NAME

ld – link-editor for object files

SYNOPSIS

ld [**-32** | **-64**] [**-B** direct | nodirect] [**-B** dynamic | static]
[**-B** eliminate[=*mode*] | local[=*mode*]] [**-B** reduce] [**-c** *name*]
[**-C**] [**-D** [**!**]*token1*,[**!**]*token2*,...] [**-e** *epsym*]
[**-f** *name* | **-F** *name*] [**-G**] [**-h** *name*] [**-i**] [**-l** *x*] [**-L** *path*]
[**-m**] [**-M** *mapfile*] [**-o** *outfile*] [**-p** *auditlib*] [**-P** *auditlib*]
[**-Q** y | n] [**-r**] [**-R** *path*] [**-s**] [**-S** *supportlib*] [**-u** *symname*]
[**-V**] [**-z** allextract | defaultextract | weakextract]
[**-z** ancillary[=*outfile*]] [**-z** assert-deflib[=*libname*]]
[**-z** compress-class=[**!**]*class1*,[**!**]*class2*,...]
[**-z** compress-sections[=*cmp-type*[,cmp-opt...]]]
[**-z** ctf[=*opt1*,*opt2*,...] [**-z** deferred | nodeferred]
[**-z** defs | nodefs] [**-z** direct | nodirect] [**-z** discap]
[**-z** discard-unused=*item1*,*item2*,...] [**-z** endfiltee]
[**-z** fatal-warnings | nofatal-warnings] [**-z** finiarray=*function*]
[**-z** globalaudit] [**-z** guidance[=*item1*,*item2*,...]] [**-z** help]
[**-z** ignore | record] [**-z** initarray=*function*] [**-z** initfirst]
[**-z** interpose] [**-z** lazyload | nolazyload]
[**-z** ld32=*arg1*,*arg2*,...] [**-z** ld64=*arg1*,*arg2*,...]
[**-z** loadfltr] [**-z** mapfile-add=*name*] [**-z** nodelete]
[**-z** nodlopen] [**-z** parent=*object*] [**-z** preinitarray=*function*]
[**-z** rescan-now] [**-z** rescan-start ... **-z** rescan-end]
[**-z** strip-class=[**!**]*class1*,[**!**]*class2*,...] [**-z** stub]
[**-z** sx=*extension*[=*mode*]] [**-z** symbolcap] [**-z** sysroot=*dir*]
[**-z** target=sparc | x86] [**-z** text | textwarn | textoff]
[**-z** type=*object-type*] [**-z** verbose] [**-z** wrap=*symbol*]
file ...

DESCRIPTION

The link-editor, **ld**, combines relocatable object files by resolving symbol references to symbol definitions, together with performing relocations. In all cases, the default output of the link-editor is left in the file **a.out**. See NOTES.

The link-editor has a large number of options. Those options that are relevant to modern programming practices are defined in the SYNOPSIS, and described in the sections that follow. Other options are less commonly used, and are described under the SECONDARY OPTIONS section.

The link-editor takes a variety of input files, typically generated from compilers, assemblers, or previous invocations of the link-editor. The link-editor concatenates and interprets the data within these input files to form an output file. The output file that is produced is one of the following basic types.

- **Dynamic Executable** - A concatenation of relocatable objects that can be executed by **exec**(2). A dynamic executable is loaded at a fixed virtual address space and is executed under control of the runtime linker, **ld.so.1**(1), to produce a runtime process. Dynamic executables typically have one or more dependencies in the form of shared objects.

  A dynamic executable is created when the **-z type=exec** option is used, or is the default when no other options that control the output file type are provided.

- **Kernel Module** - A special case of a relocatable object, containing dynamic linking information, that can be loaded by the operating system kernel. Kernel modules can have dependencies on other kernel modules.

A kernel module is created when the **-z type=kmod** option is used.

- **Position-independent Executable (PIE)** - A position independent form of a dynamic executable that can be executed by **exec**(2). A position-independent executable is loaded at an arbitrary virtual address space and is executed under control of the runtime linker, **ld.so.1**(1), to produce a runtime process. Position-independent executables typically have one or more dependencies in the form of shared objects.

  A position-independent executable is created when the **-z type=pie** option is used.

- **Relocatable Object** - A concatenation of relocatable objects that can be used in subsequent link-edit phases.

  A relocatable object is created when the **-z type=reloc** option, or **-r** option are used.

- **Shared Object** - A concatenation of relocatable objects that provide services for other dynamic objects. A shared object is loaded at an arbitrary virtual address space under the control of the runtime linker, **ld.so.1**(1) and is bound to executables, or other shared objects, at runtime. Shared objects can have dependencies on other shared objects.

  A shared object is created when the **-z type=shared** option, or **-G** option are used.

- **Static Executable** - A concatenation of relocatable objects that can be executed directly by **exec**(2), without assistance from the runtime linker, **ld.so.1**(1).

  A static executable is created when the **-z type=static** option or **-a** option are used. However, modern versions of the Oracle Solaris operating system do not support the creation of static executables. See **Static Executables**.

All discussion in this section assumes the construction of dynamic objects, relocatable objects, or kernel modules.

The dynamic linking environment tightly couples the work of the link-editor and the runtime linker, **ld.so.1**(1). These utilities are extensively documented in *Oracle Solaris 11.4 Linkers and Libraries Guide*.

Dynamic executables and position-independent executables are collectively referred to as executables. These executables, together with shared objects, are collectively referred to as dynamic objects.

Shared objects and relocatable archives, created with **ar**(1), are collectively referred to as libraries. If any argument is a library, the link-editor by default, searches the library exactly once at the point the library is encountered on the command line.

A shared object consists of an indivisible, whole unit that has been generated by a previous link-edit of one or more input files. When the link-editor processes a shared object, the entire contents of the shared object become a logical part of the resulting output file image. The shared object is not physically copied during the link-edit as its actual inclusion is deferred until process execution. This logical inclusion means that all symbol entries defined in the shared object are made available to the link-editing process. See Chapter 4, *Shared Objects* in *Oracle Solaris 11.4 Linkers and Libraries Guide*.

For an archive library, the link-editor, by default, loads only those archive members that define an unresolved external reference. The link-editor searches the symbol table of the archive library sequentially to resolve external references that can be satisfied by archive members. This search is repeated until no external references can be resolved by the archive. Thus, the order of members in the library is functionally unimportant, unless multiple archive members exist that define the same external symbol. See also the **-z allextract** option. Archive libraries that have interdependencies can require multiple command line definitions, or the use of one of the **-z rescan** options. See *Archive Processing* in *Oracle Solaris 11.4 Linkers and Libraries Guide*.

The link-editor is a **cross** link-editor, able to link 32-bit objects or 64-bit objects, for SPARC or x86 targets. The link-editor uses the **ELF** class and machine type of the first relocatable object on the command line to govern the mode in which to operate. The mixing of 32-bit objects and 64-bit objects is not permitted. Similarly, only objects of a single machine type are allowed. See the **-32**, **-64** and **-z target** options.

To provide for efficient arbitrary virtual address space mapping, position-independent executables and shared objects should be created from position-independent code (**PIC**). See the **-z text** option.

Static Executables

The creation of static executables has been discouraged for many releases. In fact, 64-bit system archive libraries have never been provided with Solaris. Because a static executable is built against system archive libraries, the executable contains system implementation details. This self-containment has a number of drawbacks.

- The executable is immune to the benefits of system patches delivered as shared objects. The executable therefore, must be rebuilt to take advantage of many system improvements.
- The ability of the executable to run on future releases can be compromised.
- The duplication of system implementation details negatively affects system performance.

Starting with Oracle Solaris 10, 32-bit system archive libraries are no longer provided. Without these libraries, specifically **libc.a**, the creation of static executables is no longer achievable without access to undocumented and changeable system details. However, the capability of the link-editor to process static linking options, and the processing of archive libraries, remains unchanged.

Option Processing

Typically, link edits are completely specified using command line options. In addition, a variety of environment variables are provided to augment command line processing. These variables provide for supplying options that might clash with compiler options. These variables also provide for overriding, or unsetting, the command line options that are embedded in scripts and build environments.

Any inconsistencies between command line options result in a fatal error condition. Any inconsistencies that involve an option provided by an environment variable result in a warning, and the first option taking precedence.

Initial options are processed from the **LD_OPTIONS** environment variable, followed by the command line, followed by the **LD_UNSET** environment variable.

From these three components, the initial object type of the output file being created, is determined. This object type is then used to investigate any **LD_{object-type}_UNSET**, and **LD_{object-type}_OPTIONS** environment variables. These variables can remove, or add, options specific to the object type being built.

The *object-type* corresponds to the values accepted by the **-z type** option, in uppercase, and is one of **EXEC**, **KMOD**, **PIE**, **RELOC** or **SHARED**. For example, the **LD_EXEC_OPTIONS** environment variable is interpreted when the object type is a dynamic executable.

If an **LD_{object-type}_OPTIONS** exists, the variable is first searched to discover whether a **-z type** option is specified. This search provides a final chance to affect the object type of the output file being created.

The object type is now finalized, and the associated environment variables for this final object type are processed. Note that these variables can not change the object type or class.

Option processing can be summarized by the following order.

- From the **LD_OPTIONS** environment variable.
- From the command line.
- From the **LD_UNSET** environment variable.
- From the **LD_{object-type}_OPTIONS** environment variable - searched only for a **-z type** option.
- From the **LD_{object-type}_UNSET** environment variable.
- From the **LD_{object-type}_OPTIONS** environment variable.

The following example executes a build process where all objects created by the link-editor have guidance enabled. Any dynamic executables become position-independent executables, and have a number of security extensions enabled. Any shared objects are ensured to contain position-independent code and have all their dependencies defined.

```
$ LD_OPTIONS=-zguidance \
  LD_EXEC_OPTIONS=-ztype=pie \
  LD_PIE_OPTIONS=-zsx=aslr,nxheap,nxstack \
  LD_SHARED_OPTIONS="-ztext -zdefs"  <build process>
```

Any command line options that are inconsistent with this output object type result in a fatal error condition. Any inconsistent option provided by an environment variable results in a warning, and the option being ignored.

Any **UNSET** operation is accompanied with a warning notification. Any other option processing warnings can be promoted to a fatal error condition by setting the **-z fatal-warnings** option.

OPTIONS

The following options are supported.

-32 | -64
-m32 | -m64

Create a 32-bit, or 64-bit object.

By default, the class of the object being generated is determined from the first **ELF** object processed from the command line. If no objects are specified, the class is determined by the first object encountered within the first archive processed from the command line. If there are no objects or archives, the link-editor creates a 32-bit object.

The **-64** option is required to create a 64-bit object solely from a **mapfile**.

The **-32** or **-64** options can also be used in the rare case of linking entirely from an archive that contains a mixture of 32 and 64-bit objects. If the first object in the archive is not the class of the object that is required to be created, then the **-32** or **-64** option can be used to direct the link-editor.

-B direct | nodirect

These options govern direct binding. The **-B direct** option establishes direct binding information by recording the relationship between a symbol reference and the dependency that provides the symbol definition. In addition, direct binding information can be established between a symbol reference and an associated definition within the object being created. The runtime linker uses this information to search directly for a symbol in the associated object rather than to carry out a default symbol search.

Direct binding information can only be established to dependencies specified with the link-edit. Thus, the **-z defs** option should also be added. Objects that wish to interpose on symbols in a direct binding environment should identify themselves as interposers with the **-z interpose** option. The use of **-B direct** also enables **-z lazyload** for all dependencies.

The **-B nodirect** option prevents any direct binding to the interfaces offered by the object being created. The object being created can continue to directly bind to external interfaces by specifying the **-z direct** option. See Chapter 7, *Direct Bindings* in *Oracle Solaris 11.4 Linkers and Libraries Guide*.

-B dynamic | static

These options govern library inclusion. The **-B dynamic** option allows a **-l** option library search to expand to a shared object and an archive library name. This option is valid, and the default, in dynamic mode only. **-B static** restricts a **-l** option library search to expand to archive library names only. These options can be specified any number of times on the command line as toggles. If the **-B static** option is given, no shared objects are accepted until **-B dynamic** is seen. See the **-l** option.

-B eliminate[=*mode***] |** local[=*mode*]

Causes any symbols that default to global binding, to be eliminated from the symbol table, or reduced to local visibility. A global symbol that has a **STV_DEFAULT** visibility, can be reduced to a more restrictive visibility by the link-editor. A global symbol that has any other **STV_** visibility is considered to have an explicit visibility. An explicit visibility is honored by the link-editor, and can not be modified. See *Symbol Visibility* in *Oracle Solaris 11.4 Linkers and Libraries Guide*. A symbols visibility can be explicitly defined through compiler directives, or through **mapfiles** that define version or interface definitions.

Mapfile version and interface definitions can contain *auto-elimination* or *auto-reduction* directives. See *SYMBOL_SCOPE and SYMBOL_VERSION Directives* in *Oracle Solaris 11.4 Linkers and Libraries Guide*. These directives result in symbols that are not explicitly defined in a **mapfile**, or do not define an explicit visibility, to be eliminated or reduced to local, respectively. Besides any explicit symbol definitions, symbol elimination or reduction can also be affected by the type of object being produced, as described in the paragraphs that follow. The **-B eliminate** option requests the same

symbol elimination as the **mapfile** *auto-elimination* directive. The **-B local** option requests the same symbol reduction as the **mapfile** *auto-reduction* directive.

Either option can be qualified with a mode, **external**, or **noexternal**, to control the selection of eliminated or reduced global symbols. This fine-tuning is usually unnecessary, as the link-editor defaults to the appropriate mode for the type of object being produced, **external** for executables, and **noexternal** for shared objects.

When building a dynamic object, it can be necessary to ensure that some symbols remain global so that they can be referenced from external dependencies. This is particularly true for executables. When building an executable, relocatable objects are contributed by the compilation environment that provide for runtime process initialization. These relocatable objects can contain global symbols that are referenced from system dependencies. These symbols should remain global regardless of any auto-elimination or auto-reduction symbol techniques, so as not to compromise runtime execution.

Defining the **mode** as **external** results in the analysis of any external dependencies to determine if any symbol reference from the dependency might bind to a symbol definition within the object being built. Any global symbol that satisfies such a binding is not eliminated or reduced to local. This mode is the default when producing an executable.

Defining the **mode** as **noexternal** circumvents the analysis of any external dependencies, and results in the reduction of all symbols that are not explicitly defined in a **mapfile**, or do not define an explicit visibility. This mode is the default when producing a shared object.

See also the **-B reduce** option.

-B reduce

When generating a relocatable object, causes the reduction of symbolic information defined by any symbol visibility attribute, or through **mapfiles** that define version or interface definitions. By default, when a relocatable object is generated, visibility attributes, version definitions, or interface definitions are only recorded in the output image. Visibility attributes, or **mapfile** version or interface definitions, are always applied to any symbolic information when creating a dynamic object or kernel module.

-c *name*

Records the configuration file *name* for use at runtime. Configuration files can be employed to alter default search paths, provide a directory cache, together with providing alternative object dependencies. See **crle**(1). This option is only available when creating a dynamic executable or position-independent executable.

-C

Demangles C++ symbol names displayed in diagnostic messages.

-D [!]*token1*,[!]*token2*,...

Prints debugging information as specified by each *token*. The special token **help** indicates the full list of tokens available. See *Debugging Aids* in *Oracle Solaris 11.4 Linkers and Libraries Guide*. If the special token **help** is specified by itself, output goes to the standard output. If any other tokens are specified, output goes to the standard error.

-e *epsym*
--entry *epsym*

Sets the entry point address for the output file to be the symbol *epsym*.

-f *name*
--auxiliary *name*

Used only when building a shared object. Specifies that the symbol table of the shared object is used as an auxiliary filter on the symbol table of the shared object specified by *name*. Multiple instances of this option are allowed. This option can not be combined with the **-F** option. See *Generating Auxiliary Filters* in *Oracle Solaris 11.4 Linkers and Libraries Guide*.

-F *name*
--filter *name*

Used only when building a shared object. Specifies that the symbol table of the shared object is used as a filter on the symbol table of the shared object specified by *name*. Multiple instances of this option are allowed. This option can not be combined with the **-f** option. See *Generating Standard Filters* in *Oracle Solaris 11.4 Linkers and Libraries Guide*.

-G
-shared

In dynamic mode only, produces a shared object. Undefined symbols are allowed. See also the **-z type=shared** option. See Chapter 4, *Shared Objects* in *Oracle Solaris 11.4 Linkers and Libraries Guide*.

-h *name*
-soname *name*

In dynamic mode only, when building a shared object or kernel module, records *name* in the object's dynamic section. *name* is recorded in any dynamic objects that are linked with this object rather than the object's file system name. Accordingly, *name* is used by the runtime linker or kernel runtime linker respectively, as the name of the shared object to search for at runtime. See *Recording a Shared Object Name* in *Oracle Solaris 11.4 Linkers and Libraries Guide*.

-i

Ignores **LD_LIBRARY_PATH**. This option is useful when an **LD_LIBRARY_PATH** setting is in effect to influence the runtime library search, which would interfere with the link-editing being performed.

-l *x*
--library *x*

When building a dynamic object, searches a library **lib*x*.so** or **lib*x*.a**, the conventional names for shared object and relocatable archives, respectively. In dynamic mode, unless the **-B static** option is in effect, the link-editor searches each directory specified in the library search path for a **lib*x*.so** or **lib*x*.a** file. The directory search stops at the first directory containing either. The link-editor chooses the file ending in **.so** if **-l*x*** expands to two files with names of the form **lib*x*.so** and **lib*x*.a**. If no **lib*x*.so** is found, then the link-editor accepts **lib*x*.a**. In static mode, or when the **-B static** option is in effect, the link-editor selects only the file ending in **.a**.

When building a kernel module, searches for a dependency kernel module *x*. The dependency can be specified as a single name *x* or as a slash delimited *interface/module* pair.

The link-editor searches a library when the library is encountered, so the placement of **-l** is significant. See *Linking With Additional Libraries* in *Oracle Solaris 11.4 Linkers and Libraries Guide*.

-L *path*
--library-path *path*

Adds *path* to the library search directories. The link-editor searches for libraries first in any directories specified by the **-L** options and then in the standard directories. This option is useful only if the option precedes the **-l** options to which the **-L** option applies. See *Directories Searched by the Link-Editor* in *Oracle Solaris 11.4 Linkers and Libraries Guide*.

The environment variable **LD_LIBRARY_PATH** can be used to supplement the library search path, however the **-L** option is recommended, as the environment variable is also interpreted by the runtime environment. See **LD_LIBRARY_PATH** under ENVIRONMENT VARIABLES.

If *path* starts with the prefix **$SYSROOT/**, then the prefix is replaced by the **sysroot** directory specified by the **-z sysroot** option, or with **/** if **-z sysroot** is not used. For compatibility with **GNU** link-editors, the prefix **=/** is also accepted, and has the same meaning as a **$SYSROOT/** prefix. The use of a **sysroot** prefix can allow a single **-L** definition to work for both native and cross linking applications.

-m

Produces a memory map or listing of the input/output sections, together with any non-fatal multiply-defined symbols, on the standard output.

-M *mapfile*

Reads *mapfile* as a text file of directives to the link-editor. This option can be specified multiple times. If *mapfile* is a directory, then all regular files, as defined by **stat**(2), within the directory are processed. See Chapter 10, *Mapfiles in the Link-Editor* in *Oracle Solaris 11.4 Linkers and Libraries Guide*. Example **mapfiles** are provided in **/usr/lib/ld**. See also FILES.

-o *outfile*
--output *outfile*

Produces an output object file that is named *outfile*. The name of the default object file is **a.out**.

-p *auditlib*

Identifies an audit library, *auditlib*. This audit library is used to audit the object being created at runtime. A shared object identified as requiring auditing with the **-p** option, has this requirement inherited by any object that specifies the shared object as a dependency. See the **-P** option. See also *Runtime Linker Auditing Interface* in *Oracle Solaris 11.4 Linkers and Libraries Guide*.

-P *auditlib*

Identifies an audit library, *auditlib*. This audit library is used to audit the dependencies of the object being created at runtime. Dependency auditing can also be inherited from dependencies that are identified as requiring auditing. See the **-p** option, and the **-z globalaudit** option. See also *Runtime Linker Auditing Interface* in *Oracle Solaris 11.4 Linkers and Libraries Guide*.

-Q y | n

Under **-Q y**, an **ident** string is added to the **.comment** section of the output file. This string identifies the version of the link-editor used to create the file. This results in multiple link-editor **idents** when there have been multiple linking steps, such as when using the link-editor **-z type=reloc** or **-r** options. This identification is identical with the default action of the **cc** command. The **-Q n** option suppresses version identification. **.comment** sections can be manipulated by the **mcs**(1) utility.

-r
--relocatable

Combines relocatable objects to produce one relocatable object file. The link-editor does not complain about unresolved references. This option cannot be used with the **-a** option. See also the **-z type=reloc** option.

-R *path*
-rpath *path*

A colon-separated list of directories used to specify library search directories to the runtime linker. If present and not NULL, the path is recorded in the output object file and used by the runtime linker. Multiple instances of this option are concatenated together with each *path* separated by a colon. See *Directories Searched by the Runtime Linker* in *Oracle Solaris 11.4 Linkers and Libraries Guide*.

The use of a **runpath** within an associated object is preferable to setting global search paths such as through the **LD_LIBRARY_PATH** environment variable. Only the **runpaths** that are necessary to find the objects dependencies should be recorded. **ldd**(1) can also be used to discover unused **runpaths** in dynamic objects, when used with the **-U** option.

Various tokens can also be supplied with a **runpath** that provide a flexible means of identifying system capabilities or an objects location. See Chapter 12, *Establishing Dependencies with Dynamic String Tokens* in *Oracle Solaris 11.4 Linkers and Libraries Guide*. The **$ORIGIN** token is especially useful in allowing dynamic objects to be relocated to different locations in the file system.

-s
--strip-all

Strip any symbolic information from the output file. These options are equivalent to using the **-z stripclass** option with the **debug** and **symbol** class identifiers. See also the **-z redlocsym** and **-z noldynsym** options.

-S *supportlib*

The shared object *supportlib* is loaded with the link-editor and given information regarding the linking process. Shared objects that are defined by using the **-S** option can also be supplied using the **SGS_SUPPORT** environment variable. See *Link-Editor Support Interface* in *Oracle Solaris 11.4 Linkers and Libraries Guide*.

-u *symname*
--undefined *symname*

Enters *symname* as an undefined symbol in the symbol table. This option is useful for loading entirely from an archive library. In this instance, an unresolved reference is needed to force the loading of the first routine. The placement of this option on the command line is significant. This option must be placed before the library that defines the symbol. See *Defining Additional Symbols with the -u option* in *Oracle Solaris 11.4 Linkers and Libraries Guide*.

-V
--version

Outputs a message giving information about the version of the link-editor being used.

-z allextract | defaultextract | weakextract
--whole-archive | --no-whole-archive

Alters the extraction criteria of objects from any archives that follow. By default, archive members are extracted to satisfy undefined references and to promote tentative definitions with data definitions. Weak symbol references do not trigger extraction. Under the **-z allextract** or **--whole-archive** options, all archive members are extracted from the archive. Under **-z weakextract**, weak references trigger archive extraction. The **-z defaultextract** or **--no-whole-archive** options provide a means of returning to the default following use of the former extract options. See *Archive Processing* in *Oracle Solaris 11.4 Linkers and Libraries Guide*.

-z ancillary[=*outfile*]

Specifies an ancillary output file, which will receive any non-allocable sections that would normally be added to the output object. Non-allocable sections are not required at runtime, and are primarily for use by debuggers and other observability tools. If *outfile* is present, the ancillary file is created with the given name. If *outfile* is not present, the ancillary file is given the same name as the primary output file with the addition of a **.anc** suffix. See *Ancillary Objects* in *Oracle Solaris 11.4 Linkers and Libraries Guide*.

The **-z ancillary** option is quietly ignored if used in conjunction with the **-z stub** option. The **-z ancillary** option is also ignored if specified without *outfile*, and the output file specified via the **-o** option is

a device special file such as **/dev/null**.

-z assert-deflib[=*libname*]

Enables warning messages for libraries specified with the **-l** command line option that are found by examining the default search paths provided by the link-editor. If a *libname* value is provided, the default library warning feature is enabled, and the specified library is added to a list of libraries for which no warnings will be issued. Multiple **-z assert-deflib** options can be specified in order to specify multiple libraries for which warnings should not be issued.

The *libname* value should be the name of the library file, as found by the link-editor, without any path components. For example, the following enables default library warnings, and excludes the standard C library.

$ **ld ... -z assert-deflib=libc.so ...**

-z assert-deflib is a specialized option, primarily of interest in build environments where multiple objects with the same name exist and tight control over the library used is required. This option is not intended for general use.

-z compress-class=[!]*class1*,[!]*class2*,...

Specify candidate sections for compression. This option provides fine grained control over the selection of candidate sections to be compressed in the output file. **-z compress-class** is used in conjunction with the **-z compress-sections** option. If **-z compress-class** is specified without **-z compress-sections**, the link-editor applies the default style of compression to the selected sections, as if **-z compress-sections=zlib** option had been set.

The compress class descriptions that follow only apply to non-allocatable sections.

Some classes cause other classes to be implicitly included, or **encapsulated**. Such cases are noted in the class descriptions below. Each class token can be prepended with a '**!**' to indicate that the class should not be included. This definition can be useful to prevent a class from including another normally encapsulated class. For example, while the **symbol** class encapsulates non-allocable sort sections, **-z compress-class=symbol,!sort** targets the non-allocable symbol table, but excludes any associated ELF sort sections.

The following classes of section can be defined.

nonalloc

Compress any non-allocatable section. These sections are identified as not including the **SHF_ALLOC** section flag. This class encapsulates all of the other classes, except for the **shstrtab** class.

annotate

Compress any annotation section. These sections provide information that is used by memory access tools, and coverage related tools. These sections are identified by having a **SHT_SUNW_ANNOTATE** section type.

comment

Compress any comment section. These sections are identified by having a **.comment** section name.

compcom

Compress any compiler commentary section. These sections are identified by having a **.compcom** section name.

ctf

Compress **CTF** (Compact C Type Format) sections. These sections are identified by having a **.SUNW_ctf** section name, and also by having a **SHT_SUNW_ctf** section type.

There are multiple options for compressing **CTF** sections. This form of compression is distinct from, and unrelated to the form of compression provided by the **CTF** data format, which is specified using the **-z ctf=compress** option. See **ctf**(7).

debug

Compress sections commonly used to contain debugging data. Debug sections are identified by having a **.debug***, **.line**, **.stab***, or **.zdebug*** section name. These sections are also identified by having a section type of **SHT_PROGBITS**, **SHT_SUNW_DEBUG**, or **SHT_SUNW_DEBUGSTR**. This class also encapsulates the **compcom** class.

shstrtab

Compress the **.shstrtab** section used to hold **ELF** section names. This option is intended for specialized dynamic objects, and is not recommended for general use.

sort_sym_addr, **sort_sym_name**, **sort**, **sort_sym**

Compress the symbol sort sections associated to **SHT_SYMTAB** symbol tables. Each of these symbol tables typically have associated symbol sort sections, sorted by address, and by name.

sort_sym_addr By-address **.symtab** symbol sort sections (**SHT_SUNW_symsort**, **SHT_SUNW_tlssort**).

sort_sym_name By-name **.symtab** symbol sort sections (**SHT_SUNW_symnsort**).

sort, **sort_sym** All **.symtab** symbol sort sections. Equivalent to specifying both **sort_sym_addr** and **sort_sym_name**.

symbol

Compress any non-allocatable symbol table. These sections are identified by having a **SHT_SYMTAB** section type. Any associated string table is also compressed. This class also

encapsulates the **sort_sym** classes.

-z compress-sections[=*cmp-type*[,cmp-opt...]]
--compress-debug-sections *cmp-type[,cmp-opt...]*

Enables the compression of output sections. The following values for *cmp-type* are recognized.

none

No compression is done. This is equivalent to not specifying the **-z compress-sections** option.

zlib

Compress candidate sections using **ZLIB** compression.

zstd

Compress sections using **ZStandard** (**ZSTD**) compression.

zlib-gnu

Compress sections with **ZLIB** compression, using the older deprecated **GNU** style. Candidate sections must have a name that begins with **.debug**. Each resulting section is renamed to start with **.zdebug** to identify the use of compression.

The following *cmp-opt* option can be specified.

force

By default, the link-editor will only compress sections if their resulting size is equal to or smaller than the original data. Specify **force** to force compression even when the resulting size is larger than the original.

If *cmp-type* is omitted, the **zlib** style is used.

The candidate sections for compression can be specified using the **-z compress-class** option. If **-z compress-sections** is specified without **-z compress-class**, the link-editor defaults to compressing **annotate** and **debug** sections, as if the **-z compress-class=annotate,debug** option had been set.

The **zlib** and **zstd** compression types are implemented using standard **ELF** compression features, and can be applied to a wide variety of sections. The resulting sections will have the **SHF_COMPRESSED** section flag set to identify the use of compression. The **zlib-gnu** style predates the introduction of standard compression to **ELF**, and is more limited. Unless there is a specific requirement to use the **zlib-gnu** style, the more general default **zlib** or **zstd** styles are recommended. See *Compressed Debug Sections* in *Oracle Solaris 11.4 Linkers and Libraries Guide*.

The **zlib-gnu** compression type is limited to sections with a name that starts with **.debug**. When **zlib-gnu** is used, sections that would otherwise be candidates for compression are not compressed. The underlying **ZLIB** compression is identical for the **zlib** and **zlib-gnu** styles, and both formats deliver the same amount of compression for a given input section. The two styles differ in the selection of

candidate sections, the format of the compression header, and in how compressed sections are identified.

-z ctf[=*opt1*,*opt2*,...]

Add a **CTF** (Compact C Type Format) section, **.SUNW_ctf**, to the output object. **CTF** is used by observability tools such as mdb(1) and dtrace(8). The **-z ctf** option provides the capabilities of the **ctfconvert** and **ctfmerge** utilities from within the link-edit, eliminating the need to apply those operations separately. In many cases, **-z ctf** will be simpler to use. Most options provided by **-z ctf** correspond directly to **ctfmerge** options. For more detailed information, see **ctfconvert**(1), and **ctfmerge**(1).

The following options can be specified.

additive-merge=*withfile*

Additive merge against the file given by *withfile*. See **ctf**(7).

compress

Compress the contents of the resulting **CTF** section using native **CTF** format compression.

There are multiple options for compressing **CTF** sections. This form of compression is distinct from, and unrelated to the general section compression provided by the **ELF** object format, which is specified using the **-z compress-class=ctf** option. See **ctf**(7).

convert

Perform the **ctfconvert** operation on any input object that does not already have a **.SUNW_ctf** section, and merge the **CTF** data produced into the output. The input object is not modified. Successful conversion depends on the presence of compatible debug sections. For details, see **ctfconvert**(1).

dynsym

Associate the generated **CTF** section with the dynamic symbol tables, **.dynsym**, and if present, **.SUNW_ldynsym**, rather than the default **.symtab**. In general, use of **.symtab** is preferred for **CTF**, as it contains a superset of the symbols contained in the dynamic symbol tables. However, the use of of **-s** can allow the **.symtab** to be later stripped without also removing the associated CTF. See **strip**(1).

dynsym-only

Associate the generated **CTF** section with the dynamic **.dynsym** symbol table rather than the default **.symtab**. The **.SUNW_ldynsym** symbol table is not included. This is a specialized option, primarily of interest in emulating the behavior of older versions of Oracle Solaris for compatibility testing. This option is not intended for general use.

ignore-non-c

When the **convert** option is used, **ignore-non-c** causes input object files built from languages

other than C to be silently ignored. While **CTF** can be used with any language, the information it captures corresponds directly to C level language concepts, making **CTF** particularly useful with code written in C.

label=*label*

Specifies a label to be associated with the generated **CTF**. Labels are of use when employing **uniquification** to share definitions between parent and children objects as a space saving measure. See **ctf**(7).

label-env=*labelenv*

Specifies a label to be associated with the generated **CTF**. The value of the environment variable specified by *labelenv* provides the label to be used. If the environment variable is not defined, no label is set, and the behavior is as if no label were specified. See **ctf**(7).

require

Require each input file to have a **.SUNW_ctf** section, or when the **convert** option is specified, that the conversion operation succeeds in producing **CTF**.

uniqify-file=*uniqfile*

Uniquify against the file given by *uniqfile*. See **ctf**(7).

uniqify-label=*uniqlabel*

Uniquify against the label given by *uniqlabel*. See **ctf**(7).

version=*ctf-version*

Specify the version of the **CTF** format produced. Valid versions are 2 or 3. Version 3 **CTF**, which is recommended for most purposes, is produced by default. This is a specialized option, primarily of use for testing purposes. See **ctf**(5).

-z deferred | nodeferred

Enables or disables the identification of dynamic dependencies as deferred. Symbol references that bind to a deferred dependency are termed deferred references. Deferred dependencies behave similarly to dependencies identified as lazy loadable, and are not loaded at initial process start-up. The loading of deferred dependencies is delayed until process execution, when the first binding to a deferred reference is made.

Unlike standard symbol references, deferred symbol references are not processed as part of standard relocation processing, or **LD_BIND_NOW** processing, or through **dlopen**(3C) with the **RTLD_NOW** flag. Deferred references can only be established for function calls, and are bound directly to the associated dependency.

The use of deferred dependencies, together with **dlsym**(3C) and the **RTLD_PROBE** handle, provides

a flexible mechanism, and natural coding style, for testing for functionality. See *Testing for Functionality* in *Oracle Solaris 11.4 Linkers and Libraries Guide*, and *Providing an Alternative to dlopen* in *Oracle Solaris 11.4 Linkers and Libraries Guide*.

-z defs | nodefs
--no-undefined

The **-z defs** option and the **--no-undefined** option force a fatal error if any undefined symbols remain at the end of the link. This mode is the default when an executable is built. For historic reasons, this mode is **not** the default when building a shared object. Use of the **-z defs** option is recommended, as this mode assures the object being built is self-contained. A self-contained object has all symbolic references resolved internally, or to the object's immediate dependencies.

The **-z nodefs** option allows undefined symbols. For historic reasons, this mode is the default when a shared object is built. When used with executables, the behavior of references to such undefined symbols is unspecified. Use of the **-z nodefs** option is not recommended.

-z direct | nodirect

Enables or disables direct binding to any dependencies that follow on the command line. These options allow finer control over direct binding than the global counterpart **-B direct**. The **-z direct** option also differs from the **-B direct** option in the following areas. Direct binding information is not established between a symbol reference and an associated definition within the object being created. Lazy loading is not enabled.

-z discap

Disassemble input relocatable objects that lack a **.SUNW_cap** capabilities section in order to determine the hardware capabilities required by the object. The resulting capabilities are recorded in the **.SUNW_cap** section of the output object.

-z discard-unused=*item1*,*item2*,...
--gc-sections | --no-gc-sections

By default, the link-editor discards unused, empty sections. Other categories of input material can be determined to be unused during the link-edit. The **-z discard-unused** option enables the automatic removal of such items. The following *item* tokens are recognized.

sections

Unused sections are discarded from the output file created from the link-edit.

files

Unused relocatable object files are discarded from the output file created from the link-edit.

An input relocatable object file is determined to be unused if all allocatable sections provided by the relocatable object are unused. See also the "Non-Required Relocatable Object Files" discussion of the **-z guidance** option.

dependencies

Unused, explicit, shared object dependencies are not recorded in the output file created from the link-edit.

An explicit dependency is one that is defined on the command line, either using the path name, or more commonly by using the **-l** option. Explicit dependencies can depend on other objects, which are referred to as implicit dependencies. An explicit dependency is determined to be unused if two conditions are true.

- No global symbols that are provided by the dependency are referenced from the object being built.
- The dependency does not compensate for the requirements of any implicit dependencies.

See also the "Non-Required or Compensating Dependencies" discussion of the **-z guidance** option.

none

Disables all unused processing, including the default action of removing unused, empty sections.

For compatibility with **GNU** link-editors, **--gc-sections** is accepted as a synonym for **-z discard-unused=sections**. The **--no-gc-sections** option cancels a previous use of **-z discard-unused=sections** or **--gc-sections**, and otherwise has no effect.

See also *Removing Unused Material* in *Oracle Solaris 11.4 Linkers and Libraries Guide*.

-z endfiltee

Identifies a **filtee** so that when processed by a filter, the **filtee** terminates any further **filtee** searches by the filter. See *Reducing Capability Filtee Searches* in *Oracle Solaris 11.4 Linkers and Libraries Guide*.

-z fatal-warnings | nofatal-warnings
--fatal-warnings | --no-fatal-warnings

The **-z fatal-warnings** and the **--fatal-warnings** option cause the link-editor to treat warnings as fatal errors.

The **-z nofatal-warnings** and the **--no-fatal-warnings** option cause the link-editor to treat warnings as non-fatal. This is the default behavior.

-z finiarray=*function*

Appends an entry to the **.fini_array** section of the object being built. If no **.fini_array** section is present, a section is created. The new entry is initialized to point to *function*. See *Initialization and Termination Sections* in *Oracle Solaris 11.4 Linkers and Libraries Guide*.

-z globalaudit

This option supplements an audit library definition that has been recorded with the **-P** option. This option is only available when creating executables. Audit libraries that are defined within an object with the **-P** option typically allow for the auditing of the immediate dependencies of the object. The **-z globalaudit** promotes the auditor to a global auditor, thus allowing the auditing of all dependencies. See *Invoking the Auditing Interface* in *Oracle Solaris 11.4 Linkers and Libraries Guide*.

An auditor established with the **-P** option and the **-z globalaudit** option, is equivalent to the auditor being established with the **LD_AUDIT** environment variable. See **ld.so.1**(1).

-z guidance[=*item1*,*item2*,...]

Provide guidance messages to suggest the link-editor options that can improve the quality of the resulting object, or which are otherwise considered to be beneficial. The specific guidance offered is subject to change over time as the system evolves. Obsolete guidance offered by older versions of the link-editor may be dropped in new versions. Similarly, new guidance may be added to new versions of the link-editor. Guidance therefore always represents current best practices.

It is possible to enable guidance, while preventing specific guidance messages, by providing a list of *item* tokens, representing the class of guidance to be suppressed. In this way, unwanted advice can be suppressed without losing the benefit of other guidance. Unrecognized *item* tokens are quietly ignored by the link-editor, allowing a given command line to be executed on a variety of older or newer versions of Solaris.

The guidance offered by the current version of the link-editor, and the *item* tokens used to disable these messages, are as follows.

Specify Required Dependencies

Dynamic objects and kernel modules should explicitly define all of the dependencies they require. Guidance recommends the use of the **-z defs** option, should any symbol references remain unsatisfied when building these objects. This guidance can be disabled with **-z guidance=nodefs**.

Do Not Specify Non-Required or Compensating Dependencies

Dynamic objects and kernel modules should not define any explicit dependencies that do not satisfy the symbol references made by the object. Guidance recommends that non-required, or unused dependencies, be removed. Unused dependencies, can fall into one of two categories.

- Explicit dependencies that satisfy no symbol references.
- Explicit dependencies that satisfy no symbol references from the dynamic object being built, but that compensate for implicit dependencies. See the "dependencies" discussion of the **-z discard-unused** option.

Guidance for both of these categories can be disabled with **-z guidance=nounused-dependencies**, or the synonym **-z guidance=nounused**. Guidance for compensating dependencies can be disabled with **-z guidance=nounused-compensators**.

See also *Removing Unused Material* in *Oracle Solaris 11.4 Linkers and Libraries Guide*.

Do Not Specify Non-Required Relocatable Object Files

The output file being created should not contain any information from a relocatable object whose allocatable sections are not referenced by any other objects involved with the link-edit. Guidance recommends that unused relocatable objects be removed. This guidance can be disabled with **-z guidance=nounused-files**.

See also *Removing Unused Material* in *Oracle Solaris 11.4 Linkers and Libraries Guide*.

Do Not Specify Non-Required Archives

Archives that do not contribute content to the output object impose unnecessary overhead on the link-edit. Guidance recommends that unused archives be removed. This guidance can be disabled with **-z guidance=nounused-archives**.

Lazy Loading

Shared object dependencies should be identified for lazy loading. Guidance recommends the use of the **-z lazyload** option should any dependency be processed before either a **-z lazyload** or **-z nolazyload** option is encountered. This guidance can be disabled with **-z guidance=nolazyload**.

Direct Bindings

Dependencies of dynamic objects should be referenced with direct bindings. Guidance recommends the use of the **-B direct**, or **-z direct** options should any dependency be processed before either of these options, or the **-z nodirect** option is encountered. This guidance can be disabled with **-z guidance=nodirect**.

Pure Text Segment

Dynamic objects should not contain relocations to non-writable, allocable sections. If such relocations exist, and provided the **-z textoff** option is not in use, guidance is offered.

When creating shared objects and position-independent executables, this condition typically arises when input files containing non-position independent code are used. When creating dynamic executables, this condition typically arises when unresolved weak references exist. This guidance can be disabled with **-z guidance=notext**.

Mapfile Syntax

All **mapfiles** should use the version 2 **mapfile** syntax. Guidance recommends the use of the version 2 syntax should any **mapfiles** be encountered that use the version 1 syntax. This guidance can be disabled with **-z guidance=nomapfile**.

Mapfile Symbol Matching

Mapfiles should refrain from using **MATCH** expressions within named versions when those versions are intended to establish a long lived and stable **ABI**. Guidance recommends that all symbol names in named versions be listed explicitly. This guidance can be disabled with **-z guidance=nomapfile-symbol-match**.

Mapfile Capability Name

To allow debuggers such as **mdb**(1) to operate with the resulting object, the name specified for **mapfile CAPABILITY** directives should avoid characters that may be interpreted by debuggers as operators. This guidance can be disabled with **-z guidance=nomapfile-capid**.

Library Search Path

Inappropriate dependencies that are encountered by the link-editor are quietly ignored. For example, a 32-bit dependency that is encountered when generating a 64-bit object is ignored. These dependencies can result from incorrect search path settings, such as supplying an incorrect **-L** option. Although benign, this dependency processing is wasteful, and might hide a build problem that should be solved. Guidance recommends the removal of any inappropriate dependencies. This guidance can be disabled with **-z guidance=nolibpath**.

Specify Kernel Modules Explicitly

Kernel modules should be built using the **-z type=kmod** option. Historically, kernel modules were built using the link-editor **-r** and **-dy** options. This older form is ambiguous, and less capable. This guidance can be disabled with **-z guidance=nokmod**.

Avoid Aliased Symbols

Dynamic objects are built with symbol sort sections used by debuggers and other observability tools to efficiently map addresses to symbol names. Such tools can display inconsistent results when multiple symbols with the same value are present. Aliased symbols can be avoided by revising the code, or the **NOSORTSYM mapfile** directive can be used to manually prevent unwanted aliases from being placed in the sort section. This guidance can be disabled with **-z guidance=nosymalias**.

In addition, **-z guidance=noall** can be used to entirely disable the guidance feature. See Chapter 5, *Link-Editor Quick Reference* in *Oracle Solaris 11.4 Linkers and Libraries Guide* for more information on guidance and advice for building better objects.

-z help
--help
-?

Print a summary of the command line options on the standard output and exit.

-z ignore | record
--as-needed | --no-as-needed

Ignores, or records, shared object dependencies that are not referenced as part of the link-edit. These options are positional options, used to toggle how the link-editor handles unreferenced dependencies encountered on the command line. When **-z ignore** is encountered, any subsequent unreferenced dependencies are quietly ignored. When **-z record** is encountered, all dependencies are recorded without regard to whether the dependency is referenced or not.

By default, the link-editor records all dependencies whether or not the dependency is referenced. The

non-positional **-z discard-unused=dependencies** option can be used to alter this initial default. Once the initial setting is established, **-z ignore** and **-z record** can be used to alter the default behavior.

-z initarray=*function*

Appends an entry to the **.init_array** section of the object being built. If no **.init_array** section is present, a section is created. The new entry is initialized to point to *function*. See *Initialization and Termination Sections* in *Oracle Solaris 11.4 Linkers and Libraries Guide*.

-z initfirst

Identifies the object so that its runtime initialization occurs before the runtime initialization of any other objects brought into the process at the same time. In addition, the object runtime finalization occurs after the runtime finalization of any other objects removed from the process at the same time. This option is only meaningful when building a shared object.

-z interpose

Identifies the object as an interposer. At runtime, an object is identified as an explicit interposer if the object has been tagged using the **-z interpose** option. An explicit interposer is also established when an object is loaded using the **LD_PRELOAD** environment variable. See **ld.so.1**(1). Implicit interposition can occur because of the load order of objects, however, this implicit interposition is unknown to the runtime linker. Explicit interposition can ensure that interposition takes place regardless of the order in which objects are loaded. Explicit interposition also ensures that the runtime linker searches for symbols in any explicit interposers when direct bindings are in effect.

-z lazyload | nolazyload

Enables or disables the identification of dynamic dependencies to be lazily loaded. Dynamic dependencies that are identified as **lazyload** are not loaded at initial process start-up. These dependencies are delayed until the first binding to the object is made.

Note -

Lazy loading requires the correct declaration of dependencies, together with associated **runpaths** for each dynamic object used within a process. See *Lazy Loading of Dynamic Dependencies* in *Oracle Solaris 11.4 Linkers and Libraries Guide*.

-z ld32=*arg1*,*arg2*,...
-z ld64=*arg1*,*arg2*,...

The **-z 32** and **-z 64** options are used to specify options that are only interpreted when running the 32-bit or 64-bit class of the link-editor, respectively.

For example, support libraries are class specific, so the correct class of support library can be ensured using:

$ **ld ... -z ld32=-Saudit32.so.1 -z ld64=-Saudit64.so.1 ...**

Prior to Oracle Solaris 11.4, the class of link-editor that was executed was determined by the class of the **ELF** object being created. Now, the class of the link-editor that is executed is always 64-bit. These options are maintained for backward compatibility with older versions of Solaris.

-z loadfltr

Identifies a filter to indicate that **filtees** must be processed immediately at runtime. Normally, filter processing is delayed until a symbol reference is bound to the filter. The runtime processing of an object that contains this flag mimics that which occurs if the **LD_LOADFLTR** environment variable is in effect. See **ld.so.1**(1).

-z mapfile-add=*name*

Adds *name* to the list of known **mapfile** conditional input expression predicates. This option is equivalent to placing the following lines at the top of the first **mapfile** read by the link-editor.

```
$mapfile_version 2
$add name
```

Names entered via **-z mapfile-add** can be used with **mapfile $if** and **$elif** directives to conditionally process **mapfile** input. See Chapter 10, *Mapfiles in the Link-Editor* in *Oracle Solaris 11.4 Linkers and Libraries Guide*.

-z nodelete

Identifies the object as non-deletable at runtime. This mode is similar to adding the object to the process by using **dlopen**(3C) with the **RTLD_NODELETE** mode.

-z nodlopen

Identifies the object as not available to **dlopen**(3C), either as the object specified by the **dlopen()**, or as any form of dependency required by the object specified by the **dlopen()**. This option is only meaningful when building a shared object.

-z parent=*object*

Specifies a *parent* object, which can be a dynamic object against which to link the output object. This option is typically used when creating *plugin* shared objects intended to be loaded by an executable at runtime via the **dlopen()** function. The symbol table from the parent object is used to satisfy references from the **plugin** object. See *Parent Objects* in *Oracle Solaris 11.4 Linkers and Libraries Guide*.

-z preinitarray=*function*

Appends an entry to the **.preinit_array** section of the object being built. If no **.preinit_array** section is present, a section is created. The new entry is initialized to point to *function*. See *Initialization and Termination Sections* in *Oracle Solaris 11.4 Linkers and Libraries Guide*.

-z rescan-now

These options rescan the archive files that are provided to the link-edit. By default, archives are processed once as the archives appear on the command line. Archives are traditionally specified at the end of the command line so that their symbol definitions resolve any preceding references. However, specifying archives multiple times to satisfy their own interdependencies can be necessary.

-z rescan-now is a positional option, and is processed by the link-editor immediately when encountered on the command line. All archives seen on the command line up to that point are immediately reprocessed in an attempt to locate additional archive members that resolve symbol references. This archive rescanning is repeated until a pass over the archives occurs in which no new members are extracted.

-z rescan-start ... -z rescan-end
--start-group ... --end-group
-(... -)

Defines an archive rescan group. This is a positional construct, and is processed by the link-editor immediately upon encountering the closing delimiter option. Archives found within the group delimiter options are reprocessed as a group in an attempt to locate additional archive members that resolve symbol references. This archive rescanning is repeated until a pass over the archives On the occurs in which no new members are extracted. Archive rescan groups cannot be nested.

-z strip-class=[!]*class1*,[!]*class2*,...

Strip a specific class of section from any input objects, preventing these sections from being added to the output file. This option provides fine grained control over the sections that can be omitted from the output file.

The strip class descriptions that follow only apply to non-allocatable sections.

Some classes cause other classes to be implicitly included, or **encapsulated**. Such cases are noted in the class descriptions below. Each class token can be prepended with a '**!**' to indicate that the class should not be removed. This definition can be useful to prevent a class from including another normally encapsulated class. For example, while the **nonalloc** class encapsulates all non-allocable sections, **-z strip-class=nonalloc,!note** removes all non-allocatable sections except for the note section.

Stripped sections are completely removed from the output object. The use of the **-z ancillary** option alters this behavior with regard to the non-dynamic symbol table **.symtab**, the sections associated with it, and to the **.SUNW_ctf CTF** section. By default, these sections are written to both the primary and ancillary objects. If stripped, they are written to the ancillary object only, and are identified as absent in the primary object. See **ctf** and **symbol** for class specific details.

The following classes of section can be defined.

nonalloc

Strip any non-allocatable section. These sections are identified as not including the **SHF_ALLOC** section flag. This class encapsulates all of the other classes, except for the **symbol** class. The **nonalloc** class is often sufficient by itself to remove any unwanted sections.

annotate

Strip any annotation section. These sections provide information that is used by memory access tools, and coverage related tools. These sections are identified by having a **SHT_SUNW_ANNOTATE** section type.

comment

Strip any comment section. These sections are identified by having a **.comment** section name. Alternatively, the **mcs**(1) utility is commonly used to manipulate comment sections.

compcom

Strip any compiler commentary section. These sections are identified by having a **.compcom** section name.

ctf

Strip input **CTF** (Compact C Type Format) sections. These sections are identified by having a **.SUNW_ctf** section name, and also by having a **SHT_SUNW_ctf** section type. Note that this option does not cause an output **.SUNW_ctf** section produced using the **-z ctf** to be stripped. However, it does affect the treatement of such a section within ancillary objects. By default, **.SUNW_ctf** sections are written to both the primary and ancillary objects. If stripped, they are written to the ancillary object only, and are identified as absent in the primary object.

debug

Strip sections commonly used to contain debugging data. These sections are identified by having a **.line**, **.stab***, **.debug***, or **.zdebug*** section name. These sections are also identified by having a section type of **SHT_SUNW_DEBUG** or **SHT_SUNW_DEBUGSTR**. This class also encapsulates the **compcom** class.

exclude

Strip any excludable section. These sections are identified by having a **SHF_EXCLUDE** section flag. This class can be useful when creating a relocatable object. By default, such sections are automatically excluded when a dynamic object, or kernel module, is created, and are retained when creating a relocatable object.

note

Strip any note section. These sections are identified by having a **SHT_NOTE** section type.

sort_dyn_addr, **sort_dyn_name**, **sort_sym_addr**, **sort_sym_name**
sort, **sort_addr**, **sort_name**, **sort_dyn**, **sort_sym**

Strip the symbol sort sections associated to **SHT_DYNSYM** and **SHT_SYMTAB** symbol tables. Each of these symbol tables typically have associated symbol sort sections, sorted by address, and by name.

sort_dyn_addr By-address **.dynsym** symbol sort sections (**SHT_SUNW_symsort**, **SHT_SUNW_tlssort**).

sort_dyn_name By-name **.dynsym** symbol sort sections (**SHT_SUNW_symnsort**).

sort_sym_addr By-address **.symtab** symbol sort sections (**SHT_SUNW_symsort**, **SHT_SUNW_tlssort**).

sort_sym_name By-name **.symtab** symbol sort sections (**SHT_SUNW_symnsort**).

The remaining symbol sort section class tokens provide combinations of the previous items, and are provided for convenience.

sort **sort_dyn_addr**, **sort_dyn_name**, **sort_sym_addr**, and **sort_sym_name**.

sort_addr **sort_dyn_addr**, and **sort_sym_addr**.

sort_name **sort_dyn_name**, and **sort_sym_name**.

sort_dyn **sort_dyn_addr**, and **sort_dyn_name**.

sort_sym **sort_sym_addr**, and **sort_sym_name**.

symbol

Strip any non-allocatable symbol table, providing the output file is not a relocatable object. These sections are identified by having a **SHT_SYMTAB** section type. This class encapsulates the **sort_sym** classes. Any associated string table, symbol sort sections, or **CTF** section are also removed.

If **.symtab**, or its associated sections, are stripped from an object without the use of **-z ancillary**, the sections are completely removed. Specifying the **-z ancillary** option alters this behavior. When ancillary objects are produced, these sections are written to both the primary and ancillary objects. If stripped, they are written to the ancillary object only, and are identified as absent in the primary object.

-z stub

Produces a stub shared object. A stub object is a shared object, built entirely from **mapfiles**, that supplies the same linking interface as the real object, while containing no code or data. Stub objects cannot be used at runtime. However, an application can be built against a stub object, where the stub object provides the real object name to be used at runtime, and then use the real object at runtime.

Stub objects can only be produced for shared objects, and a **mapfile** defining the global symbols to be exported must be supplied. The **-z type=shared** or **-G** option, and the **-M** options are therefore required when **-z stub** is used. When building a stub object, the link-editor ignores any object or library files specified on the command line, and these files need not exist in order to build a stub. Since the compilation step can be omitted, and because the link-editor has relatively little work to do, stub objects can be built very quickly.

See *Stub Objects* in *Linkers and Libraries Guide*.

-z sx=*extension*[=*mode*],...

Specify per-object security extensions. The mode value can be set to **enable** or **disable**. If mode is omitted, the extension is enabled. Multiple instances of this option are allowed. This option is only available when creating executables. See **sxadm**(8).

The following security extensions can be specified.

adiheap

Specifies an Application Data Integrity (ADI) requirement for memory allocators within a process.

adistack

Specifies an Application Data Integrity (ADI) stack protection requirement for a process.

aslr

Specifies the Address Space Layout Randomization (ASLR) requirement for a process.

nxheap

Specifies a non-executable heap (NXHEAP) requirement for a process.

nxstack

Specifies a non-executable stack (NXSTACK) requirement for a process.

ssbd

Specifies the Speculative Store Bypass Disable (SSBD) mitigation requirement for a process.

Note -

Specifying that a security extension be disabled is always honored at process execution. Specifying that a security extension be enabled is only honored if the underlying system provides the extension, otherwise the security extension request is silently ignored.

-z symbolcap

Convert a relocatable object that defines object capabilities into a relocatable object that defines symbol capabilities. See *Converting Object Capabilities to Symbol Capabilities* in *Oracle Solaris 11.4 Linkers and Libraries Guide*.

-z sysroot=*dir*
--sysroot=*dir*

Specify the system root directory. By default, the default search path, and directories specified with the **-L**, or **-YP** options, are interpreted relative to the system root directory **/**. When **-z sysroot** is specified, these paths are instead taken relative to the specified directory. See the **-L** and **-YP** options, and LIBPATH in FILES. **-z sysroot** can be used in conjunction with **-z target** to create objects for a target other than the currently running system.

-z target=sparc | x86

Specifies the machine type for the output object. Supported targets are SPARC and x86. The 32-bit machine type for the specified target is used unless the **-64** option is also present, in which case the corresponding 64-bit machine type is used. By default, the machine type of the object being generated is determined from the first **ELF** object processed from the command line. If no objects are specified, the machine type is determined by the first object encountered within the first archive processed from the command line. If there are no objects or archives, the link-editor assumes the native machine. This option is useful when creating an object directly with the link-editor whose input is solely from a **mapfile**. See the **-M** option. It can also be useful in the rare case of linking entirely from an archive that contains objects of different machine types for which the first object is not of the desired machine type.

-z text | textoff | textwarn

These options can be used in dynamic mode only. The **-z text** option forces a fatal error if any relocations against non-writable, allocatable sections remain. For historic reasons, this mode is not the default when building dynamic objects. However, its use is recommended to ensure that the text segment of the dynamic object being built is sharable between multiple running processes. A shared text segment incurs the least relocation overhead when loaded into memory. See *Position-Independent Code* in *Oracle Solaris 11.4 Linkers and Libraries Guide*.

The **-z textoff** option allows relocations against all allocatable sections, including non-writable ones. This mode is the default when building a shared object.

The **-z textwarn** option lists a warning if any relocations against non-writable, allocatable sections remain. This mode is the default when building an executable.

-z type=*object-type*

Specify the type of object to create. The following object-types are available.

exec

A dynamic executable.

kmod

A kernel module.

pie

A position-independent executable. This option also asserts the **-z text** option.

reloc

A relocatable object. This is equivalent to specifying the **-r** option.

shared

A shared object. This is equivalent to specifying the **-G** option. This option also asserts the **-z text** option.

-z verbose

This option provides additional warning diagnostics during a link-edit. Presently, this option enables the following warnings.

- Suspicious use of displacement relocations.
- Restricted use of static **TLS** relocations when building shared objects.
- Symbol visibility inconsistencies.

In the future, this option might be enhanced to provide additional diagnostics that are deemed too noisy to be generated by default.

-z wrap=*symbol*
-wrap=*symbol*
--wrap=*symbol*

Rename undefined references to *symbol* in order to allow wrapper code to be linked into the output object without having to modify source code. When the **-z wrap** option is specified, all undefined references to *symbol* are modified to reference **__wrap_***symbol*, and all references to **__real_***symbol* are modified to reference *symbol*. You are expected to provide an object containing the **__wrap_***symbol* function. This wrapper function can call **__real_***symbol* in order to reference the actual function being wrapped.

The following is an example of a wrapper for the **malloc**(3C) function.

```
void *
__wrap_malloc(size_t c)
{
        (void) printf("malloc called with %zu\n", c);
        return (__real_malloc(c));
```

```
}
```

If other code is linked with this file using **-z wrap=malloc** to compile all the objects, then all calls to **malloc** call the function **__wrap_malloc** instead. The call to **__real_malloc** calls the real **malloc** function.

The real and wrapped functions should be maintained in separate source files. Otherwise, the compiler or assembler may resolve the call instead of leaving that operation for the link-editor to carry out, and prevent the wrap from occurring.

-?
--help

Print usage message and immediately exit.

SECONDARY OPTIONS

The following options are less commonly used. These options provide for backward compatibility, very specialized features, or options that have been superseded with improved variants.

-a

In static mode only, produces a static executable file. Undefined references are not permitted. This option is the default behavior for static mode. The **-a** option can not be used with the **-z type=reloc** or **-r** options. See **Static Executables** under DESCRIPTION.

-b

In dynamic mode only, provides no special processing for dynamic executable relocations that reference symbols in shared objects. Without the **-b** option, the link-editor applies techniques within a dynamic executable so that the text segment can remain read-only. One technique is the creation of special position-independent relocations for references to functions that are defined in shared objects. Another technique arranges for data objects that are defined in shared objects to be copied into the memory image of an executable at runtime.

The **-b** option is intended for specialized dynamic objects and is not recommended for general use. Its use suppresses all specialized processing required to ensure an object's shareability, and can even prevent the relocation of 64-bit executables.

-B group

Establishes a shared object and its dependencies as a group. Objects within the group are bound to other members of the group at runtime. This mode is similar to adding the object to the process by using **dlopen**(3C) with the **RTLD_GROUP** mode. An object that has an explicit dependency on a object identified as a group, becomes a member of the group.

As the group must be self contained, use of the **-B group** option also asserts the **-z defs** option.

Establishing a group provides a primitive means of controlling the binding of a group of objects. However, better control can be accomplished with direct bindings. See the **-B direct** option.

-B symbolic | symbolic-functions

These options are only available when creating a shared object. The **-B symbolic** option causes global symbol references to be bound to definitions within the object being built, if a definition exists. The **-B symbolic-functions** restricts this binding to function symbols.

Normally, references to global symbols within a shared object are not bound until runtime, regardless of whether a definition of the symbol exists within the shared object. This model allows a definition of the same symbol from an executable, or from another shared object, to override the object's own definition.

The **-B symbolic** options are intended for specialized dynamic objects and are not recommended for general use. Although these options can reduce runtime relocation overhead, they can compromise bindings that expect interposition. To reduce runtime relocation processing, the creation of a version definition or use of direct bindings is recommended over using these options.

Note -

The use of **mapfiles** can provide greater flexibility over defining symbol visibilities, and affecting relocation overhead. See *SYMBOL_SCOPE and SYMBOL_VERSION Directives* in *Oracle Solaris 11.4 Linkers and Libraries Guide*.

-d y | n

When **-d y**, the default, is specified, the link-editor uses dynamic linking. When **-d n** is specified, the link-editor uses static linking. These options, and the mode that the link-editor operates under, are more commonly controlled by higher level options such as **-z type**, **-r**, and **-G**. See also **Static Executables** under DESCRIPTION, and **-B dynamic | static**.

-I *name*
--dynamic-linker *name*

When building an executable, uses *name* as the path name of the interpreter to be written into the program header. The default in static mode is no interpreter. In dynamic mode, the default is the name of the runtime linker, **ld.so.1**(1). Either case can be overridden by **-I** *name*. **exec**(2) loads this interpreter when the **a.out** is loaded, and passes control to the interpreter rather than to the **a.out** directly.

-N *string*

This option causes a **DT_NEEDED** entry to be added to the **.dynamic** section of the object being built. The value of the **DT_NEEDED** string is the *string* that is specified on the command line. This option is position dependent, and the **DT_NEEDED .dynamic** entry is relative to the other dynamic dependencies discovered on the link-edit line. This option is useful for specifying dependencies within device driver relocatable objects when combined with the **-dy** and **-r** options.

-t

Turns off the warning for multiply-defined tentative (common block) data symbols that have different sizes or different alignments. This option is equivalent to specifying the **-z relax=common** option.

-Wl,*option*

The **-Wl,** option provides a means of passing link-editor options through a compiler driver. The compiler driver typically strips the **-Wl,** and passes the remaining option to the link-editor. Instances have occurred, particularly when building complex portable software, in which the link-editor has been called directly with the entire **-Wl,** option. To provide flexibility in such situations, the link-editor strips the **-Wl,** and interprets and remaining option. As such options are usually not under a users control, any unrecognized **-W** options result in a warning, rather than a fatal error, and the option being ignored.

-YP,*dirlist*

Changes the default directories used for finding libraries. *dirlist* is a colon-separated path list. There should be no need to change the default search paths. Default search paths can change in newer releases of the operating system. Use of this option to explicitly dictate complete search paths run the risk of becoming incompatible with newer releases of the operating system. This option is maintained to support the historical use of some compiler drivers. Although supported, such use is not recommended. Additional search paths that are required for a link-edit should be provided with the **-L** option.

-z absexec

Used only when building a dynamic executable. Specifies that references to external absolute symbols should be resolved immediately instead of being left for resolution at runtime. In very specialized circumstances, this option removes text relocations that can result in excessive swap space demands by an executable.

-z altexec64

Execute the 64-bit link-editor. Prior to Solaris 11, the class of link-editor that was executed was determined by the class of **ELF** object being created. Now, the class of the link-editor that is executed is always 64-bit. This option is maintained for backward compatibility with older versions of Solaris.

-z aslr[=*mode*]

This option is equivalent to specifying the **-z sx=aslr** option.

-z combreloc | nocombreloc

By default, the link-editor combines multiple relocation sections when building dynamic objects. This section combination differs from relocatable objects, in which relocation sections are maintained in a one-to-one relationship with the sections to which the relocations must be applied. The **-z nocombreloc** option disables this merging of relocation sections, and preserves the one-to-one relationship found in the original relocatable objects.

The link-editor sorts the entries of data relocation sections by their symbol reference. This sorting reduces runtime symbol lookup. When multiple relocation sections are combined, this sorting produces the least possible relocation overhead when objects are loaded into memory, and speeds the runtime loading of dynamic objects.

Historically, the individual relocation sections were carried over to any dynamic object, and the **-z combreloc** option was required to enable the relocation section merging previously described.

Relocation section merging is now the default. The **-z combreloc** option is still accepted for the benefit of old build environments, but the option is unnecessary, and has no effect.

-z gnu-version-script=*mapfile*
-z gnu-version-script-compat
--version-script *mapfile*

Provides partial support for the **GNU** version script style of **mapfile**. Version scripts are based on the original Solaris version 1 symbol definition syntax, with some extensions. **ld** supports the most common such extension, the use of wildcard characters in the specified symbol names. Other **GNU**-specific extensions may not be supported. **ld** will issue an appropriate error if an unsupported extension is encountered.

For convenience in building software developed with **GNU** version scripts, the native **GNU --version-script** option is accepted as an alias for **-z gnu-version-script**. Due to the partial nature of the support for **GNU** version scripts, the use of **--version-script** must be explicitly enabled by specifying **-z gnu-version-script-compat**.

-z groupperm | nogroupperm

Assigns, or **deassigns** each dependency that follows to a unique group. The assignment of a dependency to a group has the same effect as if the dependency had been built using the **-B group** option.

-z muldefs
--allow-multiple-definition

Allows multiple, global, symbol definitions. By default, multiple symbol definitions that occur between relocatable objects result in a fatal error condition. This option suppresses the error condition, allowing the first symbol definition to be taken. Specialized compiler options, or high levels of compiler optimization, can circumvent the use of this option. See *Fatal Resolutions* in *Oracle Solaris 11.4 Linkers and Libraries Guide*.

-z nocompstrtab

Disables the deduplication of **ELF** string tables, and comment sections. By default, string deduplication is applied to **SHT_STRTAB** sections, to **SHT_PROGBITS** sections that have their **SHF_MERGE** and **SHF_STRINGS** section flags set, and to comment sections.

The **mcs**(1) utility, together with **-c** option, can be used to compress comment sections after an object has been built.

-z nodefaultlib

Identifies the object so that the runtime default library search path, used after any **LD_LIBRARY_PATH** or **runpaths**, is ignored. This option implies that all dependencies of the object can be satisfied from its **runpath**.

-z nodump

This option was historically used to identify the object as not available to **dldump**(3C). The **-z nodump** option is still accepted, but the option is unnecessary, and has no effect.

-z noldynsym

Prevents the inclusion of a **.SUNW_ldynsym** section in a dynamic object. The **.SUNW_ldynsym** section augments the **.dynsym** section by providing symbols for local functions. Local function symbols allow debuggers to display local function names in stack traces from stripped programs. Similarly, **dladdr**(3C) is able to supply more accurate results.

The **-z noldynsym** option also prevents the inclusion of the two symbol sort sections that are related to the **.SUNW_ldynsym** section. The **.SUNW_dynsymsort** section provides sorted access to regular function and variable symbols. The **.SUNW_dyntlssort** section provides sorted access to thread local storage (**TLS**) variable symbols.

The **.SUNW_ldynsym**, **.SUNW_dynsymsort**, and **.SUNW_dyntlssort** sections, which becomes part of the allocable text segment of the resulting file, cannot be removed by **strip**(1). Therefore, the **-z noldynsym** option is the only way to prevent their inclusion.

-z nopartial

Partially initialized symbols, that are defined within relocatable objects, are expanded in the output file being generated.

-z noversion

Eliminates versioning information. Version sections or associated **.dynamic** section entries are not generated in the output image.

-z now

Identifies the object as requiring non-lazy runtime binding. This mode is similar to adding the object to the process by using **dlopen**(3C) with the **RTLD_NOW** mode. This mode is also similar to having the **LD_BIND_NOW** environment variable in effect. See **ld.so.1**(1).

-z nxheap[=*mode*]

This option is equivalent to specifying the **-z sx=nxheap** option.

-z nxstack[=*mode*]

This option is equivalent to specifying the **-z sx=nxstack** option.

-z origin

Identifies the object as requiring immediate **$ORIGIN** processing at runtime. This option is only maintained for historic compatibility, as the runtime analysis of objects to provide for **$ORIGIN** processing is now default.

-z redlocsym
-x
--discard-all

Eliminates all local symbols except for the *SECT* symbols from the symbol table **SHT_SYMTAB**. All relocations that refer to local symbols are updated to refer to the corresponding *SECT* symbol. This option allows specialized objects to greatly reduce their symbol table sizes. See also the **-z strip-class** and **-z noldynsym** options.

Although useful for special objects such as those used within the operating system kernel, the **-z redlocsym** option is not recommended for general use. The size of the symbol table **SHT_SYMTAB** does not affect runtime behavior, and the elimination of local symbols can negatively affect process observability. Eliminated local symbols can reduce the debugging information that is generated using the compiler drivers **-g** option. Eliminated local symbols will also remove the information normally written to the **.SUNW_ldynsym** section, reducing the effectiveness of debuggers and tools such as **pstack**(1) and **truss**(1).

-z relax=*item1*,*item2*,...

The link-editor performs validity checks in order to ensure that the resulting output object is valid and usable at runtime. In addition, the link-editor can transition a variety of relocations to generate more optimal instruction sequences.

The **-z relax** option can be used to relax these operations in order to produce an output object that would otherwise be rejected.

Note -

Disabling validity checks can result in the creation of a corrupt or otherwise unusable object. The **-z relax** option is a specialized option, mainly of interest to compiler authors, and is not intended for general use.

The following **item** tokens are recognized.

comdat

A relocation that remains against a symbol that has been eliminated as part of a **COMDAT** section, results in a fatal error. The **-z relax=comdat** option redirects such relocations to an equivalent symbol in a retained **COMDAT** section.

common

Multiply-defined tentative data symbols, that have different sizes or different alignments, produce a warning. These conditions typically originate from Fortran common blocks. The **-z relax=common** option disables this warning.

relocfit

The SPARC architecture defines how each relocation handles the case in which a calculated relocation value is larger than the intended field. Some relocation types are defined to verify that the

value fits, and others to truncate the result. A verifying relocation that overflows its field results in a fatal error. The **-z relax=relocfit** option suppresses relocation verification.

secadj

ELF objects require certain sections to be adjacent to each other in a specified order in the output object. This automatic layout can be disrupted by link-editor **mapfiles** that explicitly layout sections. An invalid layout results in a fatal error. The **-z relax=secadj** option suppresses layout verification.

symbound

Symbols that refer to data beyond the bounds of their associated section can result in invalid memory accesses. Such symbols result in a fatal error. The **-z relax=symbound** option suppresses this error condition.

transdisp

On SPARC, a family of **GOTDATA** relocations provide access to data relative to the Global Offset Table (**GOT**). Locally bound data items allow the link-editor to transition the associated code sequence to provide a more optimized access model. However, this model only provides for accessing the data within +/- 2 Gbytes of the **GOT** address. A relocation that exceeds this limit results in a fatal error. In the rare case that a data reference exceeds this limit, the **-z relax=transdisp** option can be used to adjust **GOTOP** access model transitions. This relaxation removes the address range limit at the expense of a slightly less optimal code sequence.

transtls

Thread-Local Storage (**TLS**) access provides numerous models for data access. See *Thread-Local Storage Access Models* in *Oracle Solaris 11.4 Linkers and Libraries Guide*. The link-editor can transition the associated code sequence based off the output file being created, and a symbols visibility. The **-z relax=transtls** option suppresses **TLS** access model transitions.

-z relaxreloc

This option is equivalent to specifying the **-z relax=comdat** option.

-z rescan

A position independent option that causes a rescan of the archive files that are provided to the link-edit. The link-editor defers the rescan operation until after it has processed the entire command line, and then initiates a final rescan operation over all archives seen on the command line. The **-z rescan** operation can interact incorrectly with objects that contain initialization (**.init**) or finalization (**.fini**) sections, preventing the code in those sections from running. For this reason, **-z rescan** is deprecated, and use of **-z rescan-now** is advised.

ENVIRONMENT VARIABLES

The following environment variables are recognized by the link-editor. Additional environment variables, prefixed with **LD_**, are available for use with the runtime linker. See **ld.so.1**(1).

LD_ALTEXEC

An alternative link-editor path name. The link-editor executes, and passes control to this alternative link-editor. This environment variable provides a generic means of overriding the default link-editor that is called from the various compiler drivers.

LD_LIBRARY_PATH

A list of directories in which to search for the libraries specified using the **-l** option. Multiple directories are separated by a colon. In the most general case, this environment variable contains two directory lists separated by a semicolon.

*dirlist1***;***dirlist2*

If the link-editor is called with any number of occurrences of **-L**, as in:

$ **ld ... -L***path1 ...* -L*pathn ...*

then the search path ordering is:

dirlist1 path1 ... pathn dirlist2 LIBPATH

When the list of directories does not contain a semicolon, the list is interpreted as *dirlist2*.

The **LD_LIBRARY_PATH** environment variable also affects the runtime linkers search for dynamic dependencies.

This environment variable can be specified with a _32 or _64 suffix. This makes the environment variable specific, respectively, to 32-bit or 64-bit processes and overrides any non-suffixed version of the environment variable that is in effect.

LD_NOEXEC_64

Prior to Oracle Solaris 11.4, the class of the link-editor that was executed was determined by the class of the underlying system, and could be 32-bit or 64-bit. The **LD_NOEXEC_64** environment variable was used to suppress the automatic execution of the 64-bit link-editor. In current systems, the link-editor is always 64-bit, and **LD_NOEXEC_64** is ignored and has no effect. This variable is maintained for backward compatibility with older versions of Solaris.

LD_OPTIONS, **LD_UNSET**, **LD_{object-type}_UNSET**, **LD_{object-type}_OPTIONS**

These environment variables provide a means of adding additional options, or removing existing options, from the link-editor command line. The object-type provides the types, in uppercase, defined by the **-z type** option. Thus, the following object-type options are available.

- **LD_EXEC_UNSET** and **LD_EXEC_OPTIONS**

- **LD_KMOD_UNSET** and **LD_KMOD_OPTIONS**
- **LD_PIE_UNSET** and **LD_PIE_OPTIONS**
- **LD_RELOC_UNSET** and **LD_RELOC_OPTIONS**
- **LD_SHARED_UNSET** and **LD_SHARED_OPTIONS**

See OPTION PROCESSING.

LD_RUN_PATH

An alternative mechanism for specifying a **runpath** to the link-editor. See the **-R** option. If both **LD_RUN_PATH** and the **-R** option are specified, **-R** supersedes.

SGS_SUPPORT

Provides a colon-separated list of shared objects that are loaded with the link-editor and given information regarding the linking process. This environment variable can be specified with a _32 or _64 suffix. This makes the environment variable specific, respectively, to the 32-bit or 64-bit class of the link-editor and overrides any non-suffixed version of the environment variable that is in effect. See the **-S** option.

Notice that environment variable-names that begin with the characters '**LD_**' are reserved for possible future enhancements to **ld** and **ld.so.1**(1).

FILES

lib*x*.so — shared object libraries.

lib*x*.a — archive libraries.

a.out — default output file.

LIBPATH — For 32-bit libraries, the default search path is **/lib**, followed by **/usr/lib**. For 64-bit libraries, the default search path is **/lib/64**, followed by **/usr/lib/64**.

/usr/lib/ld — A directory containing several **mapfiles** that can be used during link-editing. These **mapfiles** provide various capabilities, such as defining memory layouts, aligning bss, and defining non-executable stacks.

ATTRIBUTES

See **attributes**(7) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	system/linker
Interface Stability	Committed

SEE ALSO

as(1), **crle**(1), **ctfconvert**(1), **ctfmerge**(1), **elfcompress**(1), **gprof**(1), **ld.so.1**(1), **ldd**(1), **mcs**(1), **pvs**(1), **strip**(1), **exec**(2), **stat**(2), **dldump**(3C), **zlib**(3), **dlopen**(3C), **elf**(3ELF), **ar.h**(3HEAD), **a.out**(5), **ctf**(5), **attributes**(7), **ctf**(7), **sxadm**(8)

Oracle Solaris 11.4 Linkers and Libraries Guide

https://zlib.net/

https://facebook.github.io/zstd/zstd_manual.html

NOTES

Default options applied by the link-editor are maintained for historic reasons. In today's programming environment, where dynamic objects dominate, alternative defaults would often make more sense. However, historic defaults must be maintained to ensure compatibility with existing program development environments. Historic defaults are called out wherever possible in this manual. For a description of the current recommended options, see Chapter 5, *Link-Editor Quick Reference* in *Oracle Solaris 11.4 Linkers and Libraries Guide*.

If the file being created by the link-editor already exists, the file is unlinked after all input files have been processed. A new file with the specified name is then created. This allows the link-editor to create a new version of the file, while simultaneously allowing existing processes that are accessing the old file contents to continue running. If the old file has no other links, the disk space of the removed file is freed when the last process referencing the file terminates.

The behavior of the link-editor when the file being created already exists was changed with Oracle Solaris 11. In older versions, the existing file was rewritten in place, an approach with the potential to corrupt any running processes that is using the file. This change has an implication for output files that have multiple hard links in the file system. Previously, all links would remain intact, with all links accessing the new file contents. The new link-editor behavior **breaks** such links, with the result that only the specified output file name references the new file. All the other links continue to reference the old file. To ensure consistent behavior, applications that rely on multiple hard links to linker output files should explicitly remove and relink the other file names.