Invoking SEGLDR [2]

You can control the loader's operation with the invocation statement shown in this section, or with the directives explained in "General Directives," page 21. "Command options and loader directives," page 15, shows the correspondence between command-line options and the loader directives. "Directives processing order," page 14, describes the effects of using both command-line options and directives.

There are two ways to invoke the loader. The `segldr`(1) command provides a simple invocation method in which the loader handles many of the requirements of loading your program. The `ld`(1) command provides a traditional UNIX interface in which you must provide more information to the loader to load your program correctly. The `cc`(1) command uses the `ld` interface when invoking the loader, and the `cf77`(1) command uses the `segldr` interface. "Differences between `segldr` and `ld`," page 17, describes how the two invocation formats differ.

Generally, text in this reference manual refers to "`segldr`" whenever information pertains only to the `segldr` invocation. It uses "`ld`" whenever information pertains only to the `ld` invocation. "SEGLDR" or "the loader" refers to information pertaining to the loader in general.

`segldr(1)` command line

2.1

Execute the loader with the following command line. Options can be specified in any order, however the order can affect how the options are interpreted:

```
segldr [-A incfile] [-a] [-b value] [-e ename] [-f value] [-g] [-H hi[+he]]
  [-i dirfiles] [-j names] [-k] [-l names] [-m] [-n] [-o outfile] [-s] [-t]
  [-u unames] [-D dirstring] [-E] [-F] [-L ldirs] [-M arguments] [-N]
  [-O keyword] [-S si[+se]] [-V] [-Z] [-z] [objfiles] files
```

`-A` *incfile* — Specifies an existing executable file. `segldr` extracts the symbols from *incfile* and links the new object modules as a code fragment that will execute as part of the original program.

`-a` — Aligns all local code and data blocks on instruction buffer boundaries.

`-b` *value* — Adds *value* number of 1024-word blocks of memory at the end of the loaded program (BSS space).

`-e` *ename* — Indicates that program execution should begin at entry *ename*. Control is passed from the system startup routine to the *ename* entry point which, under most circumstances, is the user `main` routine. The `-e` option is equivalent to the `XFER` directive.

`-f` *value* — Fills all uninitialized words of the program with *value*, which may be one of the following:

- `zeros` — Sets uninitialized data words to `0` (default).
- `ones` — Sets uninitialized data words to `-1`.
- `indef` — Sets uninitialized data words to `O'0605054000000000000000`, which causes a floating-point error if used in a floating-point operation.
- `-indef` — Sets uninitialized data words to `O'1605054000000000000000`, which causes a floating-point error if used in a floating-point operation.

`indefa` Sets uninitialized data to the sum of a logical OR operation of `O'06050540000000000000` and the address of the word being preset. This value is the same as that of `indef`, except the address of the word referenced appears in the low-order bits of the value.

`-indefa` Sets uninitialized data to the sum of a logical OR operation of `O'10605054000000000000000` and the address of the word being preset. This value is the same as that of `-indef`, except the address of the word referenced appears in the low-order bits of the value.

A 16-bit octal value
Inserts a 16-bit, user-supplied octal value into each parcel of uninitialized data words. The *value* must be in the range `0<=`*value*`<=O'177777`.

`-g` Generates the debug symbol tables and appends them to the executable file. This option is enabled by default. See the `-s` option.

`-H` *hi*[+*he*] Assigns the initial heap values. The *hi* is the initial heap size; *he* is the heap expansion increment.

`-i` *dirfiles* Reads and processes the directives in each of the specified directive files. Separate file specifications with commas. If the file specification begins with a period (`.`) or a slash (`/`), the loader assumes that it is a complete path and uses it without modification. If a dash (`-`) is present as one of the file names, the loader reads the `stdin` file for directives; otherwise, the loader looks for the named file in the current directory.

`-j` *names* Reads and processes the directives in each of the specified directive files. Separate file specifications with commas. If the file specification begins with a period (`.`) or a slash (`/`), the loader assumes that it is a complete path and uses it without modification. Otherwise, the loader looks for the named file(s) in the `segdirs` subdirectory in each search path. See the `-L` option for the list of search directories.

`-k` Redirects all but summary-class error messages to the load map file. See the `-M` option.

`-l` *names* Lists library names. If a name begins with a period (`.`) or a slash (`/`), `segldr` assumes it is a complete path name and uses it without modification as the name of a library file. Otherwise, `segldr` checks for file `lib`*name*`.a` in the list of search directories and includes the first one found as a library file. The list is separated by commas. See the `-L` option for the list of search directories.

`-m` Generates an address-level load map and writes it to `stdout`. Equivalent to `-M ,address`.

`-n` Generates a shared text program on Cray PVP systems.

`-o` *outfile* Writes the executable program to *outfile*. If the `-o` option is not used, the executable program is written to the file specified by the `ABS` directive. If neither the `-o` option nor `ABS` is specified, the executable output is written to file `a.out`.

`-s` Inhibits the generation of debug symbol tables. Debug symbol tables are generated by default.

`-t` Executes in trial mode. Scans all object modules, checks errors, and generates load maps, but it does not produce an executable program.

`-u` *unames* Enters *unames* as undefined symbols. This is useful for loading from a library, since undefined symbols are needed to force loading of the desired routines.

`-D` *dirstrng* Specifies a character string composed of `segldr` directives. Any global `segldr` directive may be provided. Directives must be separated with semicolons. See "Directives processing order," page 14, for the order in which `segldr` processes directives.

`-E` Echoes to the load map file all directives processed. See the `-M` option.

`-F` Enables force mode. All modules from `bin` and object files are loaded, whether or not they are referenced.

`-L` *ldirs* Adds one or more directory names to the list of search directories. `segldr` uses the list of search directories to locate files specified with the `-l` and `-j` options, as well as the `LBIN`, `LLIB`, `LINCLUDE`, and `DEFLIB` directives. If the file cannot be located in any specified search directory, `segldr` looks in the directories specified in the default directory search list. See "`DEFDIR` directive," page 109, for more information on default directory search lists. You may specify up to 100 directory names.

`-M` *file*
`-M ,`*opts*
`-M` *file*`,`*opts* Selects optional load map file and type of map to produce. If a file name is present, the loader writes the load maps to that file in paginated 132-column format. If a file name is not provided, load maps are written to the `stdout` file in nonpaginated, 80-column format. If no load-map options are specified, a block map that is sorted by address is the default type. Load-map options (*opts*) are as follows (you may specify more than one, separated by commas):

`s` or `stat`	Lists only load statistics.
`a` or `address`	Sorts block map by address; the default map, if no *opt* is specified.

`al` or `alpha`	Sorts block map by name.
`b` or `brief`	Restricts block map to modules only from object files.
`c` or `cbxrf`	Lists common block cross-references.
`e` or `epxrf`	Lists entry-point cross-references.
`p` or `part`	Lists a combination of `address` and `alpha`.
`f` or `full`	Lists all load maps.

`-N` Inhibits inclusion of the default libraries in the load.

`-O` *keyword* Selects the memory allocation order, which may be as follows:

`tdb`	Allocates all code, followed by all initialized data, followed by all uninitialized data (text, data, BSS).
`ema`	Allocates code to maximize use of Cray PVP systems extended memory addressing.
`s`	Allocates code to create a shared-text program for Cray PVP systems.
`ss.ema`	Allocates code to create a split-segment program that maximizes use of Cray PVP systems extended memory addressing.
`ss.tdb`	Allocates code to create a split-segment program, where code is followed by initialized data, which is followed by uninitialized data (text, data, BSS).

`-S` *si*[+*se*] Assigns initial stack values. The *si* is the initial stack size; *se* is the initial stack expansion increment.

`-V` Indicates that the loader list its version line to the `stderr` file.

`-Z` Inhibits `segldr` from reading the default directives file `/lib/segdirs/def_seg`. The default directives file is required to configure programs correctly for execution under the UNICOS operating system. The `-Z` option should only be used by special-purpose programs.

`-z` Specifies an alternative default directives file. The alternative directives must configure the program correctly for execution under the UNICOS operating system.

objfiles Files containing object modules produced by the compilers or assembler and object module library files prepared by `ar`(1) or `bld`(1) can be specified. Specifying files on the command line has the same effect as specifying them in a `BIN` directive.

files Files to be loaded. They may contain any of the following:

- Sequential object modules produced by the compilers or assembler. Specifying an object file on the command line has the same effect as specifying it on a `BIN` directive.
- Object libraries produced by `ar`(1) or `bld`(1). Specifying a library on the command line has the same effect as naming it on a `BIN` directive.
- SEGLDR directives. If you enter a hyphen (-) instead of file names, SEGLDR will accept directives from `stdin`.

Load maps, if selected, are written to the `stdout` file by default (see the `-M` option). Error messages are written to the `stderr` file by default (see the `-k` option).

Any file named in the loader directives or command line may be described by a full file path name.

`ld(1)` command line

2.2

To invoke the loader with a command-line format and defaults similar to those of the traditional UNIX `ld`(1) command, you can use the `ld`(1) command.

You can specify options in any order, however the order may affect how the options are interpreted (see `-l` and `-L`). Options and file arguments may be intermixed on the command line.

```
ld [-D dirstring] [-e name] [-F] [-g] [-i] [-j names][-l names] [-L ldirs]
  [-m] [-n] [-o outfile] [-r] [-s] [-u unames] [-V] [-Z] [-z file] files
```

`-D` *dirstring* Specifies a character string composed of the loader directives separated with semicolons. See "Directives processing order," page 14, for the order in which the loader processes directives.

`-e` *name* Sets the program entry address to the value of symbol *name*. Control is passed from the system kernel to the *name* entry point which, under most circumstances, is the system startup routine. The `-e` option is equivalent to the `START` directive.

`-F` Enables default library processing. The standard system libraries are processed after any user-supplied libraries. Processing of the system libraries is disabled by default.

`-g` Generates the debug symbol tables and appends them to the executable program. This option is enabled by default. See the `-s` option.

`-i` Generates a shared-text program on Cray PVP systems. Equivalent to the `-n` option.

`-j` *names* Lists directives file names. The list is separated by commas. If a name begins with a period (`.`) or a slash (`/`), `ld` assumes it is a complete path name and uses it without modification. Otherwise, `ld` checks for a `segdirs`/*name* file in the list of search directories and uses the first one found. See the `-L` option for the list of search directories.

`-l` *names* Lists library names. If a name begins with a period (`.`) or a slash (`/`), `ld` assumes it is a complete path name and uses it without modification as the name of a library file; otherwise, `ld` checks for file *libname*`.a` in the list of search directories and includes the first one found as a library file. The list is separated by commas. See the `-L` option for the list of search directories.

`-L` *ldirs* Adds one or more directory names to the list of search directories. `ld` uses the list of search directories to locate files specified with the `-l` and `-j` options, as well as the `LBIN`, `LLIB`, `LINCLUDE`, and `DEFLIB` directives. If the file cannot be located in any specified search directory, `ld` looks first in `/lib` and then in `/usr/lib`. You may specify up to 100 directory names.

`-m` Generates a load map of the executable program and writes it to the `stdout` file.

`-n` Generates a shared-text program on Cray PVP systems. Equivalent to the `-i` option.

`-o` *outfile* Writes the executable program to *outfile*. The default *outfile* name is `a.out`.

`-r` Produces relocatable output from prior linked `.o` files. The output is suitable for use by another invocation of `ld`. It is equivalent to using the following directives:

```
OUTFORM=REL
USX=NOTE
SYSTEM=STDALONE
ZSYMS=OFF
```

`-s` Inhibits generation of debug symbol tables. Debug symbol tables are generated by default.

`-u` *unames* Enters *unames* as undefined symbols. This is useful for loading from a library, because undefined symbols are needed to force loading of the desired routines.

`-V` Lists the `ld`(1) version line on `stderr`.

`-Z` Inhibits `ld` from reading the default directives file `/lib/segdirs/def_ld`. The default directives file is required to configure programs correctly for execution under the UNICOS operating system. The `-Z` option should be used only by special-purpose programs.

`-z` *file* Specifies an alternate default directives file. The alternate directives must configure the program correctly for execution under the UNICOS operating system.

files Files to be loaded. They may contain any of the following items:

- Sequential object modules produced by the compilers or assembler. Specifying an object file on the command line has the same effect as specifying it on a `BIN` directive.
- Object libraries produced by `ar`(1) or `bld`(1). Specifying a library on the command line has the same effect as naming it on a `LIB` directive.
- `ld` directives

UNICOS environment variable processing
2.3

Seven environment variables affect the execution of the loader: `LDDIR`, `LPP`, `MSG_FORMAT`, `NLSPATH`, `SEGDIR`, `TARGET`, and `TMPDIR`.

`LDDIR` *variable*
2.3.1

The `LDDIR` variable lets you specify `ld` directives or files of directives that are included automatically each time that you use `ld`. Thus you can set up your own defaults, tailored to the way you use `ld`. `LDDIR` is recognized only when `ld` is invoked.

Set the `LDDIR` variable by using the following format:

string;*string*;*string*;...

Each *string* is either a `ld` directive or the name of a file containing `ld` directives. See "Directives processing order," page 14, for a discussion of the order in which directives are processed.

`LPP` *variable*

2.3.2

If `LPP` is defined, the loader uses the value of the variable as the number of lines to print on each page for listing output. The `LPP` value must be between 15 and 999. If `LPP` is not present, the default is 57 lines per page.

`MSG_FORMAT` *variable*

2.3.3

The `MSG_FORMAT` variable describes a printing format similar to the C library routine, `printf`, that can be used to alter the layout of error messages produced by the loader. See the `explain`(1) command for a complete description of `MSG_FORMAT`.

`NLSPATH` *variable*

2.3.4

The `NLSPATH` variable specifies a list of alternative directories that the loader should search to locate its error message catalog. The `NLSPATH` environment variable is used to select alternative catalogs for debugging purposes, or when different versions of the loader are operating on the same system. It is not needed for normal operation.

`SEGDIR` *variable*

2.3.5

The `SEGDIR` variable lets you specify `segldr` directives or files of directives that are included automatically each time that you use `segldr`. Thus you can set up your own defaults, tailored to the way you use `segldr`. `SEGDIR` is recognized only when `segldr` is invoked.

Set the `SEGDIR` variable by using the following format:

string;*string*;*string*;...

Each *string* is either a `segldr` directive or the name of a file containing `segldr` directives. See "Directives processing order," page 14, for a discussion of the order in which directives are processed.

`TARGET` *variable*
2.3.6

The `TARGET` variable specifies the machine characteristics of the system on which the program will execute. The loader generates the program so that it operates correctly on that system. If the `TARGET` variable has not been specified, the program is adapted to the host system. See `target`(1) for more information.

`TMPDIR` *variable*
2.3.7

The `TMPDIR` variable specifies the directory that the loader uses for its temporary file. If the variable is not specified or is not correct, a site-specific system default is used.

Directives processing order
2.4

The `segldr` and `ld` invocations of the loader process directives and command-line options in a similar manner. This subsection describes the order of processing and how directives interact with command-line options. Directives and command-line options are processed in the following order:

1. The loader first reads and processes the default directives file, which provides the loader with the basic information needed to construct a valid UNICOS executable program. The contents of the file may be tailored to meet the needs of each site. The `-Z` command-line option can be used to inhibit default processing of this file. The `-z` command-line option may be used to provide an alternative default directives file. The default directives files are:

   ```
   segldr     /lib/segdirs/def_seg
   ld         /lib/segdirs/def_ld
   ```

2. After the default directives file is processed, `segldr` interrogates the `SEGDIR` environment variable; `ld` interrogates the `LDDIR` environment variable. Directives and directives file names may be specified in the environment variable. The directives and file contents are processed in the order encountered.

3. The command line is processed next. Each command-line option has an equivalent directive that performs the same function. Table 1 describes the correspondence between `segldr` command-line options and directives. Table 2, page 16, provides the correspondence between `ld` command-line options and directives. Command-line options and

arguments are processed in the order encountered, with one exception: directives files specified on the command line, either as arguments or with the `segldr -i` option, are processed after all other command-line options.

Because segmentation directives must be evaluated after global directives, they can be specified only in the user directives files named on the command line. User directives files can be specified either as command-line arguments or with the `-i` command-line option.

Command options and loader directives

2.5

Table 1 and Table 2, page 16, show the correspondence between `segldr` and `ld` command-line options and loader directives.

Table 1. Directives equivalents for `segldr` command-line options

Command-line option	Directive
`a`	`align=modules`
`b` *value*	`addbss=`*value*
`e` *entry*	`xfer=`*entry*
`f` *value*	`preset=`*value*
`g`	`symbols=on`
`i` *file*	`include=`*file*
`j` *name*	`linclude=segdirs/`*name*
`k`	no directive equivalent
`l` *name*	`lib=`*libname*`.a`
`l` */filename*	`lib=`*/filename*
`m`	`map=address`
`n`	`order=shared`
`o` *file*	`abs=`*file*
`s`	`symbols=off`
`t`	`trial`
`u` *name*	`unsat=`*name*

Table 1. Directives equivalents for `segldr` command-line options
(continued)

Command-line option	Directive
`z`	no directive equivalent
`A` *file*	`incfile=`*file*
`D` *directive*	*directive*
`E`	`echo=on`
`F`	`force=on`
`H` *values*	`heap=`*values*
`L` *directory*	`libdir=`*directory*
`M` `,`*keywords*	`map=`*keywords*
`N`	`nodeflib`
`O` *keyword*	`order=`*keyword*
`S` *values*	`stack=`*values*
`V`	No directive equivalent
`Z`	No directive equivalent
`.o` object file argument	`bin=`*file*
`.a` library file argument	`bin=`*file*
Directives file argument	`include=`*file*

Table 2. Directives equivalents for `ld` command-line options

Command-line option	Directive
`e` *entry*	`start=`*entry*
`g`	`symbols=on`
`i`	`order=shared`
`j` *name*	`linclude=segdirs/`*name*
`l` *name*	`llib=`*libname*`.a`
`l` */filename*	`lib=`*/filename*
`m`	`map=address`

Table 2. Directives equivalents for `ld` command-line options
(continued)

Command-line option	Directive
`n`	`order=shared`
`o` *file*	`abs=`*file*
`r`	`outform=rel;usx=note; system=stdalone`
`s`	`symbols=off`
`u` *name*	`unsat=`*name*
`z`	No directive equivalent
`D` *directive*	*directive*
`F`	`include=ld_Flib`
`L` *directory*	`libdir=`*directory*
`V`	No directive equivalent
`Z`	No directive equivalent
`.o` object file argument	`bin=`*file*
`.a` library file argument	`lib=`*file*
Directives file argument	`include=`*file*

Differences between `segldr` and `ld`

2.6

In addition to differences in command-line invocation formats, `segldr` and `ld` vary in other ways. Table 3 summarizes these differences.

Table 3. `segldr` and `ld` differences

Feature	`segldr`	`ld`
Default directives file	`/lib/segdirs/def_seg`	`/lib/segdirs/def_ld`
Environment variable processing	`SEGDIR`	`LDDIR`

Table 3. segldr and ld differences
(continued)

Feature	segldr	ld
Object file processing	All object file names are included as bin files.	All .o files (sequential object files) are included as bin files. All .a files (library object files) are included as lib files.
Default setting of DUPENTRY directive	DUPENTRY=CAUTION:CAUTION:NOTE	DUPENTRY=CAUTION:NOTE:NOTE. Because of the different dupentry setting, and the practice of including library object files as lib files, ld issues fewer diagnostic messages about duplicated entry point names than segldr.
Default setting of DUPORDER directive	DUPORDER=OFF The first definition of an entry point is chosen, regardless of the definition's location.	DUPORDER=ON. An ordered search algorithm is used. The entry point that ld chooses depends on the order of definitions and references. See "DUPORDER directive," page 32, for more information.
Default system libraries	A list of default libraries is included. Most common system routines are included in these libraries.	No default libraries are included. You must specify all libraries required by your program.
Default setting for USX directive	USX=CAUTION. A program that contains unsatisfied external references is still executable and segldr exits normally. Calls to unsatisfied references are intercepted when the program is run.	USX=WARNING. A program that contains unsatisfied external references is not executable and ld exits with a nonzero error status.
Default setting for FORCE directive	FORCE=OFF. Modules in bin files are included in the executable program only if they are referenced, contain a main program, or initialize global data.	FORCE=ON. All modules encountered in bin files are included in the executable program, whether or not the modules are referenced.

Default directives files

2.7

`segldr` and `ld` begin processing by reading a file of directives. The `segldr` default directives file is `/lib/segdirs/def_seg`; the `ld` default directives file is `/lib/segdirs/def_ld`. The defaults directives files provide the basic information needed for `segldr` or `ld` to create an executable UNICOS program. In addition, directives can be added to the default files to meet the loader operations needs of a particular site. Several common options for modifying the default directives files include the following:

- Adding or deleting default libraries
- Adding or deleting search directives
- Changing message severities

Defaults for directives are discussed throughout this manual. These settings reflect the values as released by Cray Research. The default values you find at your site may differ.

You can suppress default directives file processing by including the `-Z` option on your `segldr` or `ld` command line. You can substitute a different directives file by using the `-z` option. If you choose to substitute the directives file, you must provide the necessary directives to cause the loader to correctly build your program.