Link-Editor Quick Reference

A

The following sections provide a simple overview, or *cheat sheet*, of the most commonly used link-editor scenarios (see "Link-Editing" on page 2 for an introduction to the kinds of output modules generated by the link-editor).

The examples provided show the link-editor options as supplied to the compiler driver `cc(1)`, this being the most common mechanism of invoking the link-editor (see "Using a Compiler Driver" on page 9).

The link-editor places no meaning on the name of any input file. Each file is opened and inspected to determine the type of processing it requires (see "Input File Processing" on page 11).

Shared objects that follow a naming convention of `lib`*x*`.so`, and archive libraries that follow a naming convention of `lib`*x*`.a`, can be input using the `-l` option (see "Library Naming Conventions" on page 14). This provides additional flexibility in allowing search paths to be specified using the `-L` option (see "Directories Searched by the Link-Editor" on page 16).

The link-editor basically operates in one of two modes, *static* or *dynamic*.

Static Mode

This mode is selected when the `-dn` option is used, and allows for the creation of relocatable objects and static executables. Under this mode only relocatable objects and archive libraries are acceptable forms of input. Use of the `-l` option will result in a search for archive libraries.

Building a Relocatable Object

- Use the -dn and -r options:

```
$ cc -dn -r -o temp.o file1.o file2.o file3.o .....
```

Building a Static Executable

The use of static executables is limited. Static executables usually contain platform specific implementation details which restricts the ability of the executable to be run on an alternative platform. Also, many implementations of Solaris libraries depend on dynamic linking capabilities such as dlopen(3x) and dlsym(3x). (see "Loading Additional Objects" on page 67). These capabilities are not available to static executables.

- Use the -dn option *without* the -r option:

```
$ cc -dn -o prog file1.o file2.o file3.o .....
```

Note – The -a option is available to indicate the creation of a static executable, however, the use of -dn *without* a -r implies -a.

Dynamic Mode

This is the default mode of operation for the link-editor. It can be enforced by specifying the -dy option, but is implied when *not* using the -dn option.

Under this mode, relocatable objects, shared objects and archive libraries are acceptable forms of input. Use of the -l option will result in a directory search, where each directory is searched for a shared object, and if none is found the same directory is then searched for an archive library. A search for archive libraries only, can be enforced by using the -B static option (see "Linking with a Mix of Shared Objects and Archives" on page 15).

Building a Shared Object

- Use the -G option (-dy is optional as it is implied by default).

- Input relocatable objects should be built from position-independent code. Use the `-z text` option to enforce this requirement (see "Position-Independent Code" on page 100).
- Establish the shared objects *public interface* by defining the global symbols that should be visible from this shared object, and reducing any other global symbols to local scope. This definition is provided by the `-M` option together with an associated `mapfile`, and is covered in more detail inAppendix B, "Versioning Quick Reference".
- Use a *versioned* name for the shared object to allow for future upgrades (see "Coordination of Versioned Filenames" on page 136).
- If the shared object being generated has dependencies on any other shared objects, and these dependencies do not reside in `/usr/lib`, record their pathname in the output file using the `-R` option (see "Shared Objects with Dependencies" on page 89).

The following example combines the above points:

```
$ cc -c -o foo.o -Kpic foo.c
$ cc -M mapfile -G -o libfoo.so.1 -z text -R /home/lib \
foo.o -L. -lbar
```

- If the shared object being generated will be used as input to another link-edit, record within it the shared object's runtime name using the `-h` option (see "Recording a Shared Object Name" on page 86).
- Make the shared object available to the compilation environment by creating a file system link to a non-versioned shared object name (see "Coordination of Versioned Filenames" on page 136).

The following example combines the above points:

```
$ cc -M mapfile -G -o libfoo.so.1 -z text -R /home/lib \
-h libfoo.so.1 foo.o
$ ln -s libfoo.so.1 libfoo.so
```

- Consider the performance implications of the shared object; maximize shareability (see page 102) and minimize paging activity (see page 105), reduce relocation overhead, especially by minimizing symbolic relocations (see "Profiling Shared Objects" on page 111), and allow access to data via functional interfaces (see "Copy Relocations" on page 107).

Building a Dynamic Executable

- Don't use the `-G`, or `-dn` options.
- If the dynamic executable being generated has dependencies on any other shared objects, and these dependencies do not reside in `/usr/lib`, record their pathname in the output file using the `-R` option (see "Directories Searched by the Runtime Linker" on page 18).

The following example combines the above points:

```
$ cc -o prog -R /home/lib -L. -lfoo file1.o file2.o file3.o .....
```