in the task's virtual address space. Absolute areas must be placed at a fixed position in the virtual address space.

The user must ensure that an area is in fact position independent if the POSITION_INDEPENDENT qualifier is specified. The qualifier directs the Task Builder to treat the area as position independent even though the Task Builder cannot determine the position independence of the area. If it is not truly position independent, the execution of a task linked to that area is unpredictable.

Data is always position independent unless it contains internal pointers. Code can be position independent, but the code produced as a result of compiling a FORTRAN program is not position independent. Furthermore, FORTRAN subroutines cannot be used as SGAs because these programs do not satisfy the re-entrancy requirements necessary for SGAs. Refer to the IAS/RSX-11 MACRO-11 Reference Manual for a further description of position independent coding (PIC).

FORTRAN common blocks can be included in SGAs. The only way FORTRAN programs can communicate through the use of common blocks is by the common block name; to retain this name, the SGA must be declared position independent. If the area is not declared position independent, the name is not retained and no FORTRAN program can link to the common block.

Absolute SGAs are used for code or data that is not position- independent. The BASE or TOP Task Builder options are used to build such SGAs.

It is possible for an SGA to reference another SGA, using the SGA or RESSGA options. In this case, symbols are resolved from the reference SGA when the referencing SGA is built. Any task which binds to the referencing SGA is also automatically bound to the referenced SGA with the access specified when the SGA was built. A non-PI SGA may reference any SGA, whether or not that area is built position-independent, however, a PI SGA may not reference another SGA.

## 9.5 Example: CALC.TSK;6 Building and Using a Shareable Global Area

Suppose the task CALC has been completely debugged and the user wants to replace the dummy reporting routine RPRT by a generalized reporting program that operates as a separate task. This generalized reporting program GPRT was developed by another programmer in parallel with the development of CALC. Now both routines are ready and the user wants to create an SGA so that the two tasks can communicate.

In addition to creating the SGA, the user must modify the FORTRAN routine to replace the call to the dummy reporting routine by a call to REQUEST for the task GPRT; the user must also remove the dummy routine from the ODL description for the task.

### 9.5.1 Building the Shareable Global Area

The common block into which CALC places its results and from which GPRT takes its input is named DTA. The user wants to make DTA into a shareable global area so that the two tasks can communicate.

The user first creates a separate input file for DTA.FTN:

```
PDS> EDIT
FILE? DTA.FTN
[EDI -- CREATING NEW FILE]
INPUT
C
C     GLOBAL COMMON AREA FOR 'CALC' AND
C     REPORTING TASK 'GPRT'
      BLOCK DATA
      COMMON /DTA/ A(200),I
      END
 *EX
```

The user then compiles DTA.FTN:

```
PDS>FORTRAN/LIST DTA
```

Then the user builds the task image and symbol definition file for the SGA DTA.OBJ:

```
PDS> LINK/TASK:LB0:[1,1]DTA/POS/MAP-
/SYMBOLS:LB0:[1,1]DTA/NOHEADER/OPTIONS
FILE? DTA
OPTIONS? STACK=0
OPTIONS? UNITS=0
OPTIONS? /
```

or

```
TKB>LB0:[1,1]DTA/PI/-HD,LP0:,LB0:[1,1]DTA=DTA
TKB>/
ENTER OPTIONS:
TKB>STACK = 0
TKB>UNITS = 0
TKB>/
```

The task image file DTA.TSK is marked as position independent in order to retain the name of the referenced common block, DTA.

The task image and symbol definition files are created on LB0: under the UFD [1,1]. The /NOHEADER command qualifier (PDS) or /-HD switch (MCR) is applied to the symbol definition file to specify that the task has no header, the option STACK=0 is entered to eliminate the stack, and 0 logical units are specified.

The SGA DTA now exists on LB0: as a candidate for inclusion in an active system. The user can now modify the task to link to that SGA. However, before the task can be executed, the SGA must be installed.

## 9.5.2 Modifying the Task to Use the Shareable Global Area

The user now modifies the task CALC. The file containing the program RDIN is edited to include the name of the reporting task in alphanumeric (Radix-50) format:

```
DATA RPTSK/6RGPRT  /
```

And the call to the dummy reporting routine RPRT is replaced by the call:

```
CALL REQUES (RPTSK)
```

The relevant part of the program RDIN is shown below:

```
C   READ AND ANALYZE INPUT DATA
C   ESTABLISH COMMON DATA BASE
      COMMON /DTA/ A(200), I
C   SET UP NAME OF REPORTING TASK IN RADIX 50
      DATA RPTSK /6RGPRT  /
C     READ IN RAW DATA
      ...
      CALL REQUES (RPTSK)
      ...
      END
```

The user now modifies the ODL description of the task CALC to remove the file RPRT.OBJ. The .ROOT directive is changed from:

```
.ROOT RDIN-RPRT-ADTA-(*PROC1,*PROC2,P3FCTR)
```

to:

```
.ROOT RDIN-ADTA-(*PROC1,*PROC2,P3FCTR)
```

An indirect command file is then built to include the SGA keyword:

```
PDS> EDIT
FILE? CALCBLD.CMD
[EDI -- CREATING NEW FILE]
INPUT
LINK/TASK:CALC/MAP-
/OVERLAY:CALTR/OPTIONS
PAR=GEN
ACTFIL=1
SGA=DTA:RW
/
*EX
```

or

```
CALC,CALC=CALTR/MP
PAR=GEN
ACTFIL=1
SGA=DTA:RW
/
```

And the task is built with the single command referencing the indirect file:

```
PDS> @CALCBLD
```

or

```
MCR>TKB @CALCBLD
```

The communication between the two tasks, CALC and GPRT, is now established. When the SGA DTA is made resident, the two tasks can run.

## 9.5.3 The Memory Allocation Files

Example 9–1 shows the memory allocation file for the SGA DTA. The attribute list indicates that the task image is position independent (PI).

Example 9–2 shows the memory allocation file for the task CALC.TSK;6 after the SGA DTA was created and the dummy reporting routine removed from the task. The read-write memory limits for the root segment code have increased due to the call to REQUES. The read-write memory limits for the entire task have decreased because the common block DTA is now an SGA allocated at 140,000 and no longer part of the task code.

**Example 9–1 Memory Allocation File for SGA DTA**

```
DTA.TSK;1    MEMORY ALLOCATION MAP  TKB D28          PAGE 1
                      3-JUL-78   10:51


IDENTIFICATION : FORV02
TASK ATTRIBUTES: PI
TOTAL ATTACHMENT DESCRIPTORS: 0.
TASK  IMAGE  SIZE  : 384. WORDS
TASK ADDRESS LIMITS: 000000 001443
R-W DISK BLK LIMITS: 000002 000001 000000 00000.
R-O DISK BLK LIMITS: 000003 000001 177777 65535.

*** ROOT SEGMENT: DTA


R/W MEM  LIMITS: 000000 001443 001444 00804.
DISK BLK LIMITS: 000002 000003 000002 00002.


MEMORY ALLOCATION SYNOPSIS:

SECTION                                          TITLE  IDENT  FILE
-------                                          -----  -----  ----
DTA    :(RW,D,GBL,REL,OVR) 000000 001442 00802.
                           000000 001442 00802. .DATA. FORV02 DTA.OBJ;1
$CODE  :(RW,I,LCL,REL,CON) 001442 000000 00000.
                           001442 000000 00000. .DATA. FORV02 DTA.OBJ;1
$DATA  :(RW,D,LCL,REL,CON) 001442 000000 00000.
                           001442 000000 00000. .DATA. FORV02 DTA.OBJ;1
$DATAP:(RW,D,LCL,REL,CON) 001442 000000 00000.
                           001442 000000 00000. .DATA. FORV02 DTA.OBJ;1
.$$$$.:(RW,D,GBL,REL,OVR) 001442 000000 00000.
                           001442 000000 00000. .DATA. FORV02 DTA.OBJ;1
                           001442 000000 00000. .DATA. FORV02 DTA.OBJ;1



*** TASK BUILDER STATISTICS:

    TOTAL WORK FILE REFERENCES: 471.
    WORK  FILE  READS: 0.
    WORK  FILE WRITES: 0.
    SIZE OF CORE POOL: 16010. WORDS (62. PAGES)
    SIZE OF WORK FILE: 512. WORDS (2. PAGES)

    ELAPSED TIME:00:00:02
```

**Example 9–2 Memory Allocation File for CALC.TSK;6**

```
CALC.TSK;6   MEMORY ALLOCATION MAP  TKB D28          PAGE 1
                    3-JUL-78   10:52


IDENTIFICATION : FORV02
STACK    LIMITS: 000000 000777 001000 00512.
PRG XFR ADDRESS: 141442
TOTAL ATTACHMENT DESCRIPTORS: 4.
TASK  IMAGE  SIZE  : 7072. WORDS
TASK HEADER SIZE: 160. WORDS
TASK ADDRESS LIMITS: 000000 033567
R-W DISK BLK LIMITS: 000003 000044 000042 00034.



CALC.TSK;6 OVERLAY DESCRIPTION:

BASE    TOP         LENGTH
----    ---         ------
000000  027643  027644  12196.    RDIN
027644  030207  000344  00228.      PROC1
027644  032147  002304  01220.      PROC2
027644  032527  002664  01460.      PROC3
032530  033377  000650  00424.        SUB1
032530  033567  001040  00544.        SUB2
```

**Example 9-2 Memory Allocation File for CALC.TSK;6 (continued)**

```
CALC.TSK;6    MEMORY ALLOCATION MAP  TKB D28          PAGE 2
RDIN                    3-JUL-78   10:52



*** ROOT SEGMENT: RDIN


R/W MEM  LIMITS: 000000 027643 027644 12196.
DISK BLK LIMITS: 000003 000032 000030 00024.


MEMORY ALLOCATION SYNOPSIS:

SECTION                                          TITLE  IDENT  FILE
-------                                          -----  -----  ----
. BLK.:(RW,I,LCL,REL,CON) 001000 000002 00002.
ADTA   :(RW,D,GBL,REL,OVR) 001002 002260 01200.
DTA    :(RW,D,GBL,REL,OVR) 140000 001442 00802.
                           140000 001442 00802. .MAIN. FORV02 RDIN.OBJ;2
                           140000 001442 00802. RPRT   FORV02 RPRT.OBJ;1
OTS$I  :(RW,I,LCL,REL,CON) 003262 015270 06840.
                           003262 000000 00000. .MAIN. FORV02 RDIN.OBJ;2
OTS$P  :(RW,D,GBL,REL,OVR) 020552 000036 00030.
$CODE  :(RW,I,LCL,REL,CON) 141442 000000 00000.
                           141442 000000 00000. .MAIN. FORV02 RDIN.OBJ;2
                           141442 000000 00000. .MAIN. FORV02 RDIN.OBJ;2
                           141442 000116 00078. .MAIN. FORV02 RDIN.OBJ;2
                           141442 000000 00000. RPRT   FORV02 RPRT.OBJ;1
                           141442 000000 00000. RPRT   FORV02 RPRT.OBJ;1
                           141442 000014 00012. RPRT   FORV02 RPRT.OBJ;1
$DATA  :(RW,D,LCL,REL,CON) 141442 000000 00000.
                           141442 000000 00000. .MAIN. FORV02 RDIN.OBJ;2
                           141442 001750 01000. .MAIN. FORV02 RDIN.OBJ;2
                           141442 000000 00000. RPRT   FORV02 RPRT.OBJ;1
                           141442 001750 01000. RPRT   FORV02 RPRT.OBJ;1
$DATAP:(RW,D,LCL,REL,CON) 141442 000000 00000.
                           141442 000000 00000. .MAIN. FORV02 RDIN.OBJ;2
                           141442 000022 00018. .MAIN. FORV02 RDIN.OBJ;2
                           141442 000000 00000. RPRT   FORV02 RPRT.OBJ;1
                           141442 000010 00008. RPRT   FORV02 RPRT.OBJ;1
$$ALER:(RW,I,LCL,REL,CON) 020610 000024 00020.
$$ALVC:(RW,D,LCL,REL,CON) 020634 000020 00016.
$$AOTS:(RW,D,LCL,REL,CON) 020654 000704 00452.
$$AUTO:(RW,I,LCL,REL,CON) 160000 000130 00088.
$$DEVT:(RW,D,LCL,REL,OVR) 021560 001210 00648.
$$FSR1:(RW,D,GBL,REL,OVR) 022770 004100 02112.
$$FSR2:(RW,D,GBL,REL,CON) 027070 000104 00068.
$$IOB1:(RW,D,LCL,REL,OVR) 027174 000204 00132.
$$IOB2:(RW,D,LCL,REL,OVR) 027400 000000 00000.
$$LOAD:(RW,I,LCL,REL,CON) 160130 000170 00120.
$$MRKS:(RW,I,LCL,REL,OVR) 160320 000166 00118.
$$OBF1:(RW,D,LCL,REL,CON) 027400 000110 00072.
$$OBF2:(RW,I,LCL,REL,CON) 027510 000000 00000.
$$OVDT:(RW,D,LCL,REL,OVR) 027510 000020 00016.
$$OVRS:(RW,I,LCL,ABS,CON) 000000 000000 00000.
$$RDSG:(RW,I,LCL,REL,OVR) 160506 000312 00202.
$$RESL:(RW,I,LCL,REL,CON) 161020 016216 07310.
$$RGDS:(RW,D,LCL,REL,CON) 027530 000000 00000.
```

**Example 9-2 Memory Allocation File for CALC.TSK;6 (continued)**

```
CALC.TSK;6   MEMORY ALLOCATION MAP  TKB D28          PAGE 3
RDIN                  3-JUL-78   10:52


$$RTS :(RW,I,GBL,REL,OVR) 027530 000002 00002.
$$SGD0:(RW,D,LCL,REL,OVR) 027532 000000 00000.
$$SGD1:(RW,D,LCL,REL,CON) 027532 000110 00072.
$$SGD2:(RW,D,LCL,REL,OVR) 027642 000002 00002.
$$WNDS:(RW,D,LCL,REL,CON) 027644 000000 00000.
.$$$$.:(RW,D,GBL,REL,OVR) 141442 000000 00000.
                          141442 000000 00000. .MAIN. FORV02 RDIN.OBJ;2
                          141442 000000 00000. .MAIN. FORV02 RDIN.OBJ;2
                          141442 000000 00000. RPRT   FORV02 RPRT.OBJ;1
                          141442 000000 00000. RPRT   FORV02 RPRT.OBJ;1


GLOBAL SYMBOLS:

PROC1  020634-R  RPPT   141442-R  $$OTSC 141442-R
PROC2  020644-R  $RF2A1 000000-R  $$OTSI 003262-R

CALC.TSK;6   MEMORY ALLOCATION MAP  TKB D28          PAGE 4
PROC1                 3-JUL-78   10:52


*** SEGMENT: PROC1


R/W MEM  LIMITS: 027644 030207 000344 00228.
DISK BLK LIMITS: 000034 000034 000001 00001.

MEMORY ALLOCATION SYNOPSIS:

SECTION                                          TITLE  IDENT  FILE
-------                                          -----  -----  ----
. BLK.:(RW,I,LCL,REL,CON) 027644 000000 00000.
ADTA  :(RW,D,GBL,REL,OVR) 001002 002260 01200.
                          001002 002260 01200. PROC1  FORV02 PROC1.OBJ;2
DTA   :(RW,D,GBL,REL,OVR) 140000 001442 00802.
                          140000 001442 00802. PROC1  FORV02 PROC1.OBJ;2
OTS$I :(RW,I,LCL,REL,CON) 027644 000254 00172.
$CODE :(RW,I,LCL,REL,CON) 030120 000054 00044.
                          030120 000000 00000. PROC1  FORV02 PROC1.OBJ;2
                          030120 000000 00000. PROC1  FORV02 PROC1.OBJ;2
                          030120 000054 00044. PROC1  FORV02 PROC1.OBJ;2
$DATA :(RW,D,ICL,REL,CON) 030174 000002 00002.
                          030174 000000 00000. PROC1  FORV02 PROC1.OBJ;2
                          030174 000002 00002. PROC1  FORV02 PROC1.OBJ;2
$DATAP:(RW,D,LCL,REL,CON) 030176 000010 00008.
                          030176 000000 00000. PROC1  FORV02 PROC1.OBJ;2
                          030176 000010 00008. PROC1  FORV02 PROC1.OBJ;2
$$ALVC:(RW,D,LCL,REL,CON) 030206 000000 00000.
.$$$$.:(RW,D,GBL,REL,OVR) 141442 000000 00000.
                          141442 000000 00000. PROC1  FORV02 PROC1.OBJ;2
                          141442 000000 00000. PROC1  FORV02 PROC1.OBJ;2


GLOBAL SYMBOLS:

PROC1  030120-R
```

**Example 9-2 Memory Allocation File for CALC.TSK;6 (continued)**

```
CALC.TSK;6    MEMORY ALLOCATION MAP  TKB D28          PAGE 5
PROC2                 3-JUL-78   10:52


*** SEGMENT: PROC2


R/W MEM  LIMITS: 027644 032147 002304 01220.
DISK BLK LIMITS: 000035 000037 000003 00003.


MEMORY ALLOCATION SYNOPSIS:

SECTION                                          TITLE  IDENT  FILE
-------                                          -----  -----  ----
ADTA  :(RW,D,GBL,REL,OVR) 001002 002260 01200.
                          001002 002260 01200. PROC2  FORV02 PROC2.OBJ;1
$CODE :(RW,I,LCL,REL,CON) 027644 000014 00012.
                          027644 000000 00000. PROC2  FORV02 PROC2.OBJ;1
                          027644 000000 00000. PROC2  FORV02 PROC2.OBJ;1
                          027644 000014 00012. PROC2  FORV02 PROC2.OBJ;1
$DATA :(RW,D,LCL,REL,CON) 027660 002260 01200.
                          027660 000000 00000. PROC2  FORV02 PROC2.OBJ;1
                          027660 002260 01200. PROC2  FORV02 PROC2.OBJ;1
$DATAP:(RW,D,LCL,REL,CON) 032140 000010 00008.
                          032140 000000 00000. PROC2  FORV02 PROC2.OBJ;1
                          032140 000010 00008. PROC2  FORV02 PROC2.OBJ;1
$$ALVC:(RW,D,LCL,REL,CON) 032150 000000 00000.
.$$$$.:(RW,D,GBL,REL,OVR) 141442 000000 00000.
                          141442 000000 00000. PROC2  FORV02 PROC2.OBJ;1
                          141442 000000 00000. PROC2  FORV02 PROC2.OBJ;1


GLOBAL SYMBOLS:

PROC2  027644-R
```

**Example 9-2 Memory Allocation File for CALC.TSK;6 (continued)**

```
CALC.TSK;6   MEMORY ALLOCATION MAP  TKB D28          PAGE 6
PROC3                3-JUL-78   10:52


*** SEGMENT: PROC3


R/W MEM  LIMITS: 027644 032527 002664 01460.
DISK BLK LIMITS: 000040 000042 000003 00003.


MEMORY ALLOCATION SYNOPSIS:

SECTION                                          TITLE  IDENT  FILE
-------                                          -----  -----  ----
ADTA  :(RW,D,GBL,REL,OVR) 001002 002260 01200.
                          001002 002260 01200.  PROC3  FORV02 PROC3.OBJ;1
DTA   :(RW,D,GBL,REL,OVR) 140000 001442 00802.
                          140000 001442 00802.  PROC3  FORV02 PROC3.OBJ;1
$CODE :(RW,I,LCL,REL,CON) 027644 000044 00036.
                          027644 000000 00000.  PROC3  FORV02 PROC3.OBJ;1
                          027644 000000 00000.  PROC3  FORV02 PROC3.OBJ;1
                          027644 000044 00036.  PROC3  FORV02 PROC3.OBJ;1
$DATA :(RW,D,LCL,REL,CON) 027710 002570 01400.
                          027710 000000 00000.  PROC3  FORV02 PROC3.OBJ;1
                          027710 002570 01400.  PROC3  FORV02 PROC3.OBJ;1
$DATAP:(RW,D,LCL,REL,CON) 032500 000010 00008.
                          032500 000000 00000.  PROC3  FORV02 PROC3.OBJ;1
                          032500 000010 00008.  PROC3  FORV02 PROC3.OBJ;1
$$ALVC:(RW,D,LCL,REL,CON) 032510 000020 00016.
.$$$$.:(RW,D,GBL,REL,OVR) 141442 000000 00000.
                          141442 000000 00000.  PROC3  FORV02 PROC3.OBJ;1
                          141442 000000 00000.  PROC3  FORV02 PROC3.OBJ;1


GLOBAL SYMBOLS:

PROC3  027644-R  SUB1   032520-R  SUB2   032510-R
```

**Example 9-2 Memory Allocation File for CALC.TSK;6 (continued)**

```
CALC.TSK;6    MEMORY ALLOCATION MAP  TKB D28          PAGE 7
SUB1                  3-JUL-78   10:52


*** SEGMENT: SUB1


R/W MEM  LIMITS: 032530 033377 000650 00424.
DISK BLK LIMITS: 000043 000043 000001 00001.


MEMORY ALLOCATION SYNOPSIS:

SECTION                                          TITLE  IDENT  FILE
-------                                          -----  -----  ----
ADTA  :(RW,D,GBL,REL,OVR) 001002 002260 01200.
                          001002 002260 01200. SUB2   FORV02 SUB1.OBJ;1
$CODE :(RW,I,LCL,REL,CON) 032530 000014 00012.
                          032530 000000 00000. SUB2   FORV02 SUB1.OBJ;1
                          032530 000000 00000. SUB2   FORV02 SUB1.OBJ;1
                          032530 000014 00012. SUB2   FORV02 SUB1.CBJ;1
$DATA :(RW,D,LCL,REL,CON) 032544 000624 00404.
                          032544 000000 00000. SUB2   FORV02 SUB1.OBJ;1
                          032544 000624 00404. SUB2   FORV02 SUB1.OBJ;1
$DATAP:(RW,D,LCL,REL,CON) 033370 000010 00008.
                          033370 000000 00000. SUB2   FORV02 SUB1.OBJ;1
                          033370 000010 00008. SUB2   FORV02 SUB1.OBJ;1
$$ALVC:(RW,D,LCL,REL,CON) 033400 000000 00000.
.$$$$.:(RW,D,GBL,REL,OVR) 141442 000000 00000.
                          141442 000000 00000. SUB2   FORV02 SUB1.OBJ;1
                          141442 000000 00000. SUB2   FORV02 SUB1.OBJ;1


GLOBAL SYMBOLS:

SUB2   032530-R
```

**Example 9-2 Memory Allocation File for CALC.TSK;6 (continued)**

```
CALC.TSK;6    MEMORY ALLOCATION MAP  TKB D28          PAGE 8
SUB2                  3-JUL-78   10:52


*** SEGMENT: SUB2


R/W MEM  LIMITS: 032530 033567 001040 00544.
DISK BLK LIMITS: 000044 000045 000002 00002.


MEMORY ALLOCATION SYNOPSIS:

SECTION                                          TITLE  IDENT  FILE
-------                                          -----  -----  ----
. BLK.:(RW,I,LCL,REL,CON)  032530 000000 00000.
ADTA  :(RW,D,GBL,REL,OVR)  001002 002260 01200.
                           001002 002260 01200.  SUB1   FORV02 SUB2.OBJ;1
DTA   :(RW,D,GBL,REL,OVR)  140000 001442 00802.
                           140000 001442 00802.  SUB1   FORV02 SUB2.OBJ;1
OTS$I :(RW,I,LCL,REL,CON)  032530 000154 00108.
$CODE :(RW,I,LCL,REL,CON)  032704 000032 00026.
                           032704 000000 00000.  SUB1   FORV02 SUB2.OBJ;1
                           032704 000000 00000.  SUB1   FORV02 SUB2.OBJ;1
                           032704 000032 00026.  SUB1   FORV02 SUB2.OBJ;1
$DATA :(RW,D,LCL,REL,CON)  032736 000622 00402.
                           032736 000000 00000.  SUB1   FORV02 SUB2.OBJ;1
                           032736 000622 00402.  SUB1   FORV02 SUB2.OBJ;1
$DATAP:(RW,D,LCL,REL,CON)  033560 000010 00008.
                           033560 000000 00000.  SUB1   FORV02 SUB2.OBJ;1
                           033560 000010 00008.  SUB1   FORV02 SUB2.OBJ;1
$$ALVC:(RW,D,LCL,REL,CON)  033570 000000 00000.
.$$$$.:(RW,D,GBL,REL,OVR)  141442 000000 00000.
                           141442 000000 00000.  SUB1   FORV02 SUB2.OBJ;1
                           141442 000000 00000.  SUB1   FORV02 SUB2.OBJ;1


GLOBAL SYMBOLS:

SUB1   032704-R



*** TASK BUILDER STATISTICS:

    TOTAL WORK FILE REFERENCES: 22392.
    WORK  FILE  READS: 0.
    WORK  FILE WRITES: 0.
    SIZE OF CORE POOL: 16010. WORDS (62. PAGES)
    SIZE OF WORK FILE: 7680. WORDS (30. PAGES)

    ELAPSED TIME:00:00:20
```

## 9.5.4 Shared Global Areas with Memory-Resident Overlays

It is possible for an SGA to contain memory-resident overlays. The whole SGA will be loaded, but each task which binds to it will be mapped only to the parts which that task currently requires.

**Note: To run a task which uses an SGA that contains overlays you must have memory management privilege (see the IAS System Management Guide).**

If it is to contain memory-resident overlays, the SGA must be built using the LINK qualifier RESIDENT_OVERLAYS or the TKB switch /RO. The user must define the overlay structure through an ODL file prepared in the conventional manner. The Task Builder does not include the overlay data base (segment descriptors, autoload vectors) or Overlay Runtime System within the region image. Instead, this data base becomes a part of the symbol definition file that is linked to the referencing task. This means that routines within an overlay segment of an SGA cannot be called from within the SGA using the autoload mechanism, although the manual load mechanism can be used.

When the referencing task is built, the following is automatically included in its root segment:

1 The data base.

2 Global references to overlay support routines residing in the system object module library.

The symbol table file contains global definitions for only those symbols that are defined or referenced in the root segment of the shared region. Such symbols can consist of:

1 Actual entry points to routines and data elements that are in the root.

2 Autoload vector addresses that point to real definitions within a memory-resident overlay.

3 Actual definitions of symbols defined in a memory-resident overlay and referenced in the root.

The user can force the inclusion of global references in the root segment of the SGA by means of the GBLREF option. In this way, the necessary autoload vectors and definitions can be generated without explicitly including such references in an object module. The syntax of the option is:

```
GBLREF=name
```

where "name" consists of 1 to 6 characters selected from the Radix-50 character set. If the definition resides within an autoloadable segment, then an autoload vector will be built and included in the symbol table file. If the definition is not autoloadable, the real value is obtained and defined in the root segment.

No global symbol appears in the symbol table file unless:

1 It is defined in the root segment.

2 It is referenced in the root segment and defined elsewhere in the overlay structure.

The procedure for creating the overlaid SGA can be summarized as follows:

1 Define an overlay structure containing only memory-resident overlays.

2 Include a GBLREF option, or provide in the root segment, a module containing the appropriate global references for defining entry points within those overlay segments for which autoload vectors and global definitions will be generated.

These processes are illustrated in the following example. The SGA to be constructed consists of shareable code that resides within the overlay structure defined below:

```
.ROOT A-!(*B,C-*D)
.NAME A
.END
```

Root segment A contains no code or data and has a length of 0. All executable code exists within memory-resident overlay segments composed of files B.OBJ, C.OBJ, and D.OBJ, containing global entry points B, C, and D.

The task image, map, and symbol table files are generated using the following Task Builder commands:

```
PDS>LINK/NOHEAD/MAP/SYMBOL/OVERLAY:A/OPTIONS/RESID
OPTIONS?GBLREF=B,C,D
OPTIONS?UNITS = 0
OPTIONS?STACK = 0
OPTIONS?  /
```

or

```
TKB>A/RO/-HD,A,SY:A=A/MP
ENTER OPTIONS:
TKB>GBLREF=B,C,D
TKB>STACK = 0
TKB>UNITS = 0
TKB>/
```

References to entry points B, C, and D are inserted in the root segment, and subsequently appear in the symbol table file as definitions.

The definition for symbol C is resolved directly to the actual entry point. The definitions for symbols B and D are resolved to autoload vectors that are included in each referencing task. Unlike overlays that reside in the task image, each autoload vector in the SGA is allocated in every referencing task, whether or not such entry points are called during task execution. Only those global symbols defined or referenced in the root segment of the SGA appear in the symbol table file.

The symbol table file also contains the data base required by the Overlay Runtime System, in relocatable object module format. This data base includes:

1 All autoload vectors

2 Segment tables linked as described in Appendix C, Section C.7

3 Window descriptors

4 A single region descriptor

The overlay structure, as reflected in the segment table linkage, is preserved, and conveyed to the referencing task by the STB file; thus path-loading for the SGA can occur exactly as it does within a task. Aside from address space restrictions, there is no limitation on the overlay structures that can be defined for an SGA.

The following restrictions apply to shared regions existing as memory- resident overlays:

1 An SGA cannot use the autoload facility to reference memory-resident overlays within itself or any other region. If each segment is uniquely named, overlays can be mapped through the manual load facility.

2 Named p-sections in an SGA overlay cannot be referenced by the task. If reference to the storage is required, such sections must be included in the root segment of the region (with resultant loss of virtual address space).

3 Unlike task-resident overlays, the number of autoload vectors is independent of the entry points actually referenced. The maximum number of vectors will be allocated within each referencing task. In some cases the size of the allocation may be large.

4 There is an overhead of six instructions per autoload call, even when the segment is mapped.

5 Overlaid SGAs cannot be position independent.

As implied by the previous items, great care must be exercised if an efficient memory-resident overlay structure for library routines such as the FORTRAN IV OTS is to be implemented.

# 10 Supervisor-Mode Libraries

A supervisor-mode library is a resident library that doubles a user task's virtual address space by mapping the instruction space of the processor's supervisor mode. Supervisor-mode libraries are available only on PDP-11/44 and PDP-11/70 systems.

## 10.1 Introduction

A call from within a user task to a subroutine within a supervisor-mode library causes the processor to switch from user to supervisor mode. The user task transfers control to a mode-switching vector that TKB includes within the task. The mode-switching vector performs the mode switch and then transfers control to the called subroutine within the supervisor-mode library. The library routine executes with the processor in supervisor mode. When the library routine finishes executing, it transfers control to a completion routine within the library. The completion routine mode switches the processor back to user mode. The user task continues executing with the processor in user mode at the return address on the stack. This process recurs whenever the user task calls a subroutine in the supervisor-mode library.

## 10.2 Mode-Switching Vectors

In a task that links to a supervisor-mode library, TKB includes a four-word, mode-switching vector in the user task's address space for each entry point referenced of a subroutine in the library.

The following shows the contents of a mode-switching vector:

```
MOV #COMPLETION-ROUTINE,-(SP)
CSM #SUPERVISOR-MODE-ROUTINE ADDRESS
```

**NOTE: When mode switching from user to supervisor mode, all registers of the referencing task are preserved. All condition codes in the PS saved on the stack are cleared and must be restored by the completion routine.**

## 10.3 Completion Routines

After the subroutine finishes executing, its RETURN statement transfers control to a completion routine that mode-switches from the supervisor to user mode. The completion routine returns program control back to the referencing task at the instruction after the call to the subroutine. SYSLIB has two completion routines.

- $CMPCS restores only the carry bit in the user-mode PS.
- $CMPAL restores all the condition code bits in the user-mode PS.

## 10.4 Restrictions on the Contents of Supervisor-Mode Libraries

The following restrictions are placed on the contents of a supervisor-mode library:

- Only subroutines using the form JSR PC, x should be used within the library.
- The library must not contain subroutines that use the stack to pass parameters.
- If both the library and the referencing task link to a subroutine from SYSLIB, then the entry point name of the subroutine must be excluded from the .STB file for the library.
- The library must not contain data of any kind (even R/O) because the user supervisor D-space APRs map the user task by default. This includes user data, buffers, I/O status blocks, and directive parameter blocks (only the $S directive form can be used, because the DPB for this form is pushed onto the user stack at run time).

## 10.5 Supervisor-Mode Library Mapping

Supervisor-mode libraries are mapped with the supervisor I-space APRs. Supervisor D-space APRs map the user task.

Supervisor D-space APRs are copies of user I-space APRs, which map the entire user task. This gives the library access to data within the user task. Figure 10–1 illustrates this mapping.

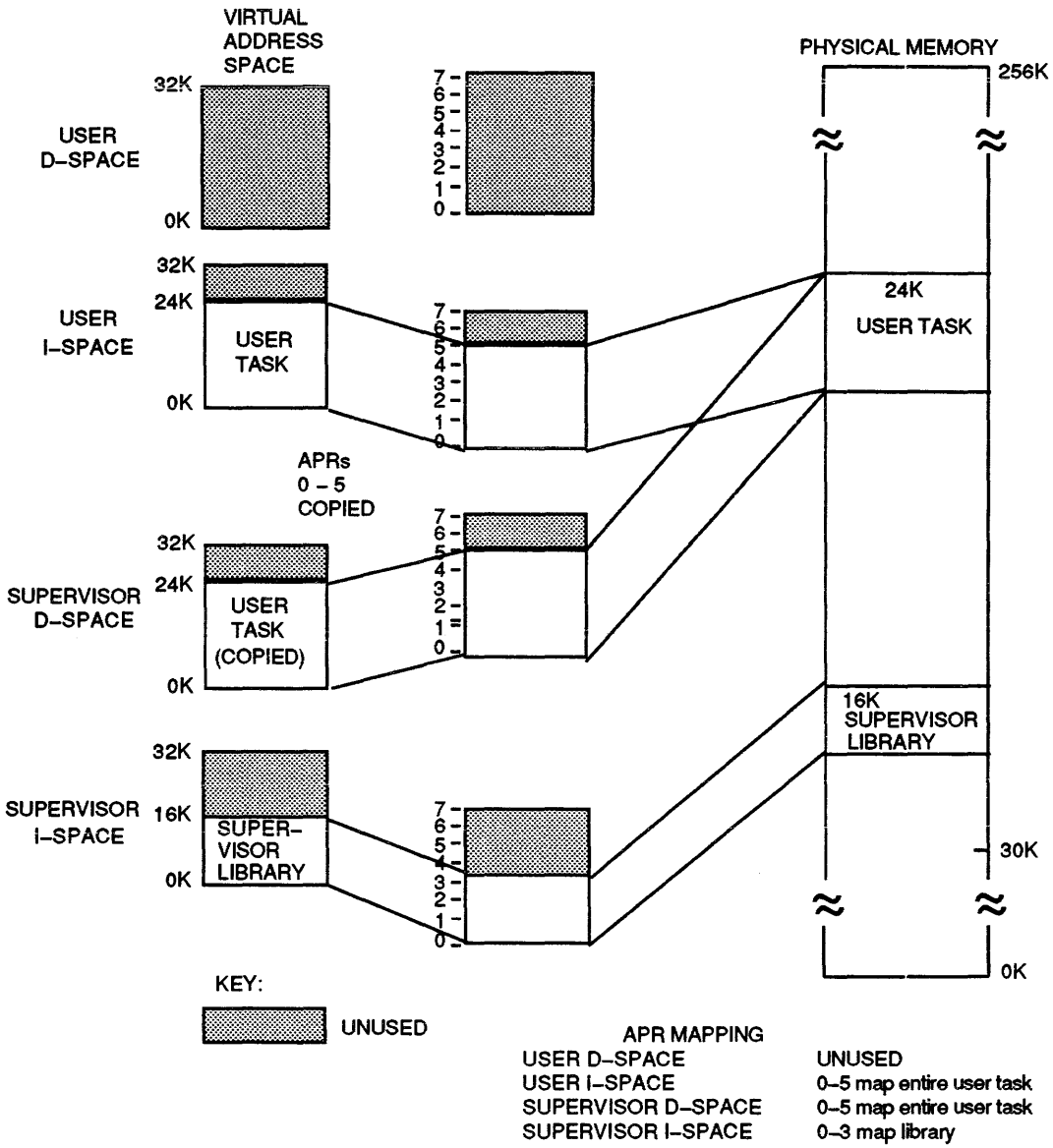
## 10.6 Building and Linking to Supervisor-Mode Libraries

Building and linking to a supervisor-mode library is essentially the same as building and linking to a conventional resident library (discussed in Chapter 6). When you build a supervisor-mode library using the TKB command line, you suppress the header by attaching /-HD to the task image file. If you use LINK, you use the /NOHEAD qualifier in the LINK command line. During option input, you suppress the stack area by specifying STACK=0. You specify the partition where the library is to reside and, optionally, the base address and length of the library with the PAR option.

### 10.6.1 Relevant TKB Options

Use the following options to build and reference supervisor-mode libraries:

**Figure 10–1 Mapping of a 24K Conventional User Task That Links to a 16K Supervisor-Mode Library**



| | |
|---|---|
| CMPRT | Indicates that you are building supervisor-mode library and specifies the name of the completion routine. |
| RESSUP (SUPLIB) | Indicates that your task references a supervisor-mode library. |
| GBLXCL | Excludes a global symbol from the .STB file of the supervisor-mode library. |

These options are discussed briefly below and are fully documented in Chapter 5.

### 10.6.2 Building The Library

You indicate to the TKB that you are building a supervisor-mode library with the CMPRT option. The argument for this option identifies the entry symbol of the completion routine. When the TKB processes this option, it places the completion routine entry point in the library's STB file. To exclude a global symbol from the library's .STB file, you specify the name of the global symbol as the argument of the GBLXCL option. You must exclude from the .STB file of a supervisor-mode library any symbol defined in the library that represents the following:

- An entry point to a subroutine that uses the stack to pass parameters
- An entry point to a subroutine mapped in user mode that the referencing user task calls

### 10.6.3 Building the Referencing Task

When you build a task that references a supervisor-mode library, use the RESSUP option if you are referencing a user-owned, supervisor-mode library and SUPLIB if you are referencing a system-owned, supervisor-mode library. (Like the RESLIB and LIBR options for linking to conventional libraries, RESSUP and SUPLIB are functionally the same.) The arguments for these options are:

- The filespec (RESSUP option) or name (SUPLIB) of the library to be referenced
- A switch that tells TKB whether to use system-supplied vectors to perform mode switching from user to supervisor mode.
- For position-independent libraries, the first available supervisor-mode I-space APR that you want to map the library.

### 10.6.4 Mode Switching Instruction

Mode switching occurs with a new instruction available on the 11/44 and emulated by the Executive on the 11/70. Throughout the remainder of the chapter, supervisor-mode libraries are referred to as CSM (change supervisor mode) libraries.

## 10.7 CSM Libraries

This section discusses how you build and link to CSM libraries. It also shows an extended example of building and linking to a CSM library and explains the context-switching vectors and completion routines for CSM libraries.

## 10.7.1 Building A CSM Library

You indicate to the Task Builder that you are building a CSM library by specifying the name of the completion routine as the argument for the CMPRT option. This option places the name of the completion routine into the library's .STB file. Link the completion routine, either $CMPAL or $CMPCS, located in LB:[1,2]SYSLIB.OLB, as the first input file. Although the completion routines are located in SYSLIB (which is ordinarily referenced by default), you must explicitly indicate it and link it as the first input file. You must also specify in the PAR option a 0 base for the partition where the library resides. These two steps locate the completion routine at virtual 0 of the library's virtual address space.

You specify the name of any global symbols that you would like to exclude from the library's .STB file as the argument to the GBLXCL option. You must exclude from the .STB file of a supervisor-mode library any symbol defined in the library that represents the following:

- An entry point to a subroutine that uses the stack to pass parameters
- An entry point to a subroutine mapped in user mode that the referencing user task calls

A sample TKB command sequence for building a CSM library in UFD [301,55] on SY: follows:

```
TKB> CSM/-HD/LI/PI,CSM/MA,CSM=
TKB> LB:[[1,2]]SYSLIB/LB:CMPAL,SY:[[301,55]]CSM
TKB> /
Enter Options:
TKB> STACK=0
TKB> PAR=GEN:0:2000
TKB> CMPRT=$CMPCS
TKB> GBLXCL=$SAVAL
TKB> //
>
```

Or, you can use the following LINK command sequence to build the same library:

```
> LINK/TAS:CSM/NOH/SHARE:LIB/CODE:PIC/MAP:CSM/SYS/SYM:CSM/OPT -
-> LB:[1,2]SYSLIB/INCLUDE:CMPAL,SY:[301,55]CSM
Option?  STACK=0
Option?  PAR=GEN:0:2000
Option?  CMPRT=CMPCS
Option?  GBLXCL=$SAVAL
Option?  RET
>
```

The library is built without a header or stack, like all shared regions. It is position independent and has only one program section named .ABS. The /LI switch in TKB or the /CODE:PIC qualifier in LINK switch accomplishes this, eliminating program section name conflicts between the library and the referencing task. The completion routine module of SYSLIB, CMPAL, is specified first in the input line. The library runs in partition GEN at 0 and is not more than 1K. These are two aspects of building supervisor-mode libraries specific to CSM libraries: the completion routine must be linked first, and must reside at virtual 0. Why the CSM library must reside at virtual 0 is discussed in Section 9.5.

The CMPRT option specifies the global symbol $CMPCS, which is the entry point of the completion routine. Note that the SYSLIB module name is "CMPCS" and its corresponding global symbol is "$CMPCS".

The GBLXCL option excludes $SAVAL from the library's .STB file because the user task must reference a copy of $SAVAL that is mapped with user mode APRs.

## 10.7.2 Linking To A CSM Library

If your task links to a user-owned CSM library, you use the RESSUP option. If your task links to a system-owned CSM library, you use the SUPLIB option. These options tell TKB that the task is to link to a supervisor-mode library. The option takes up to three arguments:

- The filespec (RESSUP option) or name (SUPLIB option) of the library
- A switch that tells the TKB whether to use system-supplied, mode-switching vectors
- For position-independent libraries, an APR must be APR 0 so that the library's completion routine is mapped at virtual 0.

This information enables the TKB to find the .STB file for the CSM library, include a four-word, mode-switching vector within the user task for each call to a subroutine within the library, and correctly map the library at virtual 0 in the library image.

The following examples of TKB and LINK command sequences build a task named REF, which references the library SUPER that you built in the previous section:

```
TKB> REF,REF=REF
TKB> /
Enter Options:
TKB> RESSUP=SUPER/SV:0
TKB> //
>

> LINK/TAS/MAP/OPT REF
Option?  RESSUP=SUPER/SV:0
Option?  RET
>
```

This sequence tells TKB to include in the logical address space of REF a user-owned, supervisor-mode library named SUPER. TKB includes a four-word, mode switching vector within the user task for each call to a subroutine within the library. The CSM library is position independent and is mapped with APR 0.

## 10.7.3 Example CSM Library And Linking Task

This example shows you the code and maps and the TKB and LINK command sequences for building and linking to a CSM library that contains no data in a system without user data space. Example 10–1 shows the code for the library SUPER, and Example 10–2 shows its accompanying map. Example 10–3 shows the code for the completion routine $CMPCS that is linked in to SUPER from SYSLIB. Example 10–4 shows the code for referencing task TSUP, and Example 10–5 shows its accompanying map.

**Example 10–1 Code for SUPER.MAC**

```
.TITLE SUPER
.IDENT /01/
```

**Example 10–1 Cont'd on next page**

**Example 10–1 (Cont.) Code for SUPER.MAC**

```
SORT::
  CALL $SAVAL    ; SAVE ALL REGISTERS
  TST  (R5)+     ; SKIP OVER NUMBER OF ARGUMENTS
  MOV  (R5)+,R0  ; GET ADDRESS OF LIST
  MOV  (R5)+,R4  ; GET ADDRESS OF LENGTH OF LIST
  MOV  (R4),R4   ; GET LENGTH OF LIST
  BEQ  40$       ; IF NO ARGUMENTS
  MOV  R0,R5     ;
  DEC  R4        ;
10$:
  MOV  R5,R0     ; COPY
  MOV  R4,R3     ; COPY LENGTH OF LIST
20$:
  TST  (R0)+     ; MOVE POINTER TO NEXT ITEM
  CMP  (R5),(R0)  ; COMPARE ITEMS
  BLE  30$       ; IF LE IN CORRECT ORDER
  MOV  (R5),R2   ; SWAP ITEMS
  MOV  (R0),(R5)  ;
  MOV  R2,(R0)   ;
30$:
  DEC  R3        ; DECREMENT LOOP COUNT
  BGE  20$       ; IF NE LOOP
  DEC  R4        ; DECREMENT
  BLE  40$       ; IF EQ SORT COMPLETED
  TST  (R5)+     ; GET POINTER TO NEXT ITEM TO BE COMPARED
  BR  10$
40$:
  RETURN

SEARCH::
  CALL $SAVAL    ; SAVE ALL THE REGISTERS
  CMP  #4,(R5)+  ; FOUR ARGUMENTS?
  BNE  20$       ; IF NE NO
  MOV  (R5)+,R0  ; GET ADDRESS OF NUMBER TO LOCATE
  MOV  (R5)+,R1  ; ADDRESS OF LIST SEARCHING
  MOV  (R5)+,R2  ; GET ADDRESS OF LENGTH OF LIST
  MOV  (R2),R2   ; GET LENGTH OF LIST
  BEQ  20$       ; IF NO ARGUMENTS
  MOV  (R5),R5   ; ADDRESS OF RETURNED VALUE
  MOV  R2,R3     ; COPY LENGTH
10$:
  CMP  (R0),(R1)+  ; IS THIS THE NUMBER?
  BEQ  30$       ; IF EQ YES
  BMI  20$       ; IF MI NUMBER NOT THERE
  DEC  R2        ; DECREMENT LOOP COUNT
  BNE  10$       ; IF NE NOT AT END OF LIST
20$:
  MOV  #-1,(R5)  ; END OF LIST PASS BACK ERROR
  RETURN
30$:
  SUB  R2,R3     ; NUMBER FOUND - GET INDEX INTO LIST
  INC  R3        ;
  MOV  R3,(R5)   ; RETURN INDEX
  RETURN
  .END
```

**Example 10–2 Memory Allocation Map for SUPER**

```
SUPER.TSK;1 Memory allocation map TKB M40.10  Page 1
     29-DEC-82 15:04


Partition name : GEN
Identification : 0203
Task UIC  : [301,55]
Task attributes: -HD,PI
Total address windows: 1.
Task image size : 160. words
Task address limits: 000000 000473
R-W disk blk limits: 000002 000002 000001 00001.

*** Root segment: CMPAL

R/W mem limits: 000000 000473 000474 00316.
Disk blk limits: 000002 000002 000001 00001.

Memory allocation synopsis:

Section           Title Ident File
-------           ----- ----- ----

. BLK.:(RW,I,LCL,REL,CON)  000000 000474 00316.
       000000 000136 00094. CMPAL 0203 SYSLIB.OLB;6
       000136 000136 00094. CMPAL 0203 SYSLIB;OLB;6
       000274 000136 00094. SUPER 01 SUPER.OBJ;3
       000432 000042 00034. SAVAL 00  SYSLIB.OLB;6

Global symbols:

SEARCH 000352-R SORT 000274-R $CMPAL 000022-R $CMPCS 000110-R $SAVAL 000432-R

*** Task builder statistics:

 Total work file references: 320.
 Work file reads: 0.
 Work file writes: 0.
 Size of core pool: 6988. words (27. pages)
 Size of work file: 1024. words (4. pages)

 Elapsed time:00:00:04
```

**Example 10–3 Completion Routine, $CMPCS, from SYSLIB.OLD**

```
    .TITLE CMPAL
    .IDENT /0204/
```

**Example 10–3 Cont'd on next page**

**Example 10–3 (Cont.) Completion Routine, $CMPCS, from SYSLIB.OLD**

```
;
;    COPYRIGHT (c) 1983 BY
;  DIGITAL EQUIPMENT CORPORATION, MAYNARD
;  MASSACHUSETTS. ALL RIGHTS RESERVED.
;
; THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED
; AND COPIED ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE
; AND WITH THE INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS
; SOFTWARE OR ANY OTHER COPIES THEREOF, MAY NOT BE PROVIDED OR
; OTHERWISE MADE AVAILABLE TO ANY OTHER PERSON. NO TITLE TO AND
; OWNERSHIP OF THE SOFTWARE IS HEREBY TRANSFERED.
;
; THE INFORMATION IN THIS DOCUMENT IS SUBJECT TO CHANGE WITHOUT
; NOTICE AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL
; EQUIPMENT CORPORATION.
;
; DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF
; ITS SOFTWARE ON EQUIPMENT THAT IS NOT SUPPLIED BY DIGITAL.
;
  .ENABL LC

;
;
; This module supports the "new" transfer vector format generated by
; the taskbuilder for entering super mode libraries. This format
; optimized for speed and size and supports user data space tasks.
;
; The CSM dispatcher routine and the standard completion routines,
; $CMPAL and $CMPCS are included in this module due to the close
; interaction between them.
;

;
; **-CSM Dispatcher-Dispatch CSM entry
;
; This module must be linked at virtual zero in the supervisor mode
; library. It is entered via a four word transfer vector of the form:
;
;  MOV  #completion-routine,-(SP)
;  CSM  #routine
;
; Note: Immediate mode emulation of the CSM instruction is required
;  in the executive.
;
; The CSM instruction transfers control to the address contained in
; supervisor mode virtual 10. At this point the stack is the following:
;
;    (SP) routine address
;   2(SP) PC (past end of transfer vector)
;   4(SP) PS with condition codes cleared
;   6(SP) Completion-routine address
;   10(SP) Return address
;
; A routine address of 0 is special cased to support return to
; supervisor mode from a user mode debugging aid (ODT). In this case
; stack is the following:
;
```

**Example 10–3 Cont'd on next page**

**Example 10–3 (Cont.) Completion Routine, $CMPCS, from SYSLIB.OLD**

```
;    (SP) zero
;   2(SP) PC from CSM to be discarded
;   4(SP) PS from CSM to be discarded
;   6(SP) Super mode PC supplied by debugger
;   10(SP) Super mode PS supplied by debugger
;
; To allow positioning at virtual zero, this code must be in the blank
; PSECT that is first in the TKBs PSECT ordering.

   .PSECT
   .ENABL LSB

; Debugger return to super mode entry. Must start at virtual zero

   CMP   (SP)+,(SP)+   ; Clean off PS and PC from CSM

;
: **-$SRTI-SUPER mode RTI
;
; This entry point performs the necessary stack management to allow
; an RTI from super mode to either super mode or user mode.
; The is as required for an RTI:
;
;   (SP) Super mode PC
;   2(SP) Super mode PS

$SRTI:: TST 2(SP)   ; Returning to user mode?
   BR   70$    ; Join common code

; CSM transfer address, this word must be at virtual 10 in super mode

   .WORD CSMSVR    ; CSM dispatcher entry

; Dispatch CSM entry

CSMSVR: MOV   6(SP),2(SP)   ; Set completion routine address for RETURN
   JMP   @(SP)+    ; Transfer to super mode library routine

;
; **-$CMPAL-Completion routine that sets up NZVC in the PS
;
; Copy all condition codes to stacked PS. Current stack:
;
;   (SP) PS with condtion codes cleared
;   2(SP) Completion routine address (to be discarded)
;   4(SP) Return address
;
```

**Example 10–3 Cont'd on next page**

**Example 10–3 (Cont.) Completion Routine, $CMPCS, from SYSLIB.OLD**

```
$CMPAL::BPL  40$    ;
  BNE  20$    ;
  BVC  10$    ;
  BIS  #16,(SP)  ; Set NZV
  BR  $CMPCS   ;
10$: BIS  #14,(SP)  ; Set NZ
  BR  $CMPCS   ;
20$: BVC  30$    ;
  BIS  #12,(SP)  ; Set NV
  BR  $CMPCS   ;
30$: BIS  #10,(SP)  ; Set N
  BR  $CMPCS   ;
40$: BNE  60$    ;
  BVC  50$    ;
  BIS  #6,(SP)   ; Set ZV
  BR  $CMPCS   ;
50$: BIS  #4,(SP)   ; Set Z
  BR  $CMPCS   ;
60$: BVC  $CMPCS   ;
  BIS  #2,(SP)   ; Set V

;
; **-$CMPCS-Completion routine that sets up only C in the PS
;
; Copy only carry to stacked PS. Current stack:
;
;  (SP) PS with condtion codes cleared
;  2(SP) Completion routine address (to be discarded)
;  4(SP) Return address
;

$CMPCS::ADC  (SP)   ; Set up carry
  MOV  4(SP),2(SP)  ; Setup return address for RTT
  MOV  (SP)+,2(SP)  ; And PS. Returning to super mode?
70$: BPL  80$    ; If PL yes
  MOV  #6,-(SP)  ; Number of bytes for (SP), PS, and PC
  ADD  SP,(SP)   ; Compute clean stack value
  MTPI SP    ; Set up previous stack pointer
80$: RTT      ; Return to previous mode and caller

  .DSABL LSB

  .END
```

**Example 10–4 Code for TSUP.MAC**

```
   .TITLE TSUP
   .IDENT /01/

   .MCALL QIOW$,DIR$,QIOW$S
WRITE: QIOW$ IO.WVB,5,1,,,,<OUT,,40>
READIN: QIOW$ IO.RVB,5,1,,,,<OUT,5>

IARRAY: .BLKW 12.
LEN: .BLKW 1
IART: .BLKW 1
INDEX: .WORD 0
OUT: .BLKW 100.
ARGBLK:
EDBUF: .BLKW 10.

FMT1: .ASCIZ /%2SARRAY(%D)=/
FMT2: .ASCIZ /%N%2SNUMBER TO SEARCH FOR?/
FMT3: .ASCIZ /%N%2S%D WAS FOUND IN ARRAY(%D)/
FMT4: .ASCIZ /%N%2S%D WAS NOT IN ARRAY/
FMT5: .ASCIZ /%2SARRAY(%D)=%D/

   .EVEN
START:
   MOV  #IARRAY,R0  ; GET ADDRESS OF ARRAY
   MOV  #10,R1   ; SET LENGTH OF ARRAY
5$:
   CLR  (R0)+   ; INITIALIZE ARRAY
   DEC  R1    ; LOOP
   BNE  5$
   MOV  #IARRAY,R0  ;
   MOV  #INDEX,R2
10$:
   MOV  #FMT1,R1  ; FORMAT SPECIFICATION (ADDRESS
         ; OF INPUT STRING)
   MOV  (R2),EDBUF  ; GET INDEX
   INC  EDBUF   ;
   CALL PRINT   ; PRINT MESSAGE
   CALL READ    ; READ INPUT
   MOV  IART,(R0)+  ; PUT BINARY KEYBOARD INPUT INTO ARRAY
   BEQ  20$     ; ZERO MARKS END OF INPUT
   INC  (R2)    ;
   CMP  (R2),#10.
   BNE  10$     ; IF NE YES
20$:
   MOV  (R2),LEN  ; CALCULATE LENGTH OF ARRAY
   MOV  #ARGBLK,R5  ; GET ADDRESS OF ARGUMENT BLOCK
   MOV  #2,(R5)+  ; NUMBER OF ARGUMENTS
   MOV  #IARRAY,(R5)+ ; PUT ADDRESS OF ARRAY
   MOV  #LEN,(R5)  ;
   MOV  #ARGBLK,R5  ;
   CALL SORT    ; SORT ARRAY
;+
;Task Builder replaced call to SORT subroutine in SUPLIB with 4-word
;context switching vector. Flow of control switches to SUPLIB via
;the vector and back via the completion routine $CMPCS. TSUP
;continues excuting at the next instruction.
;-
   CLR  R2     ;
```

**Example 10–4 Cont'd on next page**

**Example 10–4 (Cont.) Code for TSUP.MAC**

```
    MOV  #IARRAY,R0  ; GET ARRAY ADDRESS
30$:
    INC  R2     ; INCREMENT INDEX
    MOV  R2,EDBUF  ; GET INDEX FOR PRINT
    MOV  (R0)+,EDBUF+2 ; GET CONTENTS OF ARRAY
    MOV  #FMT5,R1  ; GET ADDRESS OF FORMAT SPECIFICATION
    CALL PRINT   ;
    CMP  R2,LEN    ; MORE TO PRINT?
    BLT  30$     ; IF LE YES
    MOV  #FMT2,R1  ; GET ADDRESS OF FORMAT SPECIFICATION
    CALL PRINT   ; OUTPUT MESSAGE
    CALL READ    ; READ RESPONSE
    MOV  #ARGBLK,R5  ;
    MOV  #4,(R5)+  ; SET NUMBER OF ARGUMENTS
    MOV  #IART,(R5)+  ; SET ADDRESS OF NUMBER LOOKING FOR
    MOV  #IARRAY,(R5)+ ; SET ADDRESS OF ARRAY
    MOV  #LEN,(R5)+  ; SET ADDRESS OF LEN OF ARRAY
    MOV  #INDEX,(R5)  ; ADDRESS OF RESULT
    MOV  #ARGBLK,R5  ;
    CALL SEARCH   ; SEARCH FOR NUMBER IN IART
;
;Call to SUPLIB for SEARCH subroutine.
;
    TST  INDEX   ; WAS NUMBER FOUND?
    BLT  40$     ; IF LT NO
    MOV  IART,EDBUF  ; GET NUMBER LOOKING FOR
    MOV  INDEX,EDBUF+2 ; GET ARRAY NUMBER
    MOV  #FMT3,R1  ; GET FORMAT ADDRESS
    CALL PRINT   ;
    BR  100$   ; DONE
40$:
    MOV  #FMT4,R1  ; GET FORMAT ADDRESS
    MOV  IART,EDBUF  ; GET NUMBER
    CALL PRINT
100$:
    CALL $EXST   ; EXIT WITH STATUS

PRINT:
    CALL $SAVAL   ; SAVE ALL REGISTERS
    MOV  #OUT,R0   ; ADDRESS OF OUTPUT BLOCK
    MOV  #EDBUF,R2  ; START ADDRESS OF ARGUMENT BLOCK
    CALL $EDMSG   ; FORMAT MESSAGE
    MOV  R1,WRITE+Q.IOPL+2 ; PUT LENGTH OF OUTPUT
          ; BLOCK INTO PARAMETER BLOCK
    DIR$ #WRITE   ; WRITE OUTPUT BLOCK
    RETURN

READ:
    CALL $SAVAL   ; SAVE ALL REGISTERS
    DIR$ #READIN ; READ REQUEST
    MOV  #OUT,R0 ; GET KEYBOARD INPUT
    CALL $CDTB   ; CONVERT KEYBOARD INPUT TO BINARY
    MOV  R1,IART   ; PUT INPUT INTO BUFFER
    RETURN

    .END START
```

**Example 10–5 Memory Allocation Map for TSUP**

```
TSUP.TSK;1 Memory allocation map TKB M40.10  Page 1
     29-DEC-82 15:01


Partition name : GEN
Identification : 01
Task UIC  : [301,55]
Stack limits: 000274 001273 001000 00512.
PRG xfr address: 002130
Total address windows: 2.
Task image size : 1344. words
Task address limits: 000000 005133
R-W disk blk limits: 000002 000007 000006 00006.

*** Root segment: TSUP

R/W mem limits: 000000 005133 005134 02652.
Disk blk limits: 000002 000007 000006 00006.

Memory allocation synopsis:

Section           Title Ident File
-------           ----- ----- ----

. BLK.:(RW,I,LCL,REL,CON)  001274 002334 01244.
       001274 001234 00668. TSUP 01  TSUP.OBJ;22
CMPAL :(RW,I,LCL,REL,CON)  000000 000474 00316.
PUR$D :(RO,I,LCL,REL,CON)  003630 000076 00062.
PUR$I :(RO,I,LCL,REL,CON)  003726 000752 00490.
$$RESL:(RO,I,LCL,REL,CON)  004700 000212 00138.
$$SLVC:(RO,I,LCL,REL,CON)  005112 000020 00016.

TSUP.TSK;1 Memory allocation map TKB M40.10  Page 2
     29-DEC-82 15:01


*** Task builder statistics:

 Total work file references: 2477.
 Work file reads: 0.
 Work file writes: 0.
 Size of core pool: 6988. words (27. pages)
 Size of work file: 1024. words (4. pages)

 Elapsed time:00:00:05
```

TSUP prompts you to enter numbers at your terminal. It calls a subroutine in SUPER to sort the numbers. Then it displays the numbers you entered as array entries and prompts you to request a number to search for. TSUP calls a subroutine in SUPERLIB to search for the number. Finally, TSUP indicates at your terminal either that the number was not found or the array location where the number is stored.

### Building SUPER

To build SUPER in UFD [301,55] on SY:, use the following TKB or LINK command sequence:

```
TKB> SUPER/-HD/LI/PI,SUPER/MA,SUPER=
TKB> LB:[[1,2]]SYSLIB/LB:CMPAL,SY:[[301,55]]SUPER
TKB> /
Enter Options:
TKB> STACK=0
TKB> PAR=GEN:0:2000
TKB> CMPRT=$CMPCS
TKB> GBLXCL=$SAVAL
TKB> //
>
```

```
> LINK/TAS:SUPER/NOH/SHARE:LIB/CODE:PIC/MAP:SUPER/SYS/SYM:SUPER/OPT
-
-> LB:[1,2]SYSLIB/INC:CMPAL,SY:[301,55]SUPER
Option?  STACK=0
Option?  PAR=GEN:0:2000
Option?  CMPRT=$CMPCS
Option?  GBLXCL=$SAVAL
Option?  RET
>
```

SUPER is built without a header or stack. It is position independent and has only one program section, named .BLK. The /LI switch or /SHARE:LIB qualifier eliminates program section name conflicts between the library and the referencing task.

The completion routine module of SYSLIB, CMPAL, is specified first in the input line. The library runs in partition GEN at 0 and is not more than 1K.

The GBLXCL option excludes $SAVAL from the library's .STB file. You exclude $SAVAL from the .STB file because the referencing task, TSUP, also calls $SAVAL. If TSUP finds $SAVAL in the .STB file of SUPER, it does not link a separate copy of $SAVAL into its task image from SYSLIB. If TSUP cannot link to a copy of $SAVAL that is mapped through user APRs, the TSUP would call $SAVAL as a subroutine residing within the supervisor-mode library, but without the necessary mode-switching vector and completion routine support. This option forces TKB to link $SAVAL from SYSLIB into the task image for TSUP.

The memory allocation map shows the following:

- SUPER begins at virtual 0.
- The completion routine, $CMPAL, is linked into the library from SYSLIB at virtual 0.
- The entry point $CMPAL is located at virtual 22, SEARCH is located at 35, and sort is located at 274. All of these entry points are relocatable.

### Building TSUP

Use the following TKB or LINK command sequence to build a task, TSUP, that links to SUPER:

```
TKB> TSUP,TSUP=TSUP
TKB> /
Enter Options:
TKB> RESSUP=SUPER/SV:0
TKB> //
```

```
> LINK/TAS/MAP/OPT SUPER
Option?  RESSUP=SUPER/SV:0
Option?  RET
>
```

These two command sequences tell TKB to include in the logical address space of TSUP a user-owned, supervisor-mode library named SUPER. TKB includes a four-word, mode-switching vector within the task image for each call to a subroutine within the library. The library is position independent and is mapped with supervisor I-space APR0. This is a requirement for CSM libraries because the CSM expects to find the entry point of the completion routine at location 10.

The memory allocation map for TSUP (Example 10–5) shows:

- $CMPAL is linked from the .STB file of the library and begins at location 0.
- The mode-switching vectors begin at 005136 and are 16. bytes. That means that TSUP calls subroutines within the library 2 times (4 words per vector).
- The initiation routine $SUPL is located at 4700.
- The SEARCH and SORT subroutines that were located at virtual 112 and 32, respectively, in the virtual address space of SUPER have been relocated to the mode-switching vectors residing at 5136 and 5146 respectively, in TSUP.
- The SAVAL module from SYSLIB containing $SAVAL has been linked into the task image instead of including $SAVAL from the library's .STB file.

#### Running TSUP

After building SUPER and TSUP as indicated in the task-build command sequence discussed previously, you install SUPER and run TSUP. TSUP prompts you for a number:

```
ARRAY (x)

x   The position in which to store the number in the array.
```

You enter a number. TSUP stores the number in the array and prompts you again for a number. This continues until you either have entered a 0, an illegal number, or 10 numbers. Then TSUP calls the SORT routine in SUPER.

You enter a number. TSUP calls the SEARCH routine in SUPER. Then TSUP outputs a message indicating whether the number was in the array.

## 10.7.4 The CSM Library Dispatching Process

When you build the referencing task, if you specify the SV argument to the RESSUP or SUPLIB option, then TKB includes a four-word context-switching vector for each call to a subroutine in the library. This is very generally discussed in Section 9.2. This section discusses the CSM library vector in detail.

CSM mode switching occurs as follows:

1 The vector is entered with the return address on top of the stack (TOS).

2 The vector pushes the completion routine address on the stack.

3 A CSM instruction is executed with the supervisor-mode entry point as the immediate addressing mode parameter. The CSM instruction:

   a. Evaluates the source parameter and stores the entry point address in a temporary register

   b. Copies the user stack pointer to the supervisor stack pointer

   c. Places the current PS and PC on the supervisor stack clearing the condition codes in the PS

d. Pushes the entry point address on the supervisor stack

e. Places the contents of location 10 in supervisor I-space into the PC

The stack looks like this when the processor begins to execute at the contents of virtual 10 in supervisor mode:

```
user sp ----> return address
   completion routine address
      PS
      PC

super sp ----> entry point address
```

The most important aspect of how the CSM library mode-switching vector works is that the processor begins executing at the contents of virtual 10 in supervisor mode. This is why the completion routine must be located at virtual 0, so that virtual location 10 is within the completion routine.

## 10.8 Using Supervisor-Mode Libraries as Resident Libraries

Supervisor-mode libraries can double as conventional resident libraries. For position-independent, supervisor-mode libraries, you rebuild the referencing task using the RESLIB option instead of the RESSUP option. Indicate the first available user-mode APR that you want to map the library. For CSM libraries this always changes, because you cannot map a shared region with APR 0. You do not have to rebuild the library.

For absolute supervisor-mode libraries, rebuild the referencing task using the RESLIB option instead of the RESSUP option. Rebuild the library only if the beginning partition address in the PAR option is incompatible with the address limits of your referencing task.

## 10.9 Multiple Supervisor-Mode Libraries

A user task can reference multiple supervisor-mode CSM libraries. However, all the CSM libraries must use the completion routine that begins at virtual zero in supervisor-mode instruction space.

## 10.10 Linking a Resident Library to a Supervisor-Mode Library

You can link a conventional resident library to a supervisor-mode library using the following TKB or LINK command sequence:

```
TKB> F4PRES/-HD,F4PRES,LB:[[1,1]]F4PRES=
TKB> F4PRES/LB
TKB> /
Enter Options:
TKB> STACK=0
TKB> SUPLIB=FCSFSL:SV
TKB> PAR=F4PRES:140000:20000
TKB> //
>
> LINK/TAS:F4PRES/NOH/MAP:F4PRES/SYM:LB:[1,1]F4PRES/OPT-
->  F4PRES/LIB
Option?  STACK=0
Option?  SUPLIB=FCSFSL:SV
Option?  PAR=F4PRES:140000:20000
Option?  RET
>
```

These two command sequences show you how to link F4PRES to FCSFSL.

## 10.11 Linking Supervisor-Mode Libraries

You cannot link supervisor-mode libraries together, and you cannot link a supervisor-mode library to a resident user-mode library. Calling a user-mode library is not possible because its code is not mapped through the I-space APRs while in the supervisor-mode library. However, you can link user-mode libraries to a supervisor-mode library.

## 10.12 Writing Your Own Vectors and Completion Routines

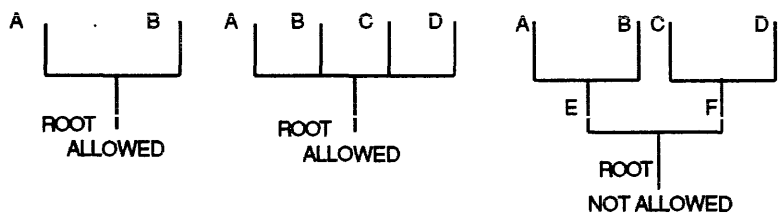
You can write your own mode-switching vectors and completion routines. This might be necessary for threaded code. If you use your own vectors, build them into the task and use the -SV switch on the RESSUP or RESLIB option when you build the referencing task. If you create your own completion routines, write your completion routine to resemble the system-supplied completion routines (see Example 10–3) as much as possible. If you do not retain the last three lines of code as indicated, then if the Executive processes an interrupt before the mode switch back to user mode has completed, your task might crash.

## 10.13 Overlaid Supervisor-Mode Libraries

It is possible to use overlaid supervisor-mode libraries. Three restrictions must be noted when building these libraries:

- The completion routine for the library must be in the root.
- Only one level of overlay is allowed. This is illustrated in Example 10–4.
- Although the Fast Task Builder (FTB) can link to supervisor-mode libraries, it cannot link to overlaid supervisor-mode libraries.

**Figure 10–2 Overlay Configuration Allowed for Supervisor-Mode Libraries**



# A ERROR MESSAGES

The Task Builder produces diagnostic and fatal error messages. Error messages are printed in the following forms:

```
TKB -- *DIAG*-error-message
```

or

```
TKB -- *FATAL*-error-message
```

After a fatal error, task builder aborts the current operation and returns to PDS (or MCR) command level. Diagnostic errors do not have this effect.

Some errors can be corrected from the terminal. If you are entering text at the terminal, and a diagnostic error message is printed, the error can be corrected, and the task building sequence continued. If the same error is detected by the Task Builder in an indirect file or in Batch, the Task Builder cannot request correction and thus the error is termed fatal and the task build is aborted.

Some diagnostic error messages are simply informative and advise you of an unusual condition. If you consider the condition normal to your task, you can run the task image.

Each error has in addition a status of Warning, Error, or Severe error; which is indicated after the code Severity below. When the task builder exits, the status of the worst error or a status of success is returned. In an indirect command file the status can be tested. For example, you might decide not to continue with a command sequence which runs the task if the task build was unsuccessful. See the IAS PDS User's Guide or the IAS MCR User's Guide.

This appendix tabulates the error messages produced by the Task Builder. The numbers below are internal Task Builder error numbers and are given for information purposes only. Most of the error messages are self-explanatory. The Task Builder prints the text shown in this manual in upper case letters. In some cases, the Task Builder prints the line in which the error occurred, so that the user can examine the line which caused the problem and correct it.

0, ILLEGAL GET COMMAND LINE ERROR CODE

**Severity:** Severe

**Explanation:** (System error. (No recovery.))

1, COMMAND SYNTAX ERROR
invalid-line

**Severity:** Severe

**Explanation:** (The invalid-line printed has incorrect syntax.)

2, REQUIRED INPUT FILE MISSING

**Severity:** Severe

**Explanation:** (At least one input file is required for a task build.)

3, ILLEGAL SWITCH
invalid-line

**Severity:** Severe

**Explanation:** The invalid line printed contains an illegal switch or switch value.

4, NO DYNAMIC STORAGE AVAILABLE

**Severity:** Severe

**Explanation:** The Task Builder needs additional storage and cannot obtain it. The input has exceeded the Task Builder's capability. See Appendix F, Section F.1.1.

5, ILLEGAL ERROR/SEVERITY CODE

**Severity:** Severe

**Explanation:** System error. (No recovery.)

6, COMMAND I/O ERROR

**Severity:** Severe

**Explanation:** I/O error on command input device. (Device may not be online or possible hardware error.)

7, INDIRECT FILE OPEN FAILURE
invalid-line

**Severity:** Severe

**Explanation:** The invalid-line contains a reference to a command input file which could not be located.

8, INDIRECT COMMAND SYNTAX ERROR
invalid-line

**Severity:** Severe

**Explanation:** The invalid-line printed contains a syntactically incorrect indirect file specification.

9, INDIRECT FILE DEPTH EXCEEDED
invalid-line

**Severity:** Severe

**Explanation:** The invalid-line printed gives the file reference that exceeded the permissible indirect file depth (3).

10, I/O ERROR ON INPUT FILE file-name

**Severity:** Severe

11, OPEN FAILURE ON FILE file-name

**Severity:** Severe

12, SEARCH STACK OVERFLOW ON SEGMENT segment-name

**Severity:** Severe

**Explanation:** The segment segment-name is more than 16 branch segments from the root segment.

13, PASS CONTROL OVERFLOW AT SEGMENT segment-name

**Severity:** Severe

**Explanation:** The segment segment-name is more than 16 branch segments from the root segment.

14, FILE file-name HAS ILLEGAL FORMAT

**Severity:** Severe

**Explanation:** The file file-name contains an object module whose format is not valid.

15, MODULE module-name AMBIGUOUSLY DEFINES P-SECTION p-sect-name

**Severity:** Warning

**Explanation:** The p-section p-sect-name has been defined in two modules not on a common path and referenced ambiguously.

16, MODULE module-name MULTIPLY DEFINES P-SECTION p-sect-name

**Severity:** Warning

**Explanation:**

1 The p-section p-sect-name has been defined in the same segment with different attributes.

or

2 A global p-section has been defined in more than one segment along a common path with different attributes.

17, MODULE module-name MULTIPLY DEFINES XFR ADDR IN SEG segment-name

**Severity:** Warning

**Explanation:** This error occurs when more than one module comprising the root has a start address.

18, MODULE module-name ILLEGALLY DEFINES XFR ADDRESS p-sect-name addr

**Severity:** Warning

**Explanation:** The module module-name is in an overlay segment and has a start address. The start address must be in the root segment of the main tree.

19, P-SECTION p-sect-name HAS OVERFLOWED

**Severity:** Error

**Explanation:** A section greater than 32K has been created.

20, MODULE module-name AMBIGUOUSLY DEFINES SYMBOL sym-name

**Severity:** Warning

**Explanation:** Module module-name references or defines a symbol sym-name whose definition cannot be uniquely resolved.

21, MODULE module-name MULTIPLY DEFINES SYMBOL sym-name

**Severity:** Warning

**Explanation:** Two definitions for the relocatable symbol sym-name have occurred on a common path. Or two definitions for an absolute symbol with the same name but different values have occurred.

22, INSUFFICIENT APRS AVAILABLE TO MAP READ ONLY ROOT

**Severity:** Severe

**Explanation:** No virtual address space can be found to map the read-only portion of a task.

23, SEGMENT seg-name HAS ADDR OVERFLOW: ALLOCATION DELETED

**Severity:** Severe

**Explanation:** Within a segment, the program has attempted to allocate more than 32K. A map file will be produced if one was requested.

24, ALLOCATION FAILURE ON FILE file-name

**Severity:** Severe

**Explanation:**

1 The Task Builder could not acquire sufficient disk space to store the task image file.

If possible, delete unnecessary files on disk to make more room available.

or

2 An attempt has been made to write the task file into a directory for which the user does not have write access.

25, I/O ERROR ON OUTPUT FILE file-name

**Severity:** Severe

**Explanation:** This error may occur on any of the three output files.

26, LOAD ADDR OUT OF RANGE IN MODULE module-name

**Severity:** Error

**Explanation:** An attempt has been made to store data in the task image outside the address limits of the segment. This usually indicates incorrect use of an absolute p-section

27, TRUNCATION ERROR IN MODULE module-name

**Severity:** Warning

**Explanation:** An attempt has been made to load a global value greater than +127 or less than -128 into a byte. The low-order eight bits are loaded.

28, number UNDEFINED SYMBOLS SEGMENT seg-name

**Severity:** Error

**Explanation:** The Memory Allocation File lists each undefined symbol by segment.

29, INVALID KEYWORD IDENTIFIER
invalid-line

**Severity:** Severe

**Explanation:** The invalid-line printed contains an unrecognizable option keyword.

30, OPTION SYNTAX ERROR
invalid-line

**Severity:** Severe

**Explanation:** The invalid-line printed contains unrecognizable syntax.

31, TOO MANY PARAMETERS
invalid-line

**Severity:** Severe

**Explanation:** The invalid-line printed contains a keyword with more parameters than required.

32, ILLEGAL MULTIPLE PARAMETER SETS
invalid-line

**Severity:** Severe

**Explanation:** The invalid-line printed contains multiple parameters for an option keyword which only allows a single parameter.

33, INSUFFICIENT PARAMETERS
invalid-line

**Severity:** Severe

**Explanation:** The invalid-line contains a keyword with an insufficient number of parameters to complete the keyword meaning.

34, TASK HAS ILLEGAL MEMORY LIMITS

**Severity:** Severe

**Explanation:** The highest virtual address of the task is greater than 32K words. Relink the task without a task image file to trace the cause.

35, OVERLAY DIRECTIVE HAS NO OPERANDS
invalid-line

**Severity:** Severe

**Explanation:** All overlay directives except .END require operands.

36, ILLEGAL OVERLAY DIRECTIVE
invalid-line

**Severity:** Severe

**Explanation:** The invalid-line printed contains an unrecognizable overlay directive.

37, OVERLAY DIRECTIVE SYNTAX ERROR
invalid-line

**Severity:** Severe

**Explanation:** The invalid-line printed contains a syntax error.

38, ROOT SEGMENT IS MULTIPLY DEFINED
invalid-line

**Severity:** Severe

**Explanation:** The invalid-line printed contains the second .ROOT directive encountered. Only one .ROOT directive is allowed.

39, LABEL OR NAME IS MULTIPLY DEFINED
invalid-line

**Severity:** Severe

**Explanation:** The invalid-line printed contains a name that has already appeared on a .FCTR, .NAME, or .PSECT directive.

40, O ROOT SEGMENT SPECIFIED

**Severity:** Severe

**Explanation:** The overlay description did not contain a .ROOT directive.

41, BLANK P-SECTION NAME IS ILLEGAL
invalid-line

**Severity:** Severe

**Explanation:** The invalid-line printed contains a .PSECT directive that does not have a p-section name.

42, ILLEGAL P-SECTION/SEGMENT ATTRIBUTE
invalid-line

**Severity:** Severe

**Explanation:** The invalid-line printed contains a p-section or segment attribute that is not recognized.

43, ILLEGAL OVERLAY DESCRIPTION OPERATOR
invalid-line

**Severity:** Severe

**Explanation:** The invalid-line printed contains an unrecognizable operator in an overlay description.

44, TOO MANY NESTED .ROOT/.FCTR DIRECTIVES
invalid-line

**Severity:** Severe

**Explanation:** The invalid-line printed contains a .FCTR directive that exceeds the maximum nesting level (32).

45, TOO MANY PARENTHESES LEVELS
invalid-line

**Severity:** Severe

**Explanation:** The invalid-line printed contains a parenthesis that exceeds the maximum nesting level (32).

46, UNBALANCED PARENTHESES
invalid-line

**Severity:** Severe

**Explanation:** The invalid-line printed contains unbalanced parentheses.

47, ILLEGAL BASE OR TOP ADDRESS OFFSET

**Severity:** Severe

**Explanation:** The task is too large to fit into the space allowed by BASE= or TOP= keywords.

48, ILLEGAL LOGICAL UNIT NUMBER
invalid-line

**Severity:** Severe

**Explanation:** The invalid-line printed contains a device assignment to a unit number larger than the number of logical units specified by the UNITS keyword or assumed by default if the UNITS keyword is not used.

49, ILLEGAL NUMBER OF LOGICAL UNITS
invalid-line

**Severity:** Severe

**Explanation:** The invalid-line printed contains a logical unit number greater than 250.

50, ILLEGAL MAXIMUM EXTENSION
invalid line

**Severity:** Severe

**Explanation:** The argument to the MAXEXT option is outside the range 0-2000 (octal).

51, ILLEGAL BASE OR TOP BOUNDARY VALUE
invalid-line

**Severity:** Severe

52, ILLEGAL POOL USAGE NUMBER SPECIFIED
invalid-line

**Severity:** Severe

**Explanation:** The pool request is greater than 255 or it is zero.

53, ILLEGAL DEFAULT PRIORITY SPECIFIED
invalid-line

**Severity:** Severe

**Explanation:** The invalid-line printed contains a priority greater than 250.

54, ILLEGAL ODT OR TASK VECTOR SIZE

**Severity:** Severe

**Explanation:** SST vector size specified greater than 32 words.

55, ILLEGAL FILENAME
invalid-line

**Severity:** Severe

**Explanation:** The invalid-line printed contains a wild card (*) in a file specification. The use of wild cards is prohibited.

56, ILLEGAL DEVICE/VOLUME
invalid-line

**Severity:** Severe

**Explanation:** The device/volume string is too long.

57, LOOKUP FAILURE ON FILE file-name
invalid-line

**Severity:** Severe

**Explanation:** The invalid-line printed contains a file name which cannot be located in the directory.

58, ILLEGAL DIRECTORY
invalid-line

**Severity:** Severe

**Explanation:** The invalid-line printed contains an illegal UFD.

59, INCOMPATIBLE REFERENCE TO SGA P-SECTION p-sect-name

**Severity:** Error

**Explanation:** A task has attempted to reference more storage in a shareable global area than exists in the shareable global area definition.

60, ILLEGAL REFERENCE TO SGA P-SECTION p-sect-name

**Severity:** Error

**Explanation:** A task has attempted to reference a p-sect-name existing in a shareable global area but has not named the SGA in the SGA option.

61, SGA MEMORY ALLOCATION CONFLICT
keyword-string

**Severity:** Severe

**Explanation:** One of the following problems has occurred:

1 More than seven shareable global areas have been specified.

2 The same shareable global area has been specified more than once.

3 Shareable global areas whose memory allocations overlap have been specified.

4 BASE or TOP specifications conflict.

62, LOOKUP FAILURE SGA FILE
invalid-line

**Severity:** Severe

**Explanation:** No symbol table or task image file found for the shareable global area on SY0 under UFD [1,1].

63, Not used.

64, ILLEGAL PARTITION/SGA SPECIFIED
invalid-line

**Severity:** Severe

**Explanation:** User defined base or length not on 32 word bound or user defined length = 0.

65, NO MEMORY AVAILABLE FOR SGA library-name

**Severity:** Severe

**Explanation:** Insufficient virtual memory available to cover total memory needed by referenced shareable global areas.

66, PIC SGAS MAY NOT REFERENCE OTHER SGAS
invalid-line

**Severity:** Severe

67, ILLEGAL APR RESERVATION

**Severity:** Severe

**Explanation:** APR specified in SGA option that is outside the range 0-7.

68, I/O ERROR SGA IMAGE FILE

**Severity:** Severe

**Explanation:** An I/O error has occurred during an attempt to open or read the Task Image File of a shareable global area.

69, Not used.

70, Not used.

71, INVALID APR RESERVATION

**Severity:** Severe

**Explanation:** APR specified in SGA option for an absolute shareable global area.

72, COMPLEX RELOCATION ERROR - DIVIDE BY ZERO: MODULE
module-name

**Severity:** Warning

**Explanation:** A divisor having the value zero was detected in a complex expression. The result of the divide was set to zero. (Probable cause - division by an undefined global symbol.)

73, WORK FILE I/O ERROR

**Severity:** Severe

**Explanation:** I/O error during an attempt to reference data stored by the Task Builder in a work file. Possibly an attempt to extend the file when no more space is available on the volume. See Appendix F, Section F.1.1.

74, LOOKUP FAILURE ON SYSTEM LIBRARY FILE

**Severity:** Error

**Explanation:** The Task Builder cannot find the System Library (usually LB0:[1,1]SYSLIB.OLB) file to resolve undefined symbols.

75, UNABLE TO OPEN WORK FILE

**Severity:** Severe

**Explanation:** Work file device is not mounted or has not been initialized as Files-11, or there is no space on the volume. See Appendix F, Section F.1.1.

76, NO VIRTUAL MEMORY STORAGE AVAILABLE

**Severity:** Severe

**Explanation:** Maximum permissible size of the work file exceeded (no recovery). See Appendix F, Section F.1.1 and Section F.3.

77, MODULE module-name NOT IN LIBRARY

**Severity:** Severe

**Explanation:** The Task Builder could not find the module in the library.

78, INCORRECT LIBRARY MODULE SPECIFICATION
invalid-line

**Severity:** Severe

**Explanation:** The invalid-line contains a module name with a non-Radix-50 character.

79, LIBRARY FILE filename HAS INCORRECT FORMAT

**Severity:** Severe

**Explanation:** A module has been requested from a library file that has an empty module name table.

80, SGA IMAGE HAS INCORRECT FORMAT
invalid-line

**Severity:** Severe

**Explanation:** The invalid-line specifies a shareable global area that has one of the following problems:

1 The SGA task image file has a header.

2 The shareable global area references another shareable global area with invalid address bounds (that is, not on 4K boundary).

3 The shareable global area has invalid address bounds.

81, PARTITION partition-name HAS ILLEGAL MEMORY LIMITS

**Severity:** Severe

**Explanation:** The user has attempted to build a privileged task whose length exceeds 16K.

82, Not used.

83, ABORTED VIA REQUEST
input-line

**Severity:** Severe

**Explanation:** The input-line contains a request from the user to abort the task build.

84-87, Not used.

88, SGA REFERENCES OVERLAID SGA

**Severity:** Severe

**Explanation:** It is illegal to build an SGA which references another overlaid SGA.

89, TASK IMAGE FILE file-name IS NON-CONTIGUOUS

**Severity:** Error

**Explanation:** Not enough contiguous disk space could be found to create the task image file. The task image is placed in a non-contiguous file, which must be copied with the COPY/CONTIGUOUS PDS command (or using the PIP utility under MCR) before it can be installed or run.

90, VIRTUAL SECTION HAS ILLEGAL ADDRESS LIMITS
option-line

**Severity:** Severe

**Explanation:** The option-line printed contains a VSECT keyword whose base address plus window size exceeds 177777.

91, FILE file-name ATTEMPTED TO STORE DATA IN VIRTUAL SECTION

**Severity:** Error

**Explanation:** The file contains a module that has attempted to initialize a virtual section with data.

92, SGA MAPPED ARRAY ALLOCATION TOO LARGE
invalid-line

**Severity:** Severe

**Explanation:** The invalid-line printed contains a reference to an SGA that has allocated too much memory in the task's mapped array area. The total allocation exceeds 2.2 million bytes.

93, INVALID REFERENCE TO MAPPED ARRAY BY MODULE module-name

**Severity:** Error

**Explanation:** The module has attempted to initialize the mapped array with data. An SPR should be submitted if this problem is caused by DIGITAL-supplied software.

94, END OF FILE REACHED BEFORE .END DIRECTIVE IN file-name

**Severity:** Severe

**Explanation:** The overlay description file named in this message does not contain a .END directive as required.

95, DUPLICATE SGA NAME
invalid-line

**Severity:** Severe

**Explanation:** The shareable global area name specified has already appeared.

96, SYMBOL sym-name NOT FOUND FOR PATCH

**Severity:** Warning

**Explanation:** A global symbol specified in a GBLPAT or SYMPAT option cannot be found.

97, SEGMENT seg-name NOT FOUND FOR PATCH

**Severity:** Warning

**Explanation:** The segment name specified in an ABSPAT, GBLPAT or SYMPAT option cannot be found.

98, ILLEGAL NUMBER OF REGIONS

**Severity:** Severe

**Explanation:** The argument to the ATRG option is greater than 240.

99, INSUFFICIENT APRS TO MAP TASK

**Severity:** Severe

**Explanation:** There is not enough virtual address space, after allocating libraries common areas, the task pure area and resident overlays, to map the task root.

100, Supervisor-mode library reference error

101, Illegal system size specified

102, Conflicting base addresses in cluster library

**Explanation:** This conflict arises when you specify APRs, for both PIC and non-PIC libraries that are included in the cluster. See the APR parameter as described in te CLSTR option. This is a fatal error.

103, Library (library-name) not found in any cluster

**Explanation:** All task image and symbol table files to be included as cluster elements must reside in lb:[1,1].

104, Illegal cluster configuration

**Explanation:** If the cluster contains a non-overlaid library, that library must be the first library in the cluster. check the configuration of the libraries on the cluster. this is a fatal error.

105, Cluster library element does not have null root

**Explanation:** This is a fatal error. all libraries, except the first, must be plas-overlaid and have a null root. the first library in the group can be non-overlaid or overlaid with a null or non-null root.

107, Supervisor mode completion routine is undefined

**Explanation:** The Task Builder could not locate the symbol x , which was specified in the CMPRT=X option.

108, Library not built as a supervisor-mode library

**Explanation:** The library referred to in a ressup or suplib option was built without a completion (CMPRT=X) routine and is not a supervisor-mode library

# B TASK BUILDER DATA FORMATS

This appendix is of interest mainly to readers who need to understand the object module format.

An object module is the fundamental unit of input to the Task Builder.

Object modules are created by any of the standard language processors (for example MACRO-11, FORTRAN) or the Task Builder itself (symbol definition file). The IAS Librarian provides the ability to combine a number of object modules together into a single library file (see the IAS PDS User's Guide or IAS MCR User's Guide for a specification of the LIBRARIAN command; for a more detailed description see the *RSX-11M/M-PLUS Utilities Manual*.

An object module consists of variable length records of information that describe the contents of the module. Six record (or block) types are included in the object language. These records guide the Task Builder in the translation of the object language into a task image.

The six record types are:

Type 1 - Declare Global Symbol Directory (GSD)
Type 2 - End of Global Symbol Directory
Type 3 - Text Information (TXT)
Type 4 - Relocation Directory (RLD)
Type 5 - Internal Symbol Directory (ISD)
Type 6 - End of Module

Each object module must consist of at least five of the record types. The one record type that is not mandatory is the internal symbol directory. The appearance of the various record types in an object module follows a defined format. See Section B.1.

An object module must begin with a Declare GSD record and end with an end-of-module record. Additional Declare GSD records may occur anywhere in the file but before an end-of-GSD record. An end-of-GSD record must appear before the end-of-module record. At least one relocation directory record must appear before the first text information record. Additional relocation directory and text information records may appear anywhere in the file. The internal symbol directory records may appear anywhere in the file between the initial declare GSD and end-of-module records.

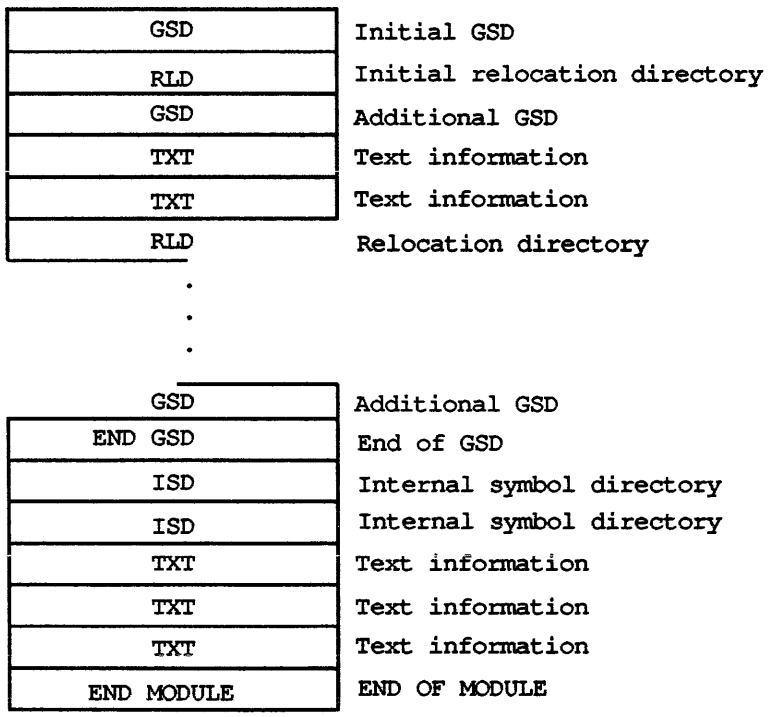
Object module records are variable length and are identified by a record type code in the first word of the record. The format of additional information in the record is dependent upon the record type.

## B.1 Global Symbol Directory (GSD)

Global symbol directory records contain all the information necessary to assign addresses to global symbols and to allocate the memory required by a task.

GSD records are the only records processed by the Task Builder in the first pass, thus significant time can be saved if all GSD records are placed at the beginning of a module (that is, less of the file must be read in phase 3).

**Figure B–1 General Object Module Format**




GSD records contain seven types of entries:

Type 0 - Module Name
Type 1 - Control Section Name
Type 2 - Internal Symbol Name
Type 3 - Transfer Address
Type 4 - Global Symbol Name
Type 5 - Program Section Name
Type 6 - Program Version Identification
Type 7 - Mapped Array Declaration

Each entry type is represented by four words in the GSD record. The first two words contain six Radix-50 characters. The third word contains a flag byte and the entry type identification. The fourth word contains additional information about the entry. See Figure B–2.

**Figure B–2 GSD Record and Entry Format**

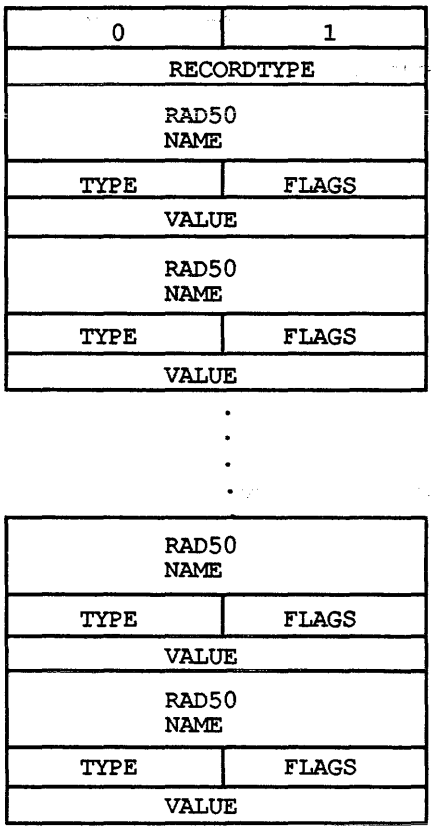
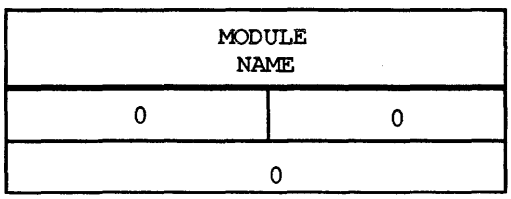



**Figure B–3 Module Name Entry Format**




## B.1.1 Module Name

The module name entry declares the name of the object module. The name need not be unique with respect to other object modules (i.e., modules are identified by file not module name) but only one such declaration may occur in any given object module. See Figure B–3.

## B.1.2 Control Section Name

Control sections, which include ASECTs, blank-CSECTs, and named-CSECTs are supplanted in IAS by PSECTs. For compatibility, the Task Builder processes ASECTs and both forms of CSECTs. Section B.2 details the entry generated for a PSECT statement. In terms of a PSECT statement we can define ASECT and CSECT statements as follows:

For a blank CSECT, a PSECT is defined with the following attributes:

```
.PSECT ,LCL,REL,CON,RW,I,LOW
```

For a named CSECT, the PSECT definition is:

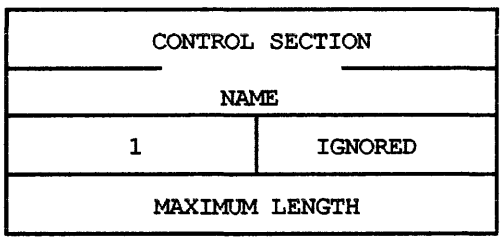
```
.PSECT name,GBL,REL,OVR,RW,I,LOW
```

For an ASECT, the PSECT definition is:

```
.PSECT . ABS.,GBL,ABS,I,OVR,RW,LOW
```

ASECTs and CSECTs are processed by the Task Builder as PSECTs with the fixed attributes defined above. The entry generated for a control section is shown in Figure B–4.

**Figure B–4 Control Section Name Entry Format**




## B.1.3 Internal Symbol Name

The internal symbol name entry declares the name of an internal symbol (with respect to the module). The Task Builder does not support internal symbol tables and therefore the detailed format of this entry is not defined (see Figure B–5. If an internal symbol entry is encountered while reading the GSD, it is merely ignored.

Figure B–5 Internal Symbol Name Entry Format

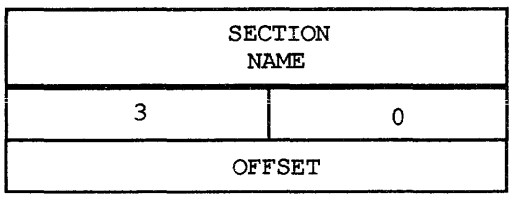



Figure B–6 Transfer Address Entry Format

Place figure here from page B-6.

## B.1.4 Transfer Address

The transfer address entry declares the transfer address of a module relative to a P-section. The first two words of the entry define the name of the P-section and the fourth word the relative offset from the beginning of that P-section. If no transfer address is declared in a module, a transfer address entry either must not be included in the GSD or a transfer address of 000001 relative to the default absolute P-section (. ABS.) must be specified. See Figure B–6.

**Note: If the P-section is absolute, then OFFSET is the actual transfer address if not 000001.**

## B.1.5 Global Symbol Name

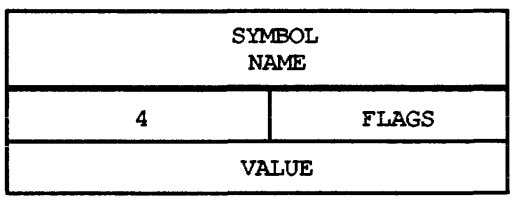
The global symbol name entry (see Figure B–7) declares either a global reference or a definition. All definition entries must appear after the declaration of the P-section under which they are defined and before the declaration of another P-section. Global references may appear anywhere within the GSD.

The first two words of the entry define the name of the global symbol. The flag byte declares the attributes of the symbol and the fourth word the value of the symbol relative to the P-section under which it is defined.

The flag byte of the symbol declaration entry has the following bit assignments:

| | | |
|---|---|---|
| Bit | 0 - | Weak Qualifier |
| | 0 = | Symbol is a strong definition or reference and is resolved in the normal manner. |
| | 1 = | Symbol is a weak definition or reference. A weak reference (Bit 3=0) is ignored. A weak definition (Bit 3=1) is ignored unless a previous reference has been made. |
| Bit | 1 - | Not used. |
| Bit | 2 | |
| | 0 = | Normal Definition or reference. |
| | 1 = | Library definition. If the symbol is defined in a resident library STB file, the base address of the library is added to the value, and the symbol is converted to absolute (bit 5 is reset); otherwise, the bit is ignored. |
| Bit | 3 | |
| | 0 = | Global symbol reference. |
| | 1 = | Global symbol definition. |
| Bit | 4 - | Not used. |
| Bit | 5 - | Relocation. |
| | 0 = | Absolute symbol value. |
| | 1 = | Relative symbol value. |
| Bits | 6 to 7 - | Not used. |

**Figure B–7 Global Symbol Entry Format**




## B.2 Program Section Name

The P-section name entry (see Figure B–8) declares the name of a P-section and its maximum length in the module. It also declares the attributes of the P-section via the flag byte.

GSD records must be constructed such that once a P-section name has been declared all global symbol definitions that pertain to that P-section must appear before another P-section name is declared. Global symbols are declared via symbol declaration entries. Thus the normal format is a P-section name followed by zero or more symbol declarations, the next P-section name followed by zero or more symbol declarations, and so on.

The flag byte of the P-section entry has the following bit assignments:

Bit 0 - Memory Speed

0 = P-section is to occupy low speed (core) memory.

1 = P-section is to occupy high speed (that is, MOS/Bipolar) memory.

Bit 1 - Library P-section

0 = Normal P-section

1 = Relocatable P-section that references a shareable global area.

Bit 2 - Allocation

0 = P-section references are to be concatenated with other references to the same P-section to form the total memory allocated to the section.

1 = P-section references are to be overlaid. The total memory allocated to the P-section is the largest request made by individual references to the same P-section.

Bit 3 - Not used but reserved.

Bit 4 - Access

0 = P-section has read/write access.

1 = P-section has read-only access.

Bit 5 - Relocation

0 = P-section is absolute and requires no relocation.

1 = P-section is relocatable and references to the control section must have a relocation bias added before they become absolute.

Bit 6 - Scope

0 = The scope of the P-section is local. References to the same P-section will be collected only within the segment in which the P-section is defined.

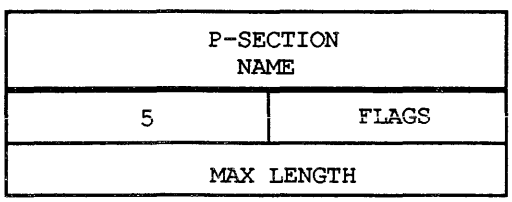
1 = The scope of the P-section is global. References to the P-section are collected across segment boundaries. The segment in which a global P-section is allocated storage is determined either by the first module that defines the P-section on a path or by direct placement of a P-section in a segment via the Overlay Description Language .PSECT directive.

Bit 7 - Type

0 = The P-section contains instruction (I) references.

1 = The P-section contains data (D) reference Identification

**Figure B–8 P-section Name Entry Format**



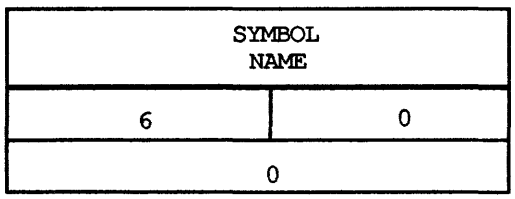

**Note: The length of all absolute sections is zero.**

## B.3 Program Version Identification

The program version identification entry (see Figure B–9) declares the version of the module. The Task Builder saves the version identification of the first module that defines a nonblank version. This identification is then included on the memory allocation map and is written in the label block of the task image file.

The first two words of the entry contain the version identification. The flag byte and fourth words are not used and contain no meaningful information.

**Figure B–9 Program Version Identification Entry Format**

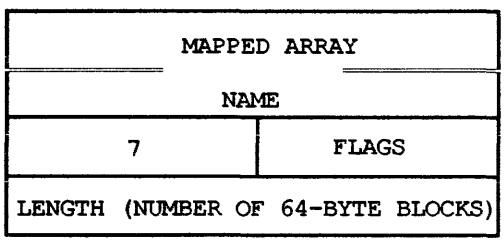



## B.4 Mapped Array Declaration

The Mapped Array Declaration (see Figure B–10) causes space to be allocated within the mapped array area of task memory. The array name is added to the list of task p-section names and may be referenced by subsequent RLD records. The length (in units of 64-byte blocks) is added to the task's mapped array allocation. The total memory allocated to each mapped array is rounded up to the nearest 512-byte boundary. The contents of the flags byte are reserved and assumed to be zero.

One additional address window is allocated whenever a mapped array is declared.

**Figure B–10 Mapped Array Declaration Format**



## B.5 End of Global Symbol Directory

The end-of-global-symbol-directory record (see Figure B–11) declares that no other GSD records are contained further on in the file. Exactly one end-of-GSD-record must appear in every object module and is one word in length.

Figure B–11 End of GSD Record Format


## B.6 Text Information

The text information record (see Figure B–12) contains a byte string of information that is to be written directly into the task image file. The record consists of a load address followed by the byte string.

Text records may contain words and/or bytes of information whose final contents are yet to be determined. This information will be bound by a relocation directory record that immediately follows the text record (see Section B.7). If the text record does not need modification, then no relocation directory record is needed. Thus multiple text records may appear in sequence before a relocation directory record.

The load address of the text record is specified as an offset from the current P-section base. At least one relocation directory record must precede the first text record. This directory must declare the current P-section.

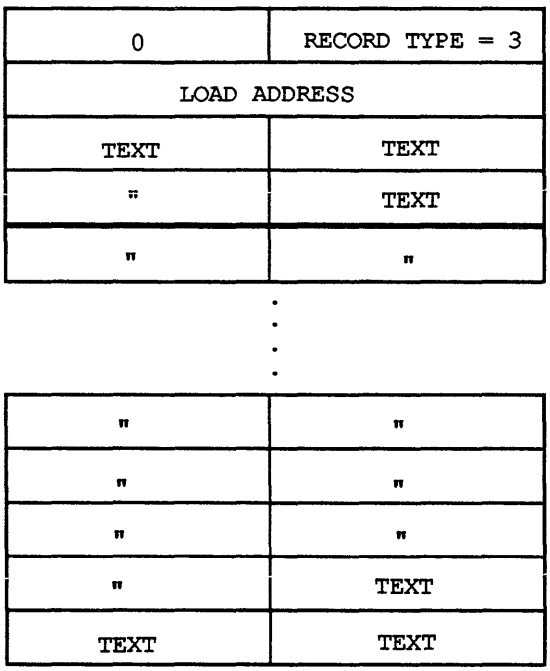
The Task Builder writes a text record directly into the task image file and computes the value of the load address minus four. This value is stored in anticipation of a subsequent relocation directory that modifies words and/or bytes that are contained in the text record. When added to a relocation directory displacement byte, this value yields the address of the word and/or byte to be modified in the task image.

## B.7 Relocation Directory

Relocation directory records (see Figure B–13) contain the information necessary to relocate and link a preceding text information record. Every module must have at least one relocation directory record that precedes the first text information record. The first record does not modify a preceding text record, but defines the current P-section and location. Relocation directory records contain 13 types of entries. These entries are classified as relocation or location modification entries. The following types of entries are defined:

Type 1 - Internal Relocation
Type 2 - Global Relocation
Type 3 - Internal Displaced Relocation

**Figure B–12 Text Information Record Format**




Type 4 - Global Displaced Relocation
Type 5 - Global Additive Relocation
Type 6 - Global Additive Displaced Relocation
Type 7 - Location Counter Definition
Type 10 - Location Counter Modification
Type 11 - Program Limits
Type 12 - P-Section Relocation
Type 13 - Not Used
Type 14 - P-Section Displaced Relocation
Type 15 - P-Section Additive Relocation
Type 16 - P-Section Additive Displaced Relocation
Type 17 - Complex Relocation
Type 20 - Library Relocation

Each type of entry is represented by a command byte (specifies type of entry and word/byte modification), a displacement byte, and the information required for the particular type of entry, in that order. The displacement byte, when added to the value calculated from the load address of the previous text information record, (see Section B.6) yields the virtual address in the image that is to be modified.

The command byte of each entry has the following bit assignments:

Bits 0 - 6 Specify the type of entry. Potentially 128 command types may be specified although only 15(decimal) are implemented.

Bit 7 - Modification

0 = The command modifies an entire word.

1 = The command modifies only one byte. The Task Builder checks for truncation errors in byte modification commands. If truncation is detected (that is, the modification value has a magnitude greater than 255), an error is produced.

**Figure B–13 Relocation Directory Record Format**

| 0 | RECORD TYPE = 4 |
|---|---|
| DISP | TYPE= CMD |
| INFO | INFO |
| " | INFO |
| " | " |
| " | " |
| " | " |
| " | " |

.
.
.
.

| CMD | " |
|---|---|
| INFO | DISP |
| " | INFO |
| " | " |
| " | " |
| " | " |
| " | " |
| DISP | CMD |
| INFO | INFO |
| INFO | INFO |
| INFO | INFO |

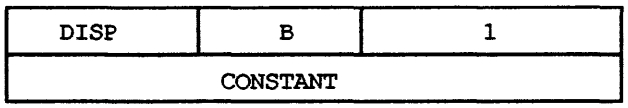
## B.8 Internal Relocation

This type of entry (see Figure B–14) relocates a direct pointer to an address within a module. The current P-section base address is added to a specified constant and the result is written into the task image file at the calculated address (that is, displacement byte added to value calculated from the load address of the previous text block).

Example:

```
A:      MOV     #A,R0
        or
        .WORD   A
```

**Figure B–14 Internal Relocation Command Format**



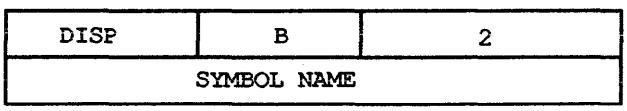

### B.8.1 Global Relocation

This type of entry (see Figure B–15) relocates a direct pointer to a global symbol. The definition of the global symbol is obtained and the result is written into the task image file at the calculated address.

Example:

```
MOV     #GLOBAL,R0
or
.WORD   GLOBAL
```

**Figure B–15 Global Relocation**



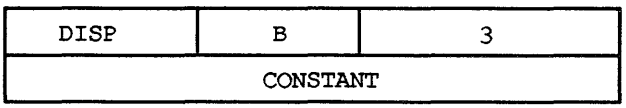
## B.8.2 Internal Displaced Relocation

This type of entry (see Figure B–16) relocates a relative reference to an absolute address from within a relocatable control section. The address plus 2 that the relocated value is to be written into is subtracted from the specified constant. The result is then written into the task image file at the calculated address.

Example:

```
CLR     177550
or
MOV     177550,R0
```

**Figure B–16 Internal Displaced Relocation**



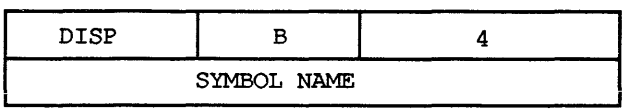

## B.8.3 Global Displaced Relocation

This type of entry (see Figure B–17) relocates a relative reference to global symbol. The definition of the global symbol is obtained and the address plus 2 that the relocated value is to be written into is subtracted from the definition value. This value is then written into the task image file at the calculated address.

Example:

```
CLR     GLOBAL
or
MOV     GLOBAL,R0
```

**Figure B–17 Global Displaced Relocation**



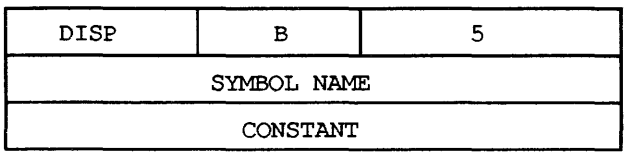
## B.8.4 Global Additive Relocation

This type of entry (see Figure B–18) relocates a direct pointer to a global symbol with an additive constant. The definition of the global symbol is obtained, the specified constant is added, and the resultant value is then written into the task image file at the calculated address.

Example:

```
MOV #GLOBAL+2,R0
or
.WORD GLOBAL-4
```

**Figure B–18 Global Additive Relocation**




## B.8.5 Global Additive Displaced Relocation

This type of entry (see Figure B–19) relocates a relative reference to a global symbol with an additive constant. The definition of the global symbol is obtained and the specified constant is added to the definition value. The address plus 2 that the relocated value is to be written into is subtracted from the resultant additive value. The resultant value is then written into the task image file at the calculated address.

Example:

```
CLR     GLOBAL+2
or
MOV     GLOBAL-5,R0
```

**Figure B–19 Global Additive Displaced Relocation**

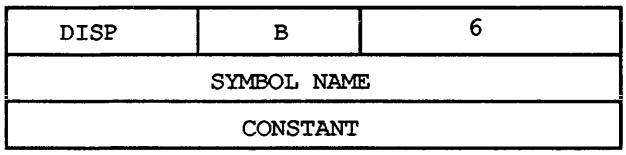



**Figure B–20 Location Counter Definition**

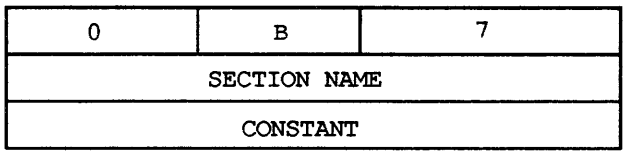
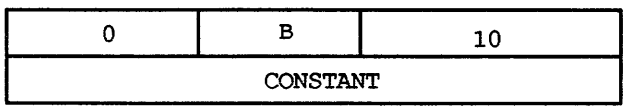



**Figure B–21 Location Counter Modification**




## B.8.6 Location Counter Definition

This type of entry (see Figure B–20) declares a current P-section and location counter value. The control base is stored as the current control section and the current control section base is added to the specified constant and stored as the current location counter value.

## B.8.7 Location Counter Modification

This type of entry (see Figure B–21) modifies the current location counter. The current P-section base is added to the specified constant and the result is stored as the current location counter.

Example:

```
.=.+N
or
.BLKB  N
```

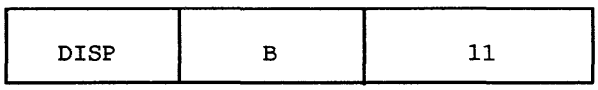
## B.9 Program Limits

This type of entry (see Figure B–22) is generated by the .LIMIT assembler directive. The first address above the header (normally the beginning of the stack) and highest address allocated to the tasks are obtained and written into the task image file at the calculated address and at the calculated address plus 2 respectively.

Example:

```
        .LIMIT
```

**Figure B–22 Program Limits**




### B.9.1 P-section Relocation

This type of entry (see Figure B–23) relocates a direct pointer to the start address of another P-section (other than the P-section in which the reference is made) within a module. The current base address of the specified P-section is obtained and written into the task image file at the calculated address.

Example:

```
        .PSECT  A
B:
        .
        .
        .
        .
        PSECT   C
        MOV     #B,R0

        or

        .WORD   B
```

**Figure B–23 P-section Relocation**

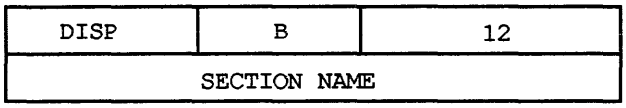
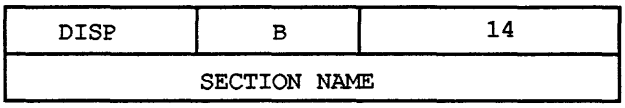



**Figure B–24 P-section Displaced Relocation**




## B.10 P-section Displaced Relocation

This type of entry (see Figure B–24) relocates a relative reference to the start address of another P-section within a module. The current base address of the specified P-section is obtained and the address plus 2 that the relocated value is to be written into is subtracted from the base value. This value is then written into the task image file at the calculated address.

Example:

```
        .PSECT   A
B:
        .
        .
        .
        .
        .PSECT   C
        MOV      B,R0
```

### B.10.1 P-section Additive Relocation

This type of entry (see Figure B–25) relocates a direct pointer to an address in another P-section within a module. The current base address of the specified p-section is obtained and added to the specified constant. The result is written into the task image file at the calculated address.

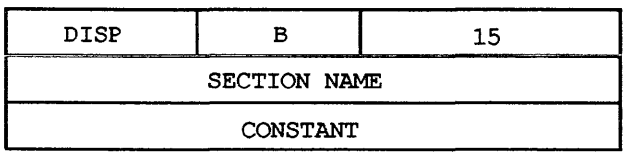
Example:

```
                .PSECT   A
B:
                .
                .
                .
                .
                .
C:              .
                .
                .
                .PSECT   D
                MOV      #B+10,R0
                MOV      #C,R0

                or

                .WORD    B+10
                .WORD    C
```

**Figure B–25 P-section Additive Relocation**




## B.10.2 P-section Additive Displaced Relocation

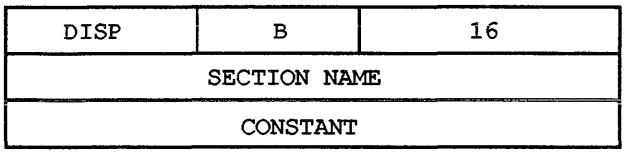
This type of entry (see Figure B–26) relocates a relative reference to an address in another P-section within a module. The current base address of the specified P-section is obtained and added to the specified constant. The address plus 2 that the relocated value is to be written into is subtracted from the resultant additive value. This value is then written into the task image file at the calculated address.

Example:

```
                .PSECT   A
B:
                .
                .
                .
C:              .
                .
                .
                .
                .PSECT   D

                MOV      B+10,R0
                MOV      C,R0
```

**Figure B–26 P-section Additive Displaced Relocation**




## B.10.3 Complex Relocation

This type of entry (see Figure B–27) resolves a complex relocation expression. Such an expression is one in which any of the MACRO-11 binary or unary operations are permitted with any type of argument, regardless of whether the argument is unresolved global, relocatable to any P-section base, absolute, or a complex relocatable subexpression.

The RLD command word is followed by a string of numerically-specified operation codes and arguments. All of the operation codes occupy one byte. The entire RLD command must fit in a single record. The following operation codes are defined.

0 - No operation

1 - Addition (+)

2 - Subtraction (-)

3 - Multiplication (*)

4 - Division (/)

5 - Logical AND (&)

6 - Logical inclusive OR (!)

10 - Negation (-)

11 - Complement (^C)

12 - Store result (command termination)

13 - Store result with displaced relocation (command termination)

16 - Fetch global symbol. It is followed by four bytes containing the symbol name in RADIX-50 representation.

17 - Fetch relocatable value. It is followed by one byte containing the sector number, and two bytes containing the offset within the sector.

20 - Fetch constant. It is followed by two bytes containing the constant.

21 - Fetch resident library base address. If the file is a resident library STB file, the library base address is obtained; otherwise, the base address of the Task Image is fetched.

The STORE commands indicate that the value is to be written into the task image file at the calculated address.

All operands are evaluated as 16-bit signed quantities using two's complement arithmetic. The results are equivalent to expressions that are evaluated internally by the assembler. The following rules are to be noted.

1 An attempt to divide by zero yields a zero result. The Task Builder issues a non-fatal diagnostic message.

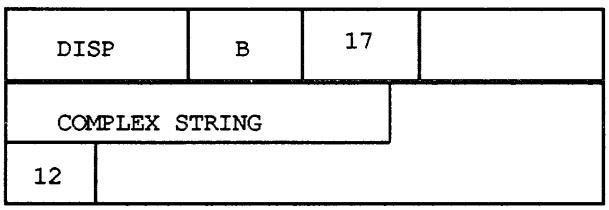
2 All results are truncated from the left in order to fit into 16 bits. No diagnostic is issued if the number was too large. If the result modifies a byte, the Task Builder checks for truncation errors.

3 All operations are performed on relocated (additive) or absolute 16-bit quantities. PC displacement is applied to the result only.

Example:

```
                .PSECT  ALPHA
A:
                .
                .
                .
                .PSECT  BETA
B:
                .
                .
                .
                MOV     #A+B-G1/G2&<^C<177120!G3>>,R1
```

**Figure B–27 Complex Relocation**




## B.10.4 Shareable Global Area Additive Relocation

This type of entry (see Figure B–28) relocates a direct pointer to address within a shareable global area (SGA).

If the current file is an SGA symbol table file (STB), the base address of the SGA is obtained and added to the specified constant. The result is written into the task image file at the calculated address. If the file is not associated with an SGA, the task base address is used.

Example:

Figure B–28 Resident Library Additive Relocation

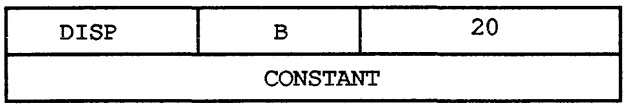



Figure B–29 Internal Symbol Directory Record Format

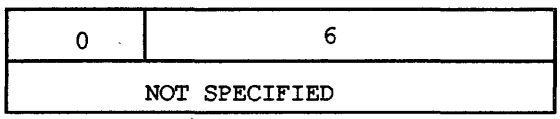
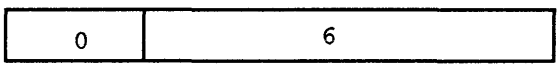



Figure B–30 End-Of-Module Record Format




## B.11 Internal Symbol Directory

Internal symbol directory records (see Figure B–29) declare definitions of symbols that are local to a module. This feature is not supported by the Task Builder and therefore a detailed record format is not specified. This type of record, if encountered, will be ignored by the Task Builder.

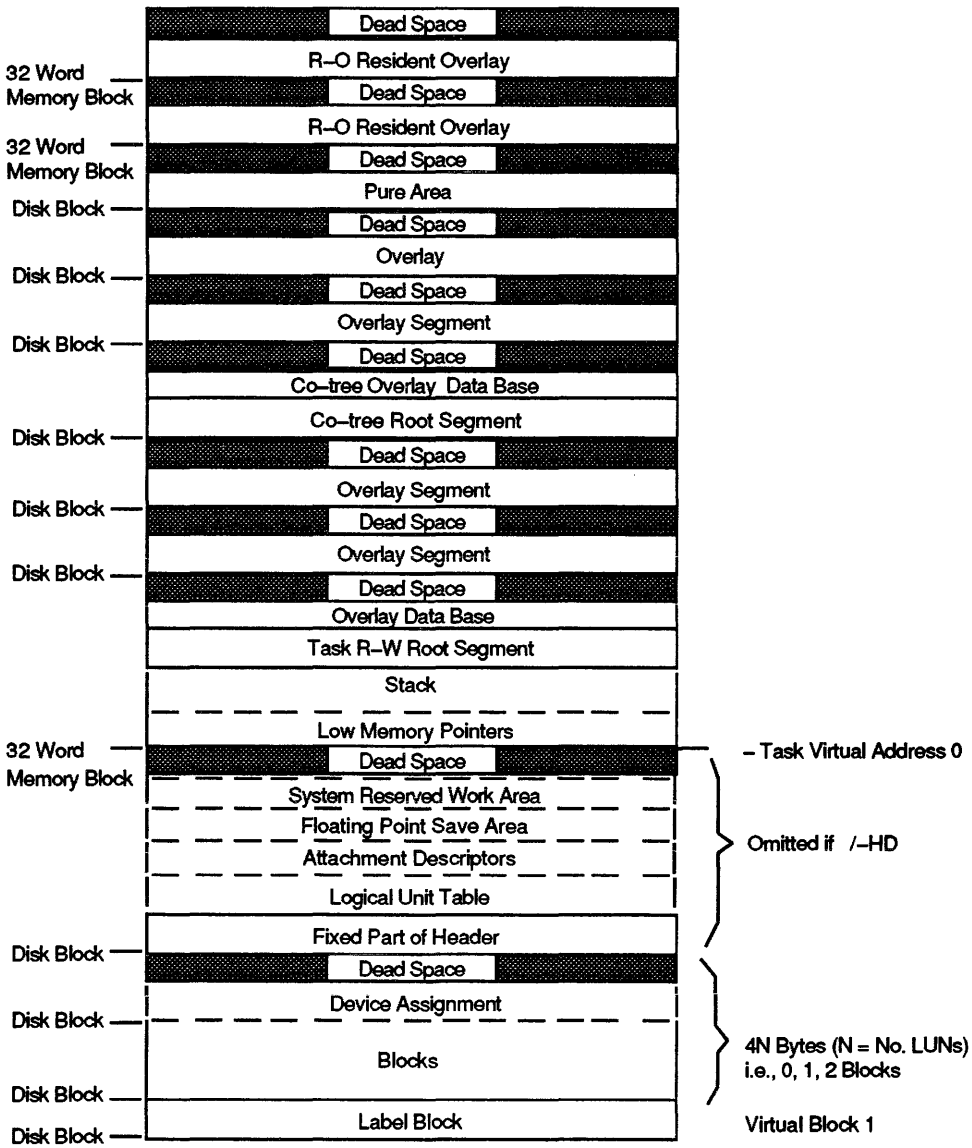
## B.12 End of Module

The end-of-module record (see Figure B–30) declares the end-of-an object module. Exactly one end of module record must appear in each object module and is one word in length.

# C TASK IMAGE FILE STRUCTURE

The task image as it is recorded on the disk appears in Figure C–1.

**Figure C–1 Task Image on Disk**



## C.1 Label Block Group

The label block group (shown in Example C–1) precedes the task on the disk and contains data that need not be resident during task execution together with up to two blocks containing device assignment data for LUNs 1-255. The task label blocks (first block in group) are read and verified by Install. The information in these blocks is used to fill in the task header.

**Example C–1 Label Block Group**

```
        .MACRO  LBLSY$  GBL
        .MCALL  DEFIN$
        .IF     IDN,<GBL>,<DEF$C>
        ...GBL=1
        .IFF
        ...GBL=0
        .ENDC

        DEFIN$  L$BTSK,0                ; TASK NAME (RAD50)
        DEFIN$  L$BPAR,4                ; DEFAULT PARTITION (RAD50)
        DEFIN$  L$BFLG,10               ; TASK FLAGS WORD
        DEFIN$  L$BPRI,12               ; DEFAULT PRIORITY
        DEFIN$  L$BLDZ,14               ; LOAD SIZE (32-WD BLOCKS)
        DEFIN$  L$BMXZ,16               ; INITIAL SPACE ALLOCATION (32-WD BLOCKS)
        DEFIN$  L$BPOL,20               ; NODE POOL LIMIT
        DEFIN$  L$BPIC,22               ; LIBRARY FLAGS WORD
        DEFIN$  L$BDAT,24               ; CREATION DATE
        DEFIN$  L$BLIB,32               ; RESIDENT LIBRARY REQUESTS (56. WORDS)
        DEFIN$  L$BHRB,212              ; HEADER BLOCK NUMBER
        DEFIN$  L$BAPR,214              ; STARTING APR (LIBRARY)
        DEFIN$  L$BEXT,216              ; DEFAULT TASK EXTENSION
        DEFIN$  L$BUIC,220              ; DEFAULT UIC
        DEFIN$  L$BROZ,222              ; READ-ONLY AREA SIZE (BYTES)
        DEFIN$  L$BROO,224              ; DISK OFFSET OF RO AREA
        DEFIN$  L$BROB,226              ; START ADDRESS OF RO AREA
        DEFIN$  L$BPAZ,230              ; TOTAL RO REGION SIZE (32-WD BLOCKS)
        DEFIN$  L$BHSZ,232              ; HEADER SIZE (32-WD BLOCKS)
        DEFIN$  L$BAPM,234              ; APR USAGE BITMAP
        DEFIN$  L$BASG,1000             ; LUN ASSIGMENT INFORMATION

;
; FLAG BITS DEFINITIONS
;

        DEFIN$  LD$ACC,100000           ; ACCESS REQUEST (1=RW, 0=RO)
        DEFIN$  LD$RSV,040000           ; APR RESERVATION FLAG
        DEFIN$  LD$REL,000004           ; PIC INDICATOR (1=YES)
        DEFIN$  LD$TYP,000002           ; BLOCK TYPE (0=COM, 1=LIB)
        DEFIN$  LD$DEF,000001           ; BLOCK DEFINED (1=YES)

        DEFIN$  LF$PIC,000001           ; LIB IS POSITION INDEPENDENT
        DEFIN$  LF$NHD,000002           ; TASK HAS HEADER (1=NO)
        DEFIN$  LF$FP ,000004           ; TASK HAS FP SAVE AREA (1=YES)

        .MACRO  LBLSY$  GBL
        .ENDM

        .ENDM   LBLSY$
```

## C.1.1 Label Block Details

The information contained in the label block is verified by the Install task in creating a system task directory (STD) entry for the task, and in linking the task to shareable global areas.

The definitions of the symbols used below may be obtained using the macro LBLSY$, which is defined in the system macro library. This macro may be given the argument 'DEF$G', in which case the definitions are made global.

- L$BTSK - Task name, consisting of two words in Radix-50 format. The value of this parameter is set by the TASK keyword.
- L$BPAR - Partition name, consisting of two words in Radix-50 format. Its value is set by the PAR keyword.
- L$BFLG - Task flag word containing bit values that are set or cleared depending on defined task attributes. Attributes are established by appending the appropriate switches to the task image file specification.

| | Bit | Attribute if Set=1 |
|---|---|---|
| SF.MU | 6 | Task is multi-user (/MU) |
| SF.PT | 7 | Task is privileged (/PR) |
| SF.XS | 10 | Task cannot receive data (/NOSEND) |
| SF.XA | 11 | Task is not abortable (/NOAB) |
| SF.XD | 12 | Task is not disableable (/NODIS) |
| SF.XF | 13 | Task is not fixable (/NOFIX) |
| SF.XC | 14 | Task is not checkpointable (/NOCH) |
| SF.SR | 15 | Task can be sent data and requested (/REQUEST) |

The symbolic names for these flags are not defined by LBLSY$.

- L$BPRI - Default priority, set by the PRI keyword.
- L$BLDZ - Load size of the task, expressed in multiples of 32-word blocks. The value of L$BLDZ is equal to the size of the root segment, in multi-segment tasks.
- L$BMXZ - Maximum size of the task, expressed in multiples of 32-word blocks. The header size is included.
- L$BMXZ - is used by Install to verify that the task fits into the specified partition.
- L$BPOL - Pool usage limit indicating maximum number of pool nodes that can be used simultaneously by the task. The default is 40 (decimal), which is overridden by the POOL keyword.
- L$BPIC - Flags for use by INSTALL:

| Flag | Interpretation if Set=1 |
|---|---|
| LF$PIC | Image is position independent |
| LF$NHD | Image has no header |

| Flag | Interpretation if Set=1 |
|---|---|
| LF$FP | Task has floating point save area in its header |
| LF$RO | Task has resident overlays |
| LF$HND | Task has header (1=no) |
| LF$SUP | Task linked to supervisor-mode library. |
| LF.SLB | Task is a supervisor-mode library mode request. |

- L$BDAT - Three words, containing the task creation date as 2-digit integer values, as follows:

```
YEAR (since 1900)
MONTH OF YEAR
DAY OF MONTH
```

- L$BHRB - Virtual block number of the task header. Between 2 and 4 depending on number of LUNs, as follows:

```
UNITS = 0 virtual block 2
UNITS = 1-128 virtual block 3
UNITS = 129-255 virtual block 4
```

- L$BAPR - Starting APR number if this image is a shareable global area. Calculated from BASE or TOP keywords.
- L$BEXT - The default number of words by which the memory allocated to a task at install time will be increased. This value is overridden by the /INC qualifier to INSTALL. Value is set with EXTTSK option of the Task Builder.
- L$BUIC - The UIC with which the task is built. Set by UIC keyword.
- L$BROZ - The size in bytes of the task read-only area. Zero if the task has no read-only area.
- L$BROO - Relative block number in the task image file of the start of the read-only area.
- L$BROB - Base virtual address of task read-only area (always on a 4K-word boundary).
- L$BPAZ - Total size (in 32-word blocks) of the task read-only region, including RO resident overlays.
- L$BHSZ - Task header size (in 32-word blocks).
- L$BAPM - Task APR usage bitmap. Bits 0-7 are set according to whether the corresponding APR is in use.

The following paragraphs describe components of the Shareable Global Area Name Block. An 8-word block is generated for each SGA referenced by the task. Because SGAs need not be installed in the system when the task is built, the Task Builder builds the block from the area's disk image, using information in the label blocks of that image.

- Library Name - A 2-word Radix-50 name specified in the LIBR or COMMON keyword.
- Creation Date - Obtained from the creation date in the shareable global area disk image label block.
- Starting Address - First address used to map the Shareable Global Area into the task addressing space.

The flags are used as follows:

| Flag | Meaning |
|---|---|
| LD$REL | Global area is PIC. Set if value of LF$PIC in the library image flags word (L$FLG) is =1. |
| | Global area is absolute. Cleared if LF$PIC in L$LFLG of global area image is 0. |
| LD$ACC | Read/Write access request. Set if RW specified in SGA option. |
| | Read-only ACCESS request. Cleared if RO specified in SGA option. |
| LD$CLS | Library is part of a cluster. |
| LD$SCL | Library is first library in a cluster. (Set in Phase 4 processing.) |
| LD$SUP | Library is a supervisor-mode library. |

## C.2 Header

The task is read into main memory starting at the base of the Header. Example C–2 illustrates the format of the fixed part. Futher details can be found in the IAS Executive Facilities Reference Manual. As shown in Figure C–1, the variable part includes the Logical Unit Table, the Attachment Descriptor Blocks and the Floating Point Save Area. The Logical Unit Table identifies to the Executive which device is assigned to which LUN. The Attachment Descriptors identify currently attached regions. The Floating Point Save Area is storage for the floating point registers when this option is requested. There may also be a System Reserved work area.

The Header is always a multiple of 32-word blocks. This ensures that the root segment code starts on a 32-word boundary, a requirement for the allocation of a APR pair of relocation registers. The Task Header is not covered by a task relocation register, and is therefore, not part of the virtual address space of the task.

The task header offsets may be defined using the macro HDRSY$, which is defined in the system macro library. The optional argument 'DEF$G' may be used to make the definitions global.

**Example C–2 Task Header Fixed Part**

```
.MACRO HDRSY$ GBL
.MCALL DEFIN$
.IF IDN,<GBL>,<DEF$G>
...GBL=1
.IFF
...GBL=0
.ENDC

DEFIN$ H.CR1 ,0 ; CONTEXT REFERENCE 1 (FP SAVE AREA POINTER)
DEFIN$ H.PD0 ,2 ; PAGE DESCRIPTOR REGISTER 0
DEFIN$ H.PD1 ,4 ; PAGE DESCRIPTOR REGISTER 1
DEFIN$ H.PD2 ,6 ; PAGE DESCRIPTOR REGISTER 2
DEFIN$ H.PD3 ,10 ; PAGE DESCRIPTOR REGISTER 3
DEFIN$ H.PD4 ,12 ; PAGE DESCRIPTOR REGISTER 4
DEFIN$ H.PD5 ,14 ; PAGE DESCRIPTOR REGISTER 5
DEFIN$ H.PD6 ,16 ; PAGE DESCRIPTOR REGISTER 6
DEFIN$ H.PD7 ,20 ; PAGE DESCRIPTOR REGISTER 7
```

Example C–2 Cont'd on next page

**Example C–2 (Cont.) Task Header Fixed Part**

```
DEFIN$ H.PA0 ,22 ; PAGE ADDRESS REGISTER 0
DEFIN$ H.PA1 ,24 ; PAGE ADDRESS REGISTER 1
DEFIN$ H.PA2 ,26 ; PAGE ADDRESS REGISTER 2
DEFIN$ H.PA3 ,30 ; PAGE ADDRESS REGISTER 3
DEFIN$ H.PA4 ,32 ; PAGE ADDRESS REGISTER 4
DEFIN$ H.PA5 ,34 ; PAGE ADDRESS REGISTER 5
DEFIN$ H.PA6 ,36  ; PAGE ADDRESS REGISTER 6
DEFIN$ H.PA7 ,40  ; PAGE ADDRESS REGISTER 7

DEFIN$ H.PF0 ,42 ; PAGE FLAGS REGISTER 0
DEFIN$ H.PF1 ,44 ; PAGE FLAGS REGISTER 1
DEFIN$ H.PF2 ,46 ; PAGE FLAGS REGISTER 2
DEFIN$ H.PF3 ,50 ; PAGE FLAGS REGISTER 3
DEFIN$ H.PF4 ,52 ; PAGE FLAGS REGISTER 4
DEFIN$ H.PF5 ,54 ; PAGE FLAGS REGISTER 5
DEFIN$ H.PF6 ,56 ; PAGE FLAGS REGISTER 6
DEFIN$ H.PF7 ,60 ; PAGE FLAGS REGISTER 7
DEFIN$ H.PF8 ,62 ; DUMMY PAGE FLAGS REGISTER TO STOP SCANS

DEFIN$ H.PL0, 64 ; PAGE LENGTH REGISTER 0
DEFIN$ H.PL1 ,66 ; PAGE LENGTH REGISTER 1
DEFIN$ H.PL2 ,70 ; PAGE LENGTH REGISTER 2
DEFIN$ H.PL3 ,72 ; PAGE LENGTH REGISTER 3
DEFIN$ H.PL4 ,74 ; PAGE LENGTH REGISTER 4
DEFIN$ H.PL5 ,76 ; PAGE LENGTH REGISTER 5
DEFIN$ H.PL6 ,100 ; PAGE LENGTH REGISTER 6
DEFIN$ H.PL7 ,102 ; PAGE LENGTH REGISTER 7

DEFIN$ H.PO0 ,104 ; PAGE OFFSET REGISTER 0
DEFIN$ H.PO1 ,106 ; PAGE OFFSET REGISTER 1
DEFIN$ H.PO2 ,110 ; PAGE OFFSET REGISTER 2
DEFIN$ H.PO3 ,112 ; PAGE OFFSET REGISTER 3
DEFIN$ H.PO4 ,114 ; PAGE OFFSET REGISTER 4
DEFIN$ H.PO5 ,116 ; PAGE OFFSET REGISTER 5
DEFIN$ H.PO6 ,120 ; PAGE OFFSET REGISTER 6
DEFIN$ H.P07 ,122 ; PAGE OFFSET REGISTER 7

DEFIN$ H.TPS ,124 ; TASK PROGRAM STATUS WORD
DEFIN$ H.TPC ,126 ; TASK PROGRAM COUNTER
DEFIN$ H.TR0 ,130 ; TASK R0
DEFIN$ H.TR1 ,132 ; TASK R1
DEFIN$ H.TR2 ,134 ; TASK R2
DEFIN$ H.TR3 ,136 ; TASK R3
DEFIN$ H.TR4 ,140 ; TASK R4
DEFIN$ H.TR5 ,142 ; TASK R5
DEFIN$ H.TSP ,144 ; TASK SP
DEFIN$ H.CR2 ,144 ; CONTEXT REFERENCE POINT 2 (NO STORAGE ALLOCATED)

DEFIN$ H.IPS ,146 ; INITIAL PROGRAM STATUS WORD
DEFIN$ H.IPC ,150 ; INITIAL PROGRAM COUNTER
DEFIN$ H.ISP ,152 ; INITIAL STACK POINTER

DEFIN$ H.DSV ,154 ; DEBUGGING SST VECTOR ADDRESS
DEFIN$ H.TSV ,156 ; TASK SST VECTOR ADDRESS
DEFIN$ H.DVZ ,160 ; DEBUGGING SST VECTOR SIZE
DEFIN$ H.TVZ ,161 ; TASK SST VECTOR SIZE
```

**Example C–2 Cont'd on next page**

**Example C–2 (Cont.) Task Header Fixed Part**

```
     DEFIN$ H.PUN ,162 ; POWERFAIL AST NODE ADDRESS
     DEFIN$ H.FEN ,164 ; FLOATING POINT EXCEPTION AST NODE ADDRESS
     DEFIN$ H.DUI ,166 ; DEFAULT UIC
     DEFIN$ H.UIC ,170 ; RUN UIC
     DEFIN$ H.HSZ ,172 ; HEADER SIZE (BLOCKS)
     DEFIN$ H.FZI ,174 ; FILE SIZE INDICATOR (OFFSET TO FIRST BLOCK PAST IMAGE)
     DEFIN$ H.REC ,176 ; RECEIVE AST NODE ADDRESS
     DEFIN$ H.RRA ,200 ; RECEIVE BY REF AST NODE ADDRESS
     DEFIN$ H.ADB ,202 ; OFFSET TO ATTACHMENT DESCRIPTOR BLOCKS
     DEFIN$ H.NADB,204 ; NUMBER OF ATTACHMENT DESCRIPTORS
     DEFIN$ H.TAT ,206 ; TASK ATTRIBUTES
     DEFIN$ H.RWZ ,210 ; SIZE OF READ/WRITE RESIDENT OVERLAY REGION
     DEFIN$ H.IOQ ,212 ; I/O REQUEST QUEUE, USED BY HANDLERS (2 WORDS)
     DEFIN$ H.EAF ,216 ; TASK HEADER FLAGS WORD
     DEFIN$ H.WNCT,220 ; WAIT-FOR-NODES RETRY COUNT
     DEFIN$ H.NML ,222 ; NETWORK MAILBOX LUN (USED BY DECNET)
     DEFIN$ H.ULC ,223 ; UNLOAD LOCK COUNT, FOR HANDLERS
; (NOT YET IMPLEMENTED)
     DEFIN$ H.PVDI,224 ; TASK DIRECTIVE PRIVILEGE (BYTE)
     DEFIN$ H.VNUM,225 ;++003 SYSTEM VERSION NUMBER
     DEFIN$ H.TAC,226 ; ++001 TASK ACCOUNTING INFO
; ++001 (2 WORDS)
; ++001 3RD WORD RESERVED
     DEFIN$ H.STLN,234 ; STL NODE ADDRESS FOR THIS TASK
     DEFIN$ H.SPCT,236 ; COUNT OF TASKS SPAWNED BY THIS ONE (BYTE)
     DEFIN$ H.PADB,240 ; ADB ADDRESS FOR TASK PURE AREA (USED AT
; FIRST TIME LOAD)
     DEFIN$ H.CKSM,242 ; HEADER CHECKSUM, SET BEFORE MOVING
; A TASK OUT OF MEMORY AND CHECKED ON
; RELOADING IT
     DEFIN$ H.AC  ,244 ; ACCOUNTING AREA POINTER
     DEFIN$ H.PTSM,246 ; PRIVILEGED TASK SEMAPHORE MASK
     DEFIN$ H.CHK,250 ; HEADER CHECK WORD (=S.DL+2)
     DEFIN$ H.RWAP,252 ; APR TO USE TO LOAD RW RESIDENT REGION
     DEFIN$ H.FXTK,254 ; STD ADDRESS OF TASK WHICH ISSUED THE FIX$
; DIRECTIVE WHICH FIXED A TASK, WHILE IT
; IS ACTIVE
     DEFIN$ H.MEX,256 ; ++001 MAXIMUM EXTENSION (SET BY TASK BUILDER)
     DEFIN$ H.LUT ,260 ; TASK'S LOGICAL UNIT TABLE
;
; FLAG BIT DEFINTIONS:
;
; PAGE FLAGS REGISTER (H.PFN):
;
     DEFIN$ PF.WIN,001 ; THIS IS FIRST APR OF A WINDOW
     DEFIN$ PF.WNO,002 ; THIS IS FIRST APR OF WINDOW ZERO
     DEFIN$ PF.CON,004 ; THIS IS A CONTINUATION OF PREVIOUS APR
     DEFIN$ PF.RAC,010 ; REGION HAS BEEN ACCESSED
     DEFIN$ PF.MAP,020 ; APR IS MAPPED ONTO REGION (H.PAN CONTAINS
; GCD NODE ADDRESS)
;
     DEFIN$ PF.RID,177400 ; HI BYTE CONTAINS REGION ID OF MAPPED REGION,
; OR ZERO IF THE REGION WAS SET UP AT INSTALL
; TIME (I.E. NOT DYNAMICALLY MAPPED)
;
; TASK ATTRIBUTES (H.TAT):
;
```

**Example C–2 Cont'd on next page**

**Example C–2 (Cont.) Task Header Fixed Part**

```
     DEFIN$ HT.FRQ,000001 ; TASK REQUIRES RECEIVE QUEUES TO BE FLUSHED
     DEFIN$ HT.NWD,000002 ; DON'T WAIT FOR NODES
     DEFIN$ HT.PRO,000004 ;++005 PRIVILEGED TASK DOESN'T MAP TO SCOMM
     DEFIN$ HT.SUP,000010 ;++006 Task has Supervisor mode save area
;
; TASK FLAGS (H.EAF):
;
     DEFIN$ HF.RMC,000001 ; MCR TO BE RECALLED ON TASK EXIT
     DEFIN$ HF.LPA,000002 ; LUNS PARTIALLY ASSIGNED, MUST BE COMPLETED
     DEFIN$ HF.SAV,000004 ;++002 SET IF TASK SAVED IN SYSTEM
;
; DIRECTIVE PRIVILEGE FLAGS (H.PVDI).  BITS ARE SET TO DISALLOW
; PARTICULAR DIRECTIVES (SEE EM10)
;
     DEFIN$ SF.RT ,001 ; TASK CANNOT ISSUE REAL-TIME DIRECTIVES
     DEFIN$ SF.PLS,002 ; TASK CANNOT ISSUE REGION-RELATED DIRECTIVES
;
; SYSTEM VERSION NUMBER.  USED TO PREVENT PRIVILEGED TASK IMAGES BUILT ON
; EARLIER SYSTEMS, FROM BEING INSTALLED.
;
     DEFIN$ HV.NUM,3 ;++004/003 SYSTEM VERSION NUMBER
; AFTER THIS, THERE ARE FOUR AREAS WHOSE SIZE DEPENDS ON THE TASK.
; THEY ARE DESCRIBED TOGETHER WITH ANY APPROPRIATE DEFINTIONS.
;
; LOGICAL UNIT TABLE (LUT):
;
;    THIS CONTAINS INFORMATION ABOUT THE TASK'S LOGICAL UNIT
;    ASSIGNMENTS.  THE FIRST WORD IS THE NUMBER OF ENTRIES
;    IN THE TABLE.  THE REST OF THE LUT CONTAINS TWO WORDS PER ENTRY:
;
;     WD.00 PUD ADDRESS OF DEVICE TO WHICH LUN IS ASSIGNED
;     WD.01 OPEN FILE INFORMATION (USED BY ACP TASK)
;
;
; ATTACHMENT DESCRIPTOR BLOCKS:
;
;    THESE CONTAIN INFORMATION ABOUT REGIONS TO WHICH THE TASK IS
;    ATTACHED.  THERE ARE TWO WORDS FOR EACH POSSIBLE REGION:
;
;     WD.00 RDL ADDRESS OF ATTACHED REGION
;     WD.01 LO BYTE - FLAGS
;      HI BYTE - RESERVED
;
;    THE FLAG BITS ARE:
;
     DEFIN$ RF.RED,001 ; TASK HAS READ ACCESS
     DEFIN$ RF.WRT,002 ; TASK HAS WRITE ACCESS
     DEFIN$ RF.EXT,004 ; TASK HAS EXTEND ACCESS
     DEFIN$ RF.DEL,010 ; TASK HAS DELETE ACCESS
     DEFIN$ RF.XDT,020 ; TASK NOT ALLOWED TO DETACH
     DEFIN$ RF.ITA,040 ; ATTACH DONE AT INSTALL TIME
; ALL OTHER BITS RESERVED
;
; FLOATING POINT SAVE AREA:
;
     THIS 25 WORD AREA IS USED TO STORE THE TASK'S FLOATING POINT
     CONTEXT, IF IT WAS SPECIFIED AT BUILD TIME THAT THE TASK USES
```

**Example C–2 Cont'd on next page**

**Example C–2 (Cont.) Task Header Fixed Part**

```
     THE FP11 FLOATING POINT UNIT.  THE FIRST WORD CONTAINS THE
;    SAVED FP STATUS WORD.  THE REMAINING 24 WORDS CONTAINS EACH
;    OF THE 6 64-BIT FLOATING POINT REGISTERS.
;
; Supervisor mode APR save area:
;
; Allocated by TKB when the task maps to supervisor mode libraries.  The
; task's supervisor mode PAR's and PDR's are stored here during a context
; switch.  This area is 8.*4  words (64. bytes) in length.
;
; TASK ACCOUNTING AREA:
;
;    (NOT YET DEFINED)

     .MACRO HDRSY$ GBL
     .ENDM HDRSY$
     .ENDM HDRSY$

     .MACRO LBLSY$ GBL
     .MCALL DEFIN$
     .IF IDN,<GBL>,<DEF$G>
     ...GBL=1
     .IFF
     ...GBL=0
     .ENDC

     DEFIN$ L$BTSK,0   ; TASK NAME (RAD50)
     DEFIN$ L$BPAR,4   ; DEFAULT PARTITION (RAD50)
     DEFIN$ L$BFLG,10   ; TASK FLAGS WORD
     DEFIN$ L$BPRI,12   ; DEFAULT PRIORITY
     DEFIN$ L$BLDZ,14   ; LOAD SIZE (32-WD BLOCKS)
     DEFIN$ L$BMXZ,16   ; INITIAL SPACE ALLOCATION (32-WD BLOCKS)
     DEFIN$ L$BPOL,20   ; NODE POOL LIMIT
     DEFIN$ L$BPIC,22   ; LIBRARY FLAGS WORD
     DEFIN$ L$BDAT,24   ; CREATION DATE
     DEFIN$ L$BLIB,32   ; RESIDENT LIBRARY REQUESTS (56. WORDS)
     DEFIN$ L$BHRB,212   ; HEADER BLOCK NUMBER
     DEFIN$ L$BAPR,214   ; STARTING APR (LIBRARY)
     DEFIN$ L$BEXT,216   ; DEFAULT TASK EXTENSION
     DEFIN$ L$BUIC,220   ; DEFAULT UIC
     DEFIN$ L$BROZ,222   ; READ-ONLY AREA SIZE (BYTES)
     DEFIN$ L$BROO,224   ; DISK OFFSET OF RO AREA
     DEFIN$ L$BROB,226   ; START ADDRESS OF RO AREA
     DEFIN$ L$BPAZ,230   ; TOTAL RO REGION SIZE (32-WD BLOCKS)
     DEFIN$ L$BHSZ,232   ; HEADER SIZE (32-WD BLOCKS)
     DEFIN$ L$BAPM,234   ; APR USAGE BITMAP
     DEFIN$ L$BASG,1000   ; LUN ASSIGMENT INFORMATION
```

**Example C–2 Cont'd on next page**

**Example C–2 (Cont.) Task Header Fixed Part**

```
************************
     DEFIN$ L$BXLN,26  ; 8.*<1$BLIB-L$BPAR> - Length of extra
; library descriptors (used only when
; linking to a supervisor mode library
************************
;
; FLAG BITS DEFINITIONS
;
     DEFIN$ LD$ACC,100000  ; ACCESS REQUEST (1=RW, 0=RO)
     DEFIN$ LD$RSV,040000  ; APR RESERVATION FLAG
     DEFIN$ LD$CLS,020000  ; Library is part of a cluster
     DEFIN$ LD$SCL,000200  ; Saved cluster attribute
     DEFIN$ LD$SUP,000010  ; Supervisor mode library
     DEFIN$ LD$REL,000004  ; PIC INDICATOR (1=YES)
     DEFIN$ LD$TYP,000002  ; BLOCK TYPE (0=COM, 1=LIB)
     DEFIN$ LD$DEF,000001  ; BLOCK DEFINED (1=YES)
     DEFIN$ LD$AMK,000060  ; APR mask bits

     DEFIN$ LF$PIC,000001  ; LIB IS POSITION INDEPENDENT
     DEFIN$ LF$NHD,000002  ; TASK HAS HEADER (1=NO)
     DEFIN$ LF$FP ,000004  ; TASK HAS FP SAVE AREA (1=YES)
     DEFIN$ LF$RO ,000010  ; TASK HAS RESIDENT OVERLAYS (1=YES)
     DEFIN$ LF$SUP,000020  ; Task linked to supervisor library
     DEFIN$ LF$SLB,000040  ; Task is a supervisor mode library
     .MACRO LBLSY$ GBL
     .ENDM

     .ENDM LBLSY$
```

## C.3 Low Memory Pointers

Several locations at the beginning of a task's virtual address space are reserved for system dependent information. These locations are as follows:

| | Address (Virtual) | Usage |
|---|---|---|
| 0 | $DSW | Directive Status Word. The Executive returns the completion code in this word for every system directive issued by the task. |
| 2 | .FSRPT | File Control Services work area and buffer pointer. |
| 4 | $OTSV | FORTRAN OTS work area pointer (that is, address of $OTSVA). |
| 6 | N.OVPT | Overlay Run Time system work area pointer. |
| 10 | $VEXT | Vector extension area pointer. |

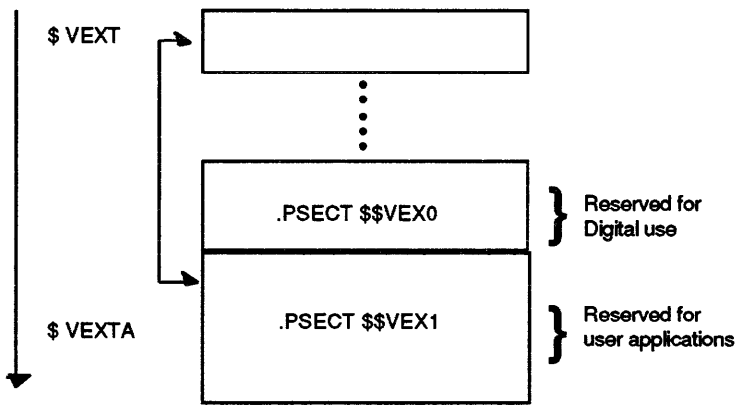
The last four of these locations contain addresses of work areas. These addresses are needed to provide re-entrancy capability to the associated system routines when these routines are placed in Shareable Global Areas.

Note that it is possible for a task to destroy these pointers if a stack overflow occurs.

The vector extension pointer ($VEXT) points to the vector extension area which contains addresses of impure work areas in the task.

Figure C–2 illustrates the format of the vector extension area. Each location within this region contains the address of an impure storage area that is referred to by subroutines; these subroutines must be re-entrant. Addresses below $VEXTA, referred to by negative offsets, are reserved for DIGITAL applications. Addresses above $VEXTA, referred to by positive offsets, are allocated for user applications.

**Figure C–2 Vector Extension Area Format**




The program sections $$VEX0 and $$VEX1 have the attributes D, GBL, RW, REL, and OVR.

The program section attribute OVR facilitates the definition of the offset to the vector and the initialization of the vector location at link time. For example:

```
                .GLOBL  $VEXTA      ;  MAKE SURE VECTOR AREA IS LINKED
                .PSECT  $$VEX1,D,GBL,RO,REL,OVR
BEG=.                               ;  POINT TO BASE OF POINTER TABLE
                .BLKW   N           ;  OFFSET TO CORRECT LOCATION
                                    ;  IN VECTOR AREA
LABEL:          .WORD   IMPURE      ;  SET IMPURE AREA ADDRESS
                                    ;  DEFINE OFFSET
OFFSET==LABEL-BEG
                .PSECT
IMPURE:
                    .
                    .
                    .
```

You should centralize all offset definitions within a single module from which the actual vector space allocation is made. Also, you should conditionalize the source to create two object modules: one that reserves the vector storage and, one that defines the global offsets which will be referred to by your resident library's subroutines.

Note that the sequence of instructions above intentionally redefines the global symbol. The Task Builder will report an error if this value differs from the centralized definition.

You can locate your vector through a sequence of instructions similar to the following:

```
MOV @#VEXT,R0          ;  GET ADDRESS OF VECTOR EXTENSIONS
MOV OFFSET(R0),R0      ;  POINT TO IMPURE AREA
.END
```

## C.4 Task R/W Root Segment

The low memory pointers, stack space and all R/W p-sections of the task root segment are concatenated by the Task Builder to form the R/W part of the root segment.

## C.5 READ/WRITE Overlays

Each read/write overlay segment (whether resident or not) is aligned on a disk block boundary.

## C.6 READ-ONLY Region

All read-only code, including the task pure area and any read-only resident overlays, is placed last in the task image file, starting on a disk block boundary. Each overlay starts on a 32-word boundary so that it can be mapped by the Memory Management Directives. Read-only resident overlays are not aligned to start on disk block boundaries, since thay are all loaded at the same time.

## C.7 Segment Table

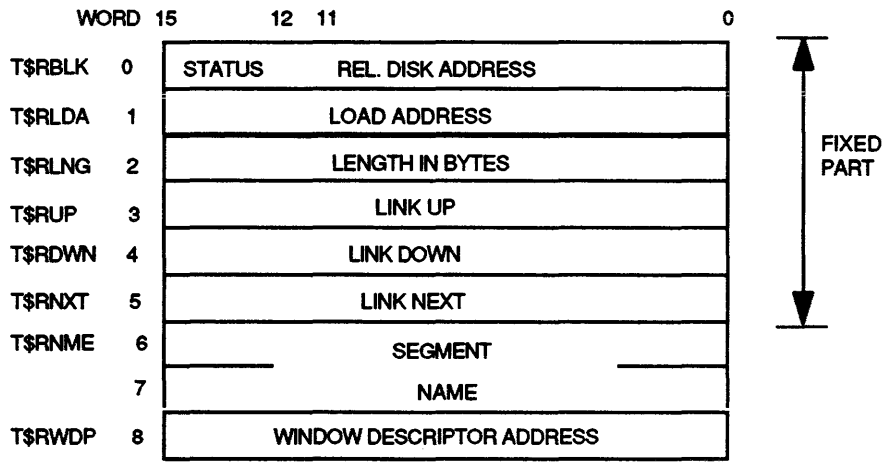
) The Segment Table contains a segment descriptor for every segment in the task. The segment descriptor is formatted as shown in Figure C–3. If the autoload method is used, the segment descriptor is six words in length. The table occupies a separate p-section called $$SGD1. A task may obtain the base and end addresses of the table as follows:

```
SEGTBL: .PSECT $$SGD0,OVR,D
SEGEND: .PSECT $$SGD2,OVR,D
```

This will define the symbol 'SEGTBL' to the base address of the segment table and 'SEGEND' to the first address beyond the segment table. If the manual load method is used, the segment descriptors are expanded to be eight words in length to include the segment names. If any overlays are resident, the descriptors are expanded to nine words to include the window pointers.

The offset names used below may be defined using the macro SEGDF$, which is defined in the system macro library. The optional argument 'DEF$G' may be used to make the definitions global.

**Figure C–3 Segment Descriptor**




## C.7.1 Status

The status bits are used in the autoload method to determine if an overlay is in memory, that is:

bit 12

0 = segment is in memory.

1 = segment is not in memory.

bit 13

0 = segment is not loaded

1 = segment is loaded

bit 14

0 = segment has disk allocation

1 = segment has no disk allocation (/NODSK)

bit 15

1 = (fixed setting)

## C.7.2 Relative Disk Address

Each segment begins on a block boundary and occupies a contiguous disk area to allow an overlay to be loaded by a single device access. The relative disk address is the relative block number of the overlay segment from the start of the task image. The maximum relative block number can not exceed 4096 since twelve bits are allocated for the relative disk address.

## C.7.3 Load Address

The load address contains the address into which the loading of the overlay segment starts.

## C.7.4 Segment Length

*Segment length* The segment length contains the length of the overlay segment in bytes and is used to construct the disk read.

## C.7.5 Link-Up

The link-up is a pointer to a segment descriptor away from the root.

## C.7.6 Link-Down

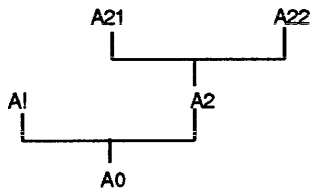
The link-down is a pointer to a segment descriptor nearer the root.

## C.7.7 Link-Next

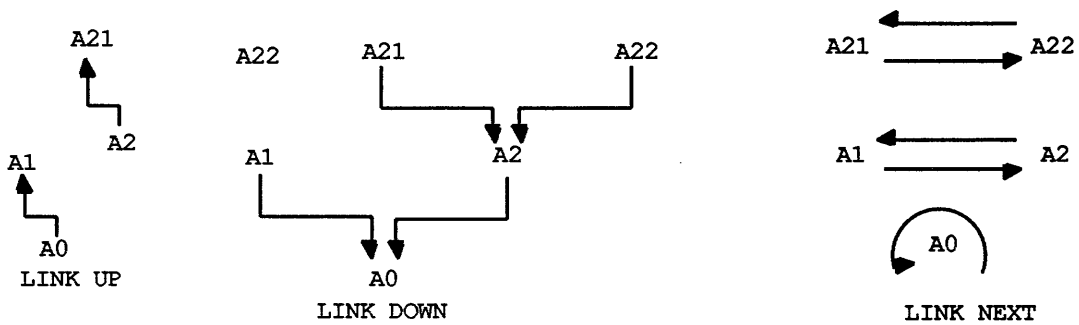
The link-next is a pointer to the adjoining segment descriptor. When a segment is loaded, the loading routine follows the link-next to determine if a segment in memory is being overlaid and should therefore be marked out-of-memory.

The link-next pointers are linked in a circular fashion:

Consider the tree:




The segment descriptors are linked in the following way:




If there is a co-tree, the link-next of the segment descriptor for the root points to the segment descriptor for the root segment of the co-tree.

## C.7.8 Segment Name

This field contains the 2-word radix-50 segment name. It is present only if the global symbol $LOAD is defined or referenced in the task.

## C.7.9 Window Descriptor Address

This field contains the address of the window descriptor for this overlay. It is present only if the task contains resident overlays.

# C.8 Autoload Vectors

Autoload vectors appear in every segment that references autoload entry points in segments that are farther from the root than the referencing segment.

The autoload vector table consists of one entry per autoload entry point in the form shown in Figure C–4.

**Figure C–4 Autoload Vector Entry**

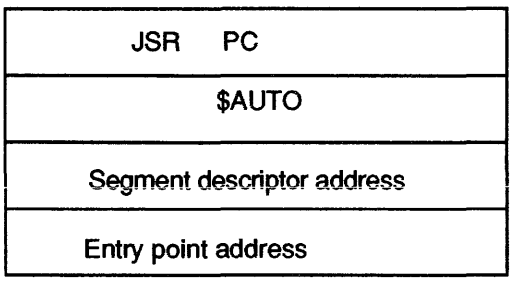



**Figure C–5 Window Descriptor**

| WORD | | |
|---|---|---|
| 0 | BASE APR | WINDOW ID |
| 1 | VIRTUAL BASE ADDRESS | |
| 2 | WINDOW SIZE IN 64-BYTE BLOCKS | |
| 3 | REGION ID | |
| 4 | OFFSET IN PARTITION | |
| 5 | LENGTH TO MAP | |
| 6 | STATUS WORD | |
| 7 | SEND/RECEIVE BUFFER ADDRESS (0) | |
| 8 | FLAGS WORD | |
| 9 | ADDRESS OF REGION DESCRIPTOR | |

## C.8.1 Window Descriptor

Window descriptors are allocated only if a structure containing memory-resident overlays is defined. Window descriptors are shown in Figure C–5.

Words 0 through 7 constitute a window descriptor in the format required by the mapping directives. The region ID is set by TKB unless the memory-resident overlay is part of a shared global area.

Words 8 and 9 contain additional data that is referenced by the overlay routines. Bit 15 of the flag's word, if set, indicates that the window is currently mapped into the task's address space. Word 9 contains the address of the associated region descriptor.

If the memory-resident overlay is not part of a shareable global area, this value is zero.

## C.8.2 Region Descriptor

The Region Descriptor is allocated only when the memory-resident overlay structure is part of a shared region. Region descriptors are shown in Figure C–6.

**Figure C–6 Region Descriptor**

| WORD | |
|---|---|
| 0 | REGION ID |
| 1 | SIZE OF REGION |
| 2 | REGION |
| 3 | NAME |
| 4 | REGION |
| 5 | PARTITION |
| 6 | REGION STATUS |
| 7 | PROTECTION CODES (ALWAYS 0) |
| 8 | FLAGS |

Words 0 through 7 constitute a region descriptor in the format required by the mapping directives. The flag's word is referenced by the overlay load routine. Bit 15 of the flag's word, when set, indicates that a valid region identification is in word 0. If this bit is clear, the overlay load routine issues an Attach Region directive (with protection code set to zero) to obtain the identification.

# D RESERVED SYMBOLS

Several global symbol and p-section names are reserved for use by the Task Builder. Special handling occurs when a definition of one of these names is encountered in a task image. This happens, for example, when a system library module containing the definition is built into a task for a particular purpose.

The definition of a reserved global symbol in the root segment causes a word in the Task Image to be modified with a value calculated by the Task Builder. The relocated value of the symbol is taken as the modification address.

The following global symbols are reserved by the Task Builder:

**Table D–1 Reserved Global Symbols**

| Global Symbol | Modification Value |
|---|---|
| .MOLUN | Error message output device. |
| .NIOST | Two word I/O status block containing the results of the load overlay request. |
| .NLUNS | The number of logical units explicitly used by the task, not including the Message Output and Overlay units. |
| .NOVLY | The overlay logical unit number. |
| .NSTBL | Reserved. |
| .TRLUN | The trace subroutine output logical unit number. |
| .ODTL1 | Logical unit number for the ODT terminal I/O device. |
| .ODTL2 | Logical unit number for the ODT listing device. |
| $OTSV | The address in low memory of the FORTRAN OTS work area ($OTSVA defined by the FORTRAN OTS). |
| .PILUN | Location containing LUN for communication with the Timesharing Control Primitives. |

The definition of one of the following reserved p-sections causes the task builder to extend that p-section if the appropriate option input is specified (see Chapter 5, Section "EXTSCT").

**Table D–2 Reserved P-sections**

| Section Name | Extension Length |
|---|---|
| $$DEVT | The extension length (in bytes) is calculated from the formula<br>EXT = <S.FDB+52>*UNITS<br>where the definition of S.FDB is obtained from the root segment symbol table and UNITS is the number of logical units used by the task, excluding the Message Output, Overlay, and ODT units. |
| $$FSR1 | The extension of this section is specified by the ACTFIL option input. |
| $$IOB1 | The extension of this section is specified by the MAXBUF option input. |

Table D–2 (Cont.) Reserved P-sections

| Section Name | Extension Length |
|---|---|
| $$OBF1 | FORTRAN OTS uses this area to parse array type format specifications. May be extended by FMTBUF keyword. |

The following p-section names and symbols are also reserved:

Table D–3 Reserved P-sections and Symbols

| Section Name or Symbol | Contents |
|---|---|
| $$ALVC | P-section containing the auto load vectors. Exists in every segment of an overlaid task. |
| $$FSR2 | Contains the first free location after the file storage region p-section $$FSR1. |
| $$RESL | P-section of the system resident library SYSRES. |
| $$RESM | P-section of the system resident library SYSRES. |
| $$RGDS | Contains region descriptions for overlaid shareable global areas. |
| $$RTS | P-section containing the RETURN instruction used when referencing segments with the GBL attribute. |
| $$SGD0 | Defines the start of the segment table p-section $$SGD1. |
| $$SGD1 | P-section containing the segment table (see Appendix C, Section C.7.) |
| $$SGD2 | Contains the first free location after $$SGD1. |
| $$WNDS | Contains window descriptions for resident overlays. |
| $$$ODT | P-section containing the system debugging aid. |

# E INCLUDING A DEBUGGING AID

To include a program which controls the execution of a task, you name the appropriate object module as an input file and apply the /DEBUG PDS qualifier (/DA MCR switch).

When a debugging aid program is input, the Task Builder causes control to be passed to the program when the task execution is initiated.

Such control programs might trace a task, printing out relevant debugging information, or monitor the task's performance for analysis.

The switch has the following effect:

1 The transfer address in the debugging aid overrides the task transfer address.

2 On initial task load, the following registers have the indicated value:

R0 - Transfer address of task
R1 - Task name in Radix-50 format (word #1)
R2 - Task name (word #2)

The following points must be taken into consideration when using debugging aids on a task (particularly ODT):

1 Breakpoints cannot be set in R-O p-sections. If such program sections are to be debugged, the task should be re-linked with the /READ_WRITE PDS qualifier (/RW MCR switch).

2 Care must be used if setting breakpoints in overlay branches.

3 Control always passes to $ALBP2 immediately before returning to the users program after an autoload of an overlay.

Examples:

```
1.  PDS> LINK/DEBUG/READ_WRITE FAULTY

    or

    MCR> TKB
    TKB> FAULTY/DA/RW=FAULTY
    TKB>/
    ENTER OPTIONS:
    TKB> SGA=SYSRES:RO
    TKB>//

    Use the default debugging and ODT, to debug  task,  including
    its read-only areas.

2.  PDS> LINK/DEBUG:[1,1]DDT/SYMBOLS/READ_WRITE BADPRG

    or

    MCR> TKB
    TKB> BADPRG/RW,,BADPRG=BADPRG,[1,1]DDT/DA
    TKB>/
    ENTER OPTIONS:
    TKB> SGA=SYSRES:RO
    TKB>//
```

```
Use the debugging aid [1,1]DDT to debug task,  including  its
read-only  areas,  also create a task symbol table to be used
during the debugging dialogue.
```

# F IMPROVING TASK BUILDER PERFORMANCE

This appendix contains procedures and suggestions to assist in maximizing Task Builder performance. Procedures are given for:

1 Evaluating and improving Task Builder throughput

2 Modifying command switch defaults to provide a more efficient user interface

The procedures given here may require relinking the Task Builder. Modifications to the Task Builder build file imply using one or more of the following files located under UFD [11,11]:

```
TKBBLD.CMD
SLOTKBBLD.CMD
```

These files are on the object distribution medium, together with the library and ODL files required for building TKB.

## F.1 Evaluating and Improving Task Builder Performance

Task Builder throughput is determined by two factors:

1 The amount of memory available for table storage

2 The amount of disk latency due to input file processing

The discussion in the following paragraphs outlines methods for improving throughput in these two cases. The methods approach their goals through judicious use of system resources and Task Builder features.

### F.1.1 Table Storage

The principal factor governing Task Builder performance is the amount of memory available for table storage. To reduce memory requirements, a work file is used to store symbol definitions and other tables. As long as the size of these tables is within the limits of available memory, the contents of this file are kept in core and the disk is not accessed. If the tables exceed this limit, some information must be displaced and moved to the disk, degrading performance accordingly.

Work file performance can be gauged by consulting the statistics portion of the Task Builder Map. The following parameters are displayed:

**Number of work file references:**

Total number of times that work file data was referenced.

**Work file reads:**

Number of work file references that resulted in disk accesses to read work file data.

**Work file writes:**

Number of work file references that resulted in disk accesses to write work file data.

**Size of core pool:**

amount of in-core table storage in words. this value is also expressed in units of 256-word pages (information is read from and written to disk in blocks of 256 words).

**Size of work file:**

Amount of work file storage in words. If this value is less than the core pool size, the number of work file reads and writes is zero. That is, no work file pages are removed to the disk. This value is also expressed in pages (256-word blocks).

**Elapsed time:**

Amount of time required to build the task image, and output the map. This value excludes odl processing, option processing, and the time required to produce the global cross-reference.

The overhead for accessing the work file can be reduced in one or more of the following ways:

1 By increasing the amount of memory available for table storage

2 By placing the work file on the fastest random access device

3 By decreasing system overhead required to access the file

4 By reducing the number of work file references

The task builder extends itself as necessary (using the EXTK$ directive) up to the limit set by the MAXEXT option when the task builder is linked or by the SET EXTENDED_TASK_SIZE PDS command or the SET /MAXEXT MCR command.

As distributed, the maximum extension for the task builder is 2000 (the default value).

The work file resides on the device WK0. It may be possible to improve performance by redirecting this device to a faster disk at System Generation or system startup.

System overhead for work file accesses is incurred in translating a relative block number in the file to a physical disk address. To minimize this overhead, the Task Builder requests disk space in contiguous increments. The size of each increment is equal to the value of symbol W$KEXT defined in the Task Builder build file. A larger positive value causes the file to be extended in larger contiguous increments and reduces the overhead required to access the file.

The increment should be set to a reasonable value because the Task Builder resorts to noncontiguous allocation whenever contiguous allocation fails.

The size of the work file can be reduced by:

1 Linking the user's task to a core-resident library containing commonly used routines (for example, FORTRAN Object Time System) whenever possible.

2 Including common modules, such as components of an object time system, in the root segment of an overlaid task.

3 Reducing to one the number of times the library and symbol definition modules appear in the task, by moving them nearer the root.

4 Using the /SELECT qualifier on symbol table files that describe absolute symbol definitions.

5 Using an object library or file of concatenated object modules if many modules are to be linked.

In the last two cases, system overhead is also significantly reduced because fewer files must be opened to process the same number of modules.

The number of work file references can be reduced by eliminating unnecessary output files and cross-reference processing, or by obtaining the short map. In addition, selected files such as the default system object module library, can usually be excluded from the map using the /NOMAP qualifier. In this case, a full map can be obtained at less frequent intervals and retained.

The following procedures summarize the above suggestions for improving work file performance:

1 Use the MAXEXT option so that the task builder can extend automatically to obtain more table space.

2 Reduce disk latency by placing the work file on the fastest random access device.

3 Reduce system overhead by modifying the command file to allocate work file space in larger contiguous increments.

4 Decrease work file size by using resident libraries, concatenated object files, and object libraries.

5 Decrease work file size by moving common modules into the root segment of an overlaid task.

6 Decrease the number of work file references by eliminating the map and global cross-reference, obtaining the short map, or excluding files from the map.

### F.1.2 Input File Processing

The suggestions for minimizing the size of the work file and number of work file accesses also drastically reduce the amount of input file processing.

A given module can be read up to four times when building the task:

1 To build the symbol table

2 To produce the task image

3 To produce the long map

4 To produce the global cross-reference

Files that are excluded from the long map are read only twice. The third and fourth passes are completely eliminated for all modules when a short map is requested without a global cross-reference.

## F.2 Modifying Command Level Defaults

The task builder contains internal switches which represent the default characteristics which it applies to a task when qualifiers (PDS) or switches (MCR) are not included in the command. The defaults in the released version of the Task Builder may not suit the requirements of all installations. For example, the default /FLOATING_POINT (/FP) (Floating Point Processor) would be unsatisfactory at an installation that did not have this hardware.

The user can tailor many of the defaults by altering the contents of the words that contain initial switch states. Modifying the Task Builder in this way is a three-step process as follows:

1 Consult the tables below to determine the switch word and bit to be altered.

2 Edit the appropriate Task Builder command file to include the switch word modification through a GBLPAT option referencing the global symbol switch word name.

3 Relink the Task Builder using the modified command file.

The command files for system tasks as provided with the released system require the standard set of Task Builder defaults; therefore, it is necessary to retain and use an unmodified copy of the Task Builder whenever such tasks are relinked.

The tables given are used to alter the defaults as follows:

1 Identify the qualifier (PDS) or command switch (MCR) and, if using MCR, the file to which it applies.

2 If using MCR, consult the file entry in each table to locate the applicable switch words.

3 Scan the entries until the switch mnemonic is found. Only those switches which may be changed are included in the tables.

4 OR the desired state of the associated bit with the initial contents to obtain the new set of defaults.

5 Supply the revised value and switch word name as arguments in a GBLPAT option. The switch words are in the TASKB segment.

6 Relink the Task Builder to produce a version containing the appropriate defaults.

Example:

To change the Task Builder Floating Point Processor default to /NOFLOATING_POINT (/-FP), the steps described below are performed.

By consulting Table F-1 the user determines that two qualifier words, $DFSWT and $DFTSK contain task file qualifiers. Of these, $DFTSK contains the default setting for the /FLOATING_POINT (/FP) switch in bit 14. Setting this bit to 0 changes the initial state to /NOFLOATING_POINT (/-FP). This new value is combined with the initial contents to yield the revised setting 4040. The required keyword input is:

```
        OPTIONS?  GBLPAT=TASKB:$DFTSK:4040
or
        TKB>GBLPAT=TASKB:$DFTSK:4040
```

**Note: The state of bit positions not listed in the table must not be altered.**

**Table F–1 Task File Defaults**

For time-sharing systems only:

**File:** Task File

**Switch Word:** $DFSWT

**Initial Contents:** 10

**Bit Settings:**

| Bit | Condition if Set to 1 | |
|---|---|---|
| 15 | EXIT (XT) | Task Builder exits after errors |
| 11 | SEQUENTIAL (SQ) | Sequential .PSECT allocation |
| 7 | NORUN (-OR) | No run-time system |
| 4 | FULL_SEARCH (FU) | Full overlay tree search |
| 3 | NORES (-RO) | No resident overlays in task |
| 2 | REQUEST (SR) | All send and request/resume accepted |

For time-sharing systems only:

**File:** Task file

**Switch Word:** $DFTSK

**Initial Contents:** 44040

**Bit Settings:**

| Bit | Condition if Set to 1 | |
|---|---|---|
| 15 | NOCHECKPOINT (-CP) | Not checkpointable |
| 14 | FLOATING (FP) | Floating Point Processor |
| 13 | NOWAIT (-WN) | No waiting for nodes |
| 12 | NOHEADER (-HD) | No header |
| 11 | NOFIXABLE (-FX) | Not fixable |
| 10 | DEBUG (DA) | Debugging Aid |
| 9 | POSITION_INDEPENDENT (PI) | Position-Independent |
| 8 | PRIVILEGED (PR) | Privileged |
| 7 | TRACE (TR) | Trace |
| 6 | NOABORTABLE (-AB) | Not abortable |
| 5 | FLUSH (FR) | Flush receive queues on exit |
| 4 | NORECEV (-SE) | Cannot receive sent data |
| 3 | MULTIUSER (MU) | Multiuser |
| 2 | NODISABLABLE (-DS) | Cannot be disabled |
| 1 | READ_WRITE (RW) | Read-only attribute ignored |

**Table F–2 Map File Defaults**

For time-sharing systems only:

**File:** Map file

**Switch Word:** $DFLBS

**Initial Contents:** 120000

**Bit Settings:**

| Bit | Condition if Set to 1 | |
|---|---|---|
| 15 | NOFULL (-MA) | Do not include system library and STB files in map |

For time-sharing systems only:

**File:** Map file

**Switch Word:** $DFMAP

**Initial Contents:** 2040

**Bit Settings:**

| Bit | Condition if Set to 1 | |
|---|---|---|
| 10 | NOFILES (SH) | Short map |
| 6 | CROSS_REFERENCE (CR) | CREF |
| 5 | WIDE (WI) | Wide format |
| 1 | NOUNDEFINED_REFERENCES (-UR) | Do not print undefined references |

**Table F–3 Symbol Table File Defaults**

For time-sharing systems only:
**File:** Symbol table
**Switch Word:** $DFSTB
**Initial Contents:** 0
**Bit Settings:**

| Bit | Condition if Set to 1 | |
|---|---|---|
| 12 | NOHEADER (-HD) | Build task without header |
| 9 | POSITION_INDEPENDENT (PI) | Task is position-independent |
| 0 | NOUNDEFINED_SYMBOLS (-UN) | Do not reference undefined symbols |

**Table F–4 Input File Defaults**

For time-sharing systems only:
**File:** Input file
**Switch Word:** $DFINP
**Initial Contents:** 100
**Bit Settings:**

| Bit | Condition if Set to 1 | |
|---|---|---|
| 15 | NOMAP (-MA) | Do not include file contents in map |
| 6 | CONCATENATED (CC) | File may contain two or more concatenated object modules |

## F.3 The Slow Task Builder

TKB.TSK uses a symbol table structure that can be searched quickly, but which requires more work file space than previous versions. If the message

```
NO VIRTUAL MEMORY STORAGE AVAILABLE
```

is issued, the user should attempt to reduce work file size as described previously. Assuming these methods fail, another version of the Task Builder can be linked, which requires less storage but runs considerably slower. The build file is SLOTKBBLD.CMD, which resides on the same device and UFD as the other Task Builder command files.

# IAS Task Builder Glossary

**AUTOLOAD**: The method of loading overlay segments, in which the Overlay Runtime System automatically loads overlay segments when they are needed and handles any unsuccessful load requests.

**CO-TREE**: An overlay tree whose segments, including the root segment, are made resident in memory through calls to the Overlay Runtime System.

**EXECUTIVE PRIVILEGED TASK**: A task that has privileged memory access rights. An executive privileged task can access the Executive and the external page in addition to its own partition and referenced shareable global areas.

**GLOBAL SYMBOL**: A symbol whose definition is know outside the defining module.

**MAIN TREE**: An overlay tree whose root segment is loaded by the Executive when the task is made active.

**MANUAL LOAD**: The method of loading overlay segments in which the user includes explicit calls in his routines to load overlays and handles unsuccessful load requests.

**MEMORY ALLOCATION FILE**: The output file created by the Task Builder that describes the allocation of task memory.

**OVERLAY DESCRIPTION LANGUAGE**: A language that describes the overlay structure of a task.

**OVERLAY RUNTIME SYSTEM**: A set of subroutines linked as part of an overlaid task that are called to load segments into memory.

**OVERLAY SEGMENT**: A segment that shares storage with other segments an is loaded when it is needed.

**OVERLAY STRUCTURE**: A structure containing a main tree and optionally one or more co-trees.

**OVERLAY TREE**: A tree structure consisting of a root segment and optionally one or more overlay segments.

**PATH**: A route that is traced from one segment in the overlay tree to another segment in that tree.

**PATH-DOWN**: A path toward the root of the tree.

**PATH-UP**: A path away from the root of the tree.

**PATH-LOADING**: The technique used by the autoload method to load all segments on the path between a calling segment an a called segment.

**P-SECTION:** A section of memory that is a unit of the total allocation. A source program is translated into object modules that consist of p-sections with attributes describing access, allocation and relocatability. (See Chapter 6, Section 6.1.9 for a complete description).

**ROOT SEGMENT:** The segment of an overlay tree that, once loaded, remains in memory during the execution of the task.

**RUNNABLE TASK:** A task that has a header and stack and that can be installed and executed.

**SHAREABLE GLOBAL AREA:** A code and/or data area which can be shared by many tasks. The area is resident only when one or more referencing tasks are active. An SGA is linked using the Task Builder. SGAs used by a task are linked to it using the Task Builder. See the IAS *Executive Facilities Reference Manual* for a full definition of SGAs.

**SEGMENT:** A group of modules and/or p-section that occupy memory simultaneously and that can be loaded by a single disk access.

**SYMBOL DEFINITION FILE:** The output file created by the Task Builder that contains the global symbol definitions and values in a format suitable for reprocessing by the Task Builder. Symbol definition files are used to link tasks to shareable global areas.

**TASK IMAGE FILE:** The output file created by the Task Builder that contains the executable portion of the task. It may contain a task or a shareable global area.

# Index

## A



## B



## C

# D



# E



# F



# G

# H



# I



# L



# M



# O

# P



# R

# S

# T



# U



# V

# W

Do Not Tear - Fold Here and Tape

digital™

No Postage
Necessary
if Mailed in the
United States

**BUSINESS REPLY MAIL**

FIRST CLASS PERMIT NO 33 MAYNARD MASS

POSTAGE WILL BE PAID BY ADDRESSEE

IAS Engineering/Documentation
Digital Equipment Corporation
5 Wentworth Drive GSF/L20
Hudson, NH 03051-4929

Do Not Tear - Fold Here

## Reader's Comments

This form is for document comments only. Digital will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of user/reader that you most nearly represent:

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify)_____

Name_____Date_____

Organization_____

Street_____

City_____State_____Zip Code_____
or Country