Program Duplication and Block Assignment [6]

This section describes two related topics, duplication and block assignment. Duplication occurs when more than one module, entry point, or common block has the same name. The loader handles duplication differently for segmented and nonsegmented programs. Block assignment refers to the process the loader uses to position all modules and common blocks that you have not explicitly assigned.

Duplication and block assignment in nonsegmented programs
6.1

In a nonsegmented program, there is no duplication of modules, entry points, or common blocks.

Duplicate module names
6.1.1

In a nonsegmented load, you can load modules with duplicate names, although this is not recommended because it may result in misleading entry-point definitions, load maps, and debugging.

You can use the MODULES directive with a file specifier to make the loader load a module from a particular file.

Duplicate entry-point names
6.1.2

In a nonsegmented load, each entry point (external definition) must have a unique name. The loader uses the entry point defined in the first module loaded and ignores all subsequent entry points with the same name except to issue a warning message (see "DUPORDER directive," page 32). You can control the printing of duplicated entry-point messages by using the DUPENTRY directive.

Some of a module's entry points can be used in the load while others are ignored. The EPXRF parameter in the MAP directive causes the loader to print the Entry Point Cross-reference table, which notes all active and ignored (inactive) entry points.

Duplicate common blocks 6.1.3

Only one common block with a particular name is loaded in a nonsegmented load. The loader assumes that all common blocks with the same name are the same common block (this includes common blocks specified in modules that are never called and, thus, are not loaded).

The loader considers a common block's size to be the largest size encountered in the relocatable modules actually included in the program. You can override this size limit by using the `COMMONS` directive.

Block assignment 6.1.4

All modules and common blocks in a nonsegmented program are assigned to the single segment that makes up the program.

Duplication in segmented programs 6.2

In segmented programs, each segment may contain a module, an entry point, and a common block, each with the same name. Duplication can arise from your use of the `DUP`, `MODULES`, and `COMMONS` directives, or it can arise automatically, as a side effect of using the `FLOAT` directive.

Module duplication 6.2.1

You can manually load copies of the same module or different modules with the same name into different segments. Each segment may have only one module of a particular name.

Duplicate a module by using the `DUP` directive or by using the `MODULES` directive to place the duplicated modules in the desired segments. The loader handles duplicated entry-point names automatically, provided that you have duplicated the modules in your directives.

The loader must know where to put all duplicate module names before encountering the modules. Therefore, you must use the `MODULES` directive to assign all duplicate modules and their callers to the appropriate segment.

Entry-point duplication 6.2.2

Every active entry point in each segment must have a unique name. You must assign all modules containing duplicated entry points and all modules referencing duplicated entry points.

A module referencing a duplicated entry point is linked to the entry point in the same segment. If no entry point with the requested name is in the same segment as the calling module, there can be only one entry point with the duplicated name on the branch.

For example, assume that module `X` in segment B is in dataset `BIN1` and that another module `X` in segment E is in `BIN2`. Also assume that the module name and the entry-point name are the same, and that `W` calls the `X` in segment B, and `Y` calls the `X` in segment E.

Common block duplication 6.2.3

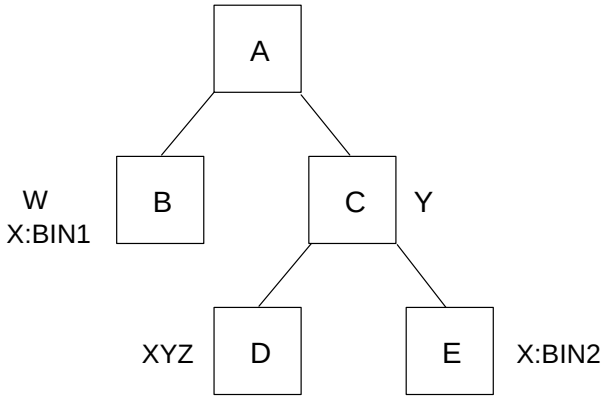
In a segmented load, you can load common blocks with the same name into different segments. Use the `COMMONS` directive to place the duplicated common blocks in the desired segments. You must also use the `MODULES` directive to assign every module that references a duplicated common block to the module you desire.

A module referencing a duplicated common block is linked to the common block in the same segment. If there is no common block with the requested name in the same segment as the referencing module, there can be only one common block with the duplicated name on the branch.

Rules for references to duplicated common blocks are the same as the rules for duplicated entry points.

Figure 8 shows the directives required to obtain this description.

```
TREE
  A(B,C)
  C(D,E)
ENDTREE
SEGMENT=A
  BIN=BIN3
ENDSEG
SEGMENT=B
  MODULES=X:BIN1,W
ENDSEG
SEGMENT=C
  MODULES=Y
ENDSEG
SEGMENT=D
  MODULES=XYZ
ENDSEG
SEGMENT=E
  MODULES=X:BIN2
ENDSEG
```



Segment assignments:

Segment containing duplicated entry point	Segment calling duplicated entry point	Comments
B,D	B,C	Calls from B are linked to the copy in B. Calls from C are linked to the copy in D.
C,E	C,E	Calls from C are linked to the copy in C. Calls from E are linked to the copy in E.
D,E	A,C	Illegal; both calls are ambiguous.
B,C	B,C,D,E	Calls from B are linked to the copy in B. All others are linked to the copy in C.
B,C	A	Illegal; reference is ambiguous.
B,B	Anywhere	Illegal; cannot have two copies in the same segment.

Figure 8. Entry-point duplication example

Common blocks loaded into different segments are considered unique because they occupy different memory locations. Modules that reference duplicated common blocks must be assigned to different segments to ensure that the program contains no ambiguous references to common block data. (See "`COMMONS` and `SCOMMONS` directives," page 31.)

For example, if common block `/ABC/` were included in segments B and C in the segment tree in Figure 9, a reference to `/ABC/` from a module in segment A would be ambiguous.

In Figure 9, assume that a copy of `/ABC/` has been included in both segments B and C. References from segments C, D, and E would be relocated to the `/ABC/` common block in segment C. References to `/ABC/` from segment B would be relocated to the `/ABC/` common block in segment B.

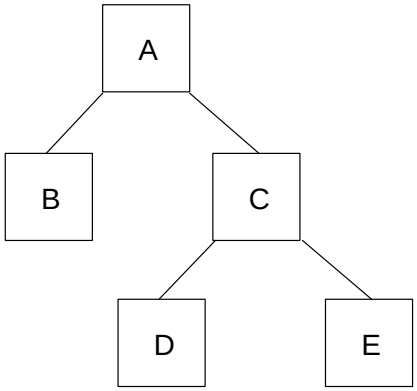


Figure 9. Segment tree with duplicate common blocks

Block assignment in segmented programs
6.3

After you have indicated the segmentation structure and assigned certain modules and common blocks to segments, the loader assigns any remaining movable blocks to segments. A movable block is any module or common block that you have not explicitly assigned to a segment. The loader uses one of two methods to assign movable blocks: floating or automatic duplication. The `FLOAT` directive lets you choose which of these two methods the loader uses.

FLOAT *directive*
6.3.1

FLOAT is a global directive. It selects the method the loader uses to handle movable blocks.

Format:

```
FLOAT=ANY|NONE
```

ANY — Enables movable block floating (default).

NONE — Disables movable block floating and enables automatic duplication of movable blocks.

Floating
6.3.2

When floating is enabled, the loader "floats" each movable block up the tree structure to the lowest segment (the one farthest from the root segment) that is a predecessor common to all segments in which the movable block is referenced. The block is thus resident in memory when any segment references it. If the movable block is a common block, all modules that reference it access the same memory space. Floating is the faster of the two methods for loading, and it yields the smallest overall program.

Automatic duplication
6.3.3

When automatic duplication is enabled, the loader assigns a copy of each movable block to each segment that references it, unless a copy of the block has been assigned to a predecessor segment of that block. The block is duplicated automatically in the target segment as if a MODULES or COMMONS directive had positioned it there. References to a block access unique copies of the block unless it has been assigned to a common predecessor of the modules referencing it. Automatic duplication takes longer to load than floating, and it generates a larger overall program, but it may generate a program that requires less memory to execute. It also allows access to a unique copy of automatically duplicated common blocks.

Example
6.3.4

The following examples show the assignment of movable blocks by floating and automatic duplication. Consider the following partial Fortran program and associated loader directives:

```
PROGRAM EXAMPLE
CALL SUB1
CALL SUB2
CALL ASUB
END

SUBROUTINE SUB1
COMMON /ACOM/ J(200)
CALL BSUB
CALL ASUB
END

SUBROUTINE SUB2
CALL BSUB
CALL SUB2A
CALL SUB2B
END

SUBROUTINE SUB2A
COMMON /BCOM/ I(100)
END

SUBROUTINE SUB2B
COMMON /ACOM/ J(200)
COMMON /BCOM/ I(100)
END
```

Along with this program are the following segmentation directives:

```
TREE
A(B,C)
C(D,E)
ENDTREE
SEGMENT=A
  MODULES=EXAMPLE
ENDSEG
SEGMENT=B
  MODULES=SUB1
ENDSEG
SEGMENT=C
  MODULES=SUB2
ENDSEG
SEGMENT=D
  MODULES=SUB2A
ENDSEG
SEGMENT=E
  MODULES=SUB2B
ENDSEG
```

Figure 10 shows the segmentation structure before movable block assignment.

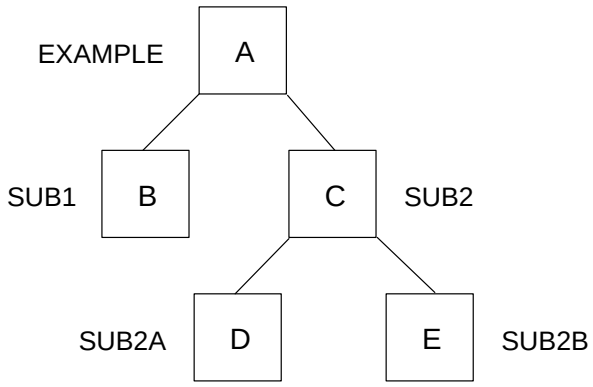


Figure 10. Segmentation structure before movable block assignment

If floating is enabled, the loader makes the following movable block assignments:

- `ASUB` is assigned to segment A because it is referenced in `EXAMPLE`.
- `BSUB` is assigned to segment A to move it to a common predecessor of segments B and C, enabling both `SUB1` and `SUB2` to reference `BSUB`.
- `ACOM` is assigned to segment A to accommodate references to it from `SUB1` in segment B and `SUB2B` in segment E.
- `BCOM` is assigned to segment C to accommodate references to it from `SUB2A` in segment D and `SUB2B` in segment E.

If automatic duplication is enabled, the loader makes the following movable block assignments:

- `ASUB` is assigned to segment A because it is referenced in `EXAMPLE`. It is not duplicated in segment B, because `ASUB` is present in predecessor segment A.
- `BSUB` is duplicated in segments B and C.
- `ACOM` is duplicated in segments B and E.
- `BCOM` is duplicated in segments D and E.

Common block use
6.4

This subsection describes some restrictions that apply to common blocks in segmented programs.

Data load restrictions
6.4.1

Data loads from modules in segments other than the segment in which the common block resides are not processed. The loader issues warning messages for data loads from other segments and skips the data.

The dynamic common, blank common, and task common blocks cannot be data loaded.

Block data routines 6.4.2

The loader always loads modules in `BIN` files that are block data routines. If block data in `LIB` routines is to be loaded, it must be referenced by a previously loaded program (using an `EXTERNAL` statement in Fortran) or by the loader's `MODULES` directive.

If you have a subroutine (not block data) that is never called but contains data loads, you can use the `MODULES` and `FORCE` directives to ensure that it is loaded.

Referencing data in common blocks 6.4.3

Data in a common block can be referenced by any module in either the same or a predecessor segment.

Caution: Referencing a common block that is in a successor segment is not recommended, because it is not guaranteed that the successor segment is memory resident at the time of the reference. This can cause unpredictable and incorrect program results.