Configuration Directives [11]

This section describes the directives you use for defining the configuration you want for the SEGLDR environment. These directives are not used in day-to-day activities.

Specifying default directory search lists

11.1

The DEFDIR directive allows you to specify default directory search lists. The LIBDIR directive allows you to add directory names to user directory search lists.

DEFDIR *directive*

11.1.1

The DEFDIR directive specifies default directory search lists. You can specify separate directory search lists for different machine characteristics.

The loader uses the default search list to find files specified on the -l and -j command-line options, and on the LBIN, LLIB, LINCLUDE, and DEFLIB directives. To find the specified files, the loader searches the directories listed in the user directory search list (specified with the -L command-line option or with the LIBDIR directive). If no user search list has been specified, or if the file is not found in any of the user directories, the loader searches the appropriate default directory search list.

Normally, the DEFDIR directive should be used in the default directive files def_seg and def_ld to establish the default search lists for all targeted machines.

Format:

```
DEFDIR[(chars)]=dirname1[, dirname2, ...]
```

chars Specifies a set of machine characteristics, including the primary machine name, logical name, and numeric characteristics. See the target(1) command for information on the characteristics that can be specified in *chars*.

dirname Specifies a UNICOS file system directory name.

When a set of machine characteristics is specified on a DEFDIR directive, the characteristics are associated with the list of search directories to create a *targeted* search list. If no characteristics are specified, the DEFDIR directive creates an *untargeted* search list. You can specify up to 10 DEFDIR directives, each with a different set of characteristics. DEFDIR directives are not cumulative. If more than one DEFDIR directive with the same characteristics has been specified, the directories specified on the latter directive replace those specified on the former. If more than one untargeted search list is specified, the latter directive replaces the former.

The loader determines the target environment of a program from the TARGET environment variable (see subsection 2.3.6, page 14, for more information on the TARGET variable), or, if TARGET is not set, from the main routine of the program. The loader scans the DEFDIR targeted search lists in the order specified. If a set of DEFDIR machine characteristics does not conflict with the characteristics of the target environment, the associated search list is used as the default search list for the program. If none of the DEFDIR characteristics sets matches the target environment, or if no targeted search lists have been specified, the untargeted search list is used.

Initially, DEFDIR specifies the /lib and /usr/lib directories in the untargeted search list and does not specify any directories in the targeted search list.

Example:

```
defdir(cray-ymp)=/lib/xlib,/usr/lib/xlib
defdir=/lib,/usr/lib,/usr/local/lib
```

When the target environment of a program is `cray-ymp`, the `/lib/xlib` and `/usr/lib/xlib` directories are searched. When any other target environment is used, the `/lib`, `/usr/lib`, and `/usr/local/lib` directories are searched.

Command-line equivalent: none

`LIBDIR` ***directive***
11.1.2

The `LIBDIR` directive adds directory names to the loader's user directory search list, which is used to find files specified on the `-l` and `-j` command-line options, as well as files specified on the `LBIN`, `LLIB`, `LINCLUDE`, and `DEFLIB` directives. The loader first searches each directory in the user search list. If directories have not been specified, or if the file cannot be located in any of the specified search directories, the loader searches the default directory search list for the file. (See "`DEFDIR` directive," page 109, for information on the default directory search list.)

Format:

```
LIBDIR=dirname1[,dirname2,...]
```

dirname UNICOS file system directory name.

You may specify up to 20 directory names. If this directive continues beyond one line, end each continued line with a comma. Multiple `LIBDIR` directives are cumulative. Each directive adds directory names until the limit of 20 is reached.

Example:

```
LIBDIR=/mydir/lib,locallib
```

The loader adds `/mydir/lib` and `locallib` (relative to the current directory) to the list of user search directories.

Command-line equivalent: `-L` option

The executable program
11.2

The `OUTFORM` directive give you a measure of control over the executable program that the loader produces. You can tell the loader the type of output file to produce.

`OUTFORM` *directive*
11.2.1

The `OUTFORM` directive specifies the type of the output file of the loader. This directive essentially allows you to build a prelinked collection of files with a `.o` extension. Within this collection of files all internal references have been resolved. This feature helps reduce application link time.

Format:

```
OUTFORM=[ABS|REL]
```

`ABS` The output file will have all internal references resolved (default).

`REL` The output file will have internal references resolved at link time.

`ld` command-line equivalent: `-r` (the executable program will have the relative attribute).

It is assumed that the relocatable output will be invoked only with the `ld` command. If you invoke the relocatable output with the `segldr` command, be certain to include the `SYSTEM=STDALONE` directive.

Controlling entry points and execution
11.3

The START and CALLXFER directives let you control the point at which your program begins executing, and they also intercept definitions of entry points at load time.

START *directive*
11.3.1

The START directive specifies the entry point that receives control from the operating system when the program begins execution. For normal programs executing under the UNICOS operating system, the entry point is the system start-up routine. The default directives file specifies the correct entry point for your system. You should use the START directive only when building a special-purpose program.

Format:

```
START=epname
```

epname Name of entry point at which program execution begins.

CALLXFER *directive*
11.3.2

The CALLXFER directive specifies the entry-point name used by the system start-up routine to call your main program. The loader links references to the CALLXFER entry point to the transfer entry point defined by the XFER directive. The default directives file specifies the correct name for your system. You should use the CALLXFER directive only when building a special-purpose program.

Format:

```
CALLXFER=epname
```

epname Symbol name used by the system start-up routine to call the XFER entry point.

Miscellaneous global directives
11.4

The SYSTEM directive specifies under which operating system your program will execute. The INCFILE directive specifies the name of a previously built executable program. The ZSYMS directive controls whether the loader will include the special zzzzzz?? symbols in the load module.

SYSTEM *directive*
11.4.1

The SYSTEM directive selects the target operating system on which your program will execute. The default directives file specifies a SYSTEM value of UNICOS.

Format:

```
SYSTEM=keyword
```

UNICOS — Sets the target operating system to UNICOS. When SYSTEM=UNICOS is specified, the loader requires that the START and CALLXFER directives are specified, and enables heap and stack processing, enable task common block processing, and adds the _infoblk information block to your program (default).

STDALONE — Sets the target operating system to be undefined. The loader does not require any directive settings and does not perform any special processing. The STDALONE directive should be used only for special-purpose programs.

INCFILE *directive*
11.4.2

The INCFILE directive specifies the name of a previously-built executable program. The loader extracts the symbol information from the file specified with the INCFILE directive. The extracted symbol information is used to satisfy external references and to allocate common blocks for object modules loaded during this invocation of the loader. When used in conjunction with the ORG, SYSTEM=STDALONE, and other directives, a program fragment is built that can execute in the address space of the original program. The original program must do the following actions: call the loader to create the program fragment, provide the memory space, to read the program fragment into its address

space, and pass control to it. The executable output produced when `INCFILE` is used cannot be executed independently. The `INCFILE` directive should be used only for special-purpose programs.

Format:

```
INCFILE=file
```

file Name of a file containing a previously linked executable program.

`ZSYMS` *directive*
11.4.3

This directive controls whether the loader will include the special `zzzzzz??` symbols in the load module. The default is `OFF`.

```
ZSYMS=[ON|OFF]
```

`ON` Include the `zzzzzz??` symbols in the load module.

`OFF` Do not include the `zzzzzz??` symbols in the load module.

Command-line equivalent: none.

Zero address directives
11.5

Zero address directives specify a block that is to occupy address zero. When these directives are used the value zero is no longer a valid pointer value. The `ZEROCOM` directive specifies the name of the common block that is to be placed at the zero address of the data space if common blocks precede local blocks; otherwise all three directives and their corresponding assembly modules are to be provided. The `ZERODATA` directive specifies the name of the module that is to be placed at the zero address of the data space. The `ZEROTEXT` directive specifies the name of the module that is to be placed at the zero address of the text space.

ZEROCOM *directive*
11.5.1

The ZEROCOM directive specifies the name of the common block that is to be placed at the zero address of the data space (if the load order is COMMONS, MODULES; otherwise this directive has no effect). The named module must contain only one common data block. If the directive is not present, or if the named module is not found, no special processing for address 0 is done.

The last ZEROCOM directive encountered is the one used; the earlier ZEROCOM directives are ignored.

This directive should only be used in the default directives file.

Format:

```
ZEROCOM=blkname
```

blkname Name of the common block to be loaded.

ZERODATA *directive*
11.5.2

The ZERODATA directive specifies the name of the module that is to be placed at the zero address of the data space. The named module must contain only one local data block. If the directive is not present, or if the named module is not found, no special processing for address 0 is done.

The last ZERODATA directive encountered is the one processed; the earlier ZERODATA directives are ignored.

This directive should only be used in the default directives file.

Format:

```
ZERODATA=modname
```

modname Name of the module to be loaded.

ZEROTEXT *directive*
11.5.3

The ZEROTEXT directive specifies the name of the module that is to be placed at the zero address of the text space. The named module must contain only one local code block. If the directive is not present, or if the named module is not found, no special processing for address 0 is done.

The last ZEROTEXT directive encountered is the one processed; the earlier ZEROTEXT directives are ignored.

This directive should only be used in the default directives file.

Format:

```
ZEROTEXT=modname
```

modname Name of the module to be loaded.

Managing global heap memory
11.6

The DEFHEAP, DEFSTACK, and FREEHEAP directives let you control the size and location of the system-managed heap and stack. Memory space can be acquired from the heap by using the system heap routines. Under the UNICOS operating system, the heap is always present and resides after the longest segment branch of your program. Heap space is available to all segments of your program.

These directives should only be used in the default directives file.

DEFHEAP *directive*
11.6.1

The DEFHEAP directive allocates memory that the heap manager can manage dynamically. When you use DEFHEAP, the HEAP directive is not needed unless you want to change the default heap values.

The DEFHEAP directive is intended for use in the default directives file to establish a minimum heap size for all programs. See the “HEAP directive,” page 87.

Format:

```
DEFHEAP=[init][+inc]
```

init Initial number of decimal words available to the heap manager. If *init* is less than or equal to 128 words or is absent, a value defined when the system is installed is used.

inc Increment size, in decimal words, of a request to the operating system for additional memory if the heap overflows. A value of zero implies that heap overflow is prohibited. A value defined when the system is installed determines the default increment value.

DEFSTACK *directive*
11.6.2

The DEFSTACK directive allocates part of heap memory to a stack for use by re-entrant programs. When you use DEFSTACK, the HEAP directive is not needed unless you want to change the default heap values.

The DEFSTACK directive is intended for use in the default directives file to establish a minimum stack size for all programs. See the "STACK directive," page 88, for an outline of the steps the loader takes in determining a program's stack size.

Format:

```
DEFSTACK=[init][+inc]
```

init Initial size, in decimal words, of a stack. If *init* is less than or equal to 128 words or is absent, a value defined when the system is installed is used.

inc Size, in decimal words, of additional increments to a stack if the stack overflows. A value of zero (0) implies that stack overflow is prohibited. A value defined when the system is installed determines the default increment value.

FREEHEAP *directive*
11.6.3

The FREEHEAP directive specifies the minimum amount of free memory available in the heap after the initial stack allocation. The initial heap size will be the sum of the initial stack size and the value specified by this directive.

Format:

```
FREEHEAP=value
```

value The number of words of space to be left free in the heap after allocation of the stack.

The use of more than one of the STACK, HEAP, and FREEHEAP directives can easily result in an inconsistent specification. If this occurs, the maximum size heap is used.

Scanning Directives [12]

When the target machine is a CRAY EL98 or CRAY J90 system, the loader invokes a special scanner to detect and correct potential problems in the program. The problems result from specific instruction sequences that generate unexpected results when the program uses multitasking on a CRAY EL98 system or enables cache memory on a CRAY J90 system. The loader provides two directives that work in conjuction with the scanner.

SCANNER directive

12.1

The SCANNER directive lets you turn the scanner off or on. The default condition is on when the target system is a CRAY EL or CRAY J90 system. If you are targetting your program for one of these systems and do not want your program scanned, add the SCANNER=OFF directive to your load step. If the target is a CRAY J90 system, your program will execute with cache memory disabled. If the target is a CRAY EL98 system and performs multitasking, you may encounter unexpected results.

Format:

```
SCANNER = [ON | OFF]
```

SCANPAD directive

12.2

When processing a segmented program, the scanner will occasionally be unable to locate enough unused memory areas to apply the necessary corrections. Use the SCANPAD directive to add additional unused memory to your program.

Format:

```
SCANPAD = nnnnnn
```

nnnnnn Additional number of words, in decimal, to add to the program.