3.1 Creating object files

To create an object file from a source file, the compiler is invoked with the `-c` flag (and any other desired flags):

```
burger$ gcc -g -O -c main.c
burger$
```

The above compiler command produces an object file, usually named `main.o`, from the source file `main.c`.

For most library systems, creating object files that become part of a static library is as simple as creating object files that are linked to form an executable:

```
burger$ gcc -g -O -c foo.c
burger$ gcc -g -O -c hello.c
burger$
```

Shared libraries, however, may only be built from *position-independent code* (PIC). So, special flags must be passed to the compiler to tell it to generate PIC rather than the standard position-dependent code.

Since this is a library implementation detail, libtool hides the complexity of PIC compiler flags and uses separate library object files (the PIC one lives in the `.libs` subdirectory and the static one lives in the current directory). On systems without shared libraries, the PIC library object files are not created, whereas on systems where all code is PIC, such as AIX, the static ones are not created.

To create library object files for `foo.c` and `hello.c`, simply invoke libtool with the standard compilation command as arguments (see Compile mode):

```
a23$ libtool --mode=compile gcc -g -O -c foo.c
gcc -g -O -c foo.c -o foo.o
a23$ libtool --mode=compile gcc -g -O -c hello.c
gcc -g -O -c hello.c -o hello.o
a23$
```

Note that libtool silently creates an additional control file on each '`compile`' invocation. The `.lo` file is the libtool object, which Libtool uses to determine what object file may be built into a shared library. On '`a23`', only static libraries are supported so the library objects look like this:

```
# foo.lo - a libtool object file
# Generated by ltmain.sh (GNU libtool) 2.4.6
#
# Please DO NOT delete this file!
# It is necessary for linking the library.

# Name of the PIC object.
pic_object=none

# Name of the non-PIC object.
non_pic_object='foo.o'
```

On shared library systems, libtool automatically generates an additional PIC object by inserting the appropriate PIC generation flags into the compilation command:

```
burger$ libtool --mode=compile gcc -g -O -c foo.c
mkdir .libs
gcc -g -O -c foo.c  -fPIC -DPIC -o .libs/foo.o
gcc -g -O -c foo.c -o foo.o >/dev/null 2>&1
burger$
```

Note that Libtool automatically created `.libs` directory upon its first execution, where PIC library object files will be stored.

Since '`burger`' supports shared libraries, and requires PIC objects to build them, Libtool has compiled a PIC object this time, and made a note of it in the libtool object:

```
# foo.lo - a libtool object file
# Generated by ltmain.sh (GNU libtool) 2.4.6
#
# Please DO NOT delete this file!
# It is necessary for linking the library.

# Name of the PIC object.
pic_object='.libs/foo.o'

# Name of the non-PIC object.
non_pic_object='foo.o'
```

Notice that the second run of GCC has its output discarded. This is done so that compiler warnings aren't annoyingly duplicated. If you need to see both sets of warnings (you might have conditional code inside '`#ifdef PIC`' for example), you can turn off suppression with the `-no-suppress` option to libtool's compile mode:

```
burger$ libtool --mode=compile gcc -no-suppress -g -O -c hello.c
gcc -g -O -c hello.c  -fPIC -DPIC -o .libs/hello.o
gcc -g -O -c hello.c -o hello.o
burger$
```
