Q: I have libraries compiled for NeXT 3.0 machines that I need to use on NEXTSTEP for Intel Processors. What should I do to make them work?

Q: I wrote an application under 3.1 on an Intel NEXTSTEP platform that uses a library I wrote under 3.0, but it won't link properlyÐwhy?

Q: How do I make my 3.0 libraries accessible in a multi-architecture form to 3.1 applications?

A: Libraries that were compiled under NEXTSTEP 3.0 and earlier should work fine under 3.1Ðfor NeXT hardware. Some extra work is required to make them work properly under NEXTSTEP 3.1 for Intel Processors. Because the m68k architecture and the i386 architectures are radically different, it is necessary for NEXTSTEP to have access to a binary for both architectures in order to provide an executable that runs on both machines. Libraries, being binary files containing machine instructions, are no different from any application in this respect. Building multi-architecture applications is easy, since the **Project Builder** application takes care of

the details of building Multi-Architecture Binary files (MAB files) when requested. But **Project Builder** doesn't manage libraries, so an application that you have compiled may fail to link properly because of the lack of a library matching the architecture type of the machine you are compiling for.

3.1 provides a new utility, **libtool**, for generating multi-architecture (or "fat") libraries. **libtool** is intended to replace both the **ar** and **ranlib** utilities. Traditionally, the construction of a library is managed by a **Makefile**, which usually does roughly the following:

1) Compiles the .c and .m (source) files into .o (object) files, using the compiler (**cc**).
2) Archives the resulting .o files into a .a (library) file, using the archiver (**ar**).
3) Generates a table of contents file in the archive, using the **ranlib** utility.

A simple sequence to generate a linkable library starting from just two C files might be:

```
/bin/cc -c -o first.o first.c
/bin/cc -c -o second.o second.c
/bin/ar ruv libsilly.a first.o second.o
/bin/ranlib libsilly.a
```

Under 3.1, however, steps 2 and 3 above are combined into one **libtool** call, so that the equivalent sequence would be:

```
/bin/cc -c -o first.o first.c
/bin/cc -c -o second.o second.c
/bin/libtool -o libsilly.a first.o second.o
```

Notice that the call to **ar**, with the r, u, and v switches on, has been replaced with a call to **libtool**, and that calling **ranlib** is no longer necessary. The r, u, and v switches allowed **ar** to update only the objects that were newer than the ones in the archive; the **libtool** call doesn't have this capability, but is much faster than **ar**, and in addition, it is capable of archiving fat objects, which **ar** cannot do.

So where do multiple-architecture objects come in? So-called "fat" objects are generated when the compiler is told to generate objects of more than one architecture type, using the **-arch** flag. If you look at the compile text field in Project Builder, you should notice that it uses this flag whenever it is asked to generate something for both the m68k and i386 architectures. The change needed to the above sequence is:

```
/bin/cc -arch m68k -arch i386 -c -o first.o first.c
/bin/cc -arch m68k -arch i386 -c -o second.o second.c
/bin/libtool -o libsilly.a first.o second.o
```

And that's all there is to it. The gains to library-building using **libtool** are the ability to make fat libraries, and the elimination of the need to use **ranlib**. The loss is the ability to update only those files in the archive that have timestamps older than the input filesÐin general, the amount of time lost in moving from **ar** and **ranlib** to **libtool** should be minimal, if any.

For the Makefile minded, here's a simple example of what needs to be changed. The first file is the original, which compiles into a single-architecture library, and the second file will generate the multi-architecture library:

```
        ------------------------------
CC          = /bin/cc
RANLIB   = /bin/ranlib
AR          = /bin/ar
ARFLAGS       = ruv

CFLAGS  = -g -O2 -arch m68k -arch i386

SRCS              = first.c second.c

OBJS = $(SRCS:.c=.o)

.c.o: ; $(CC) $(CFLAGS) -c -o $@ $<

all: libsilly.a
```

```
libsilly.a: $(OBJS)
	$(AR) $(ARFLAGS) $@ $(OBJS)
	$(RANLIB) $@

clean: ; /bin/rm -f *.o *.a *~
-----------------------------------
CC          = /bin/cc
LIBTOOL = /bin/libtool

CFLAGS  = -g -O2 -arch m68k -arch i386

SRCS              = first.c second.c

OBJS = $(SRCS:.c=.o)

.c.o: ; $(CC) $(CFLAGS) -c -o $@ $<

all: libsilly.a
```

```
libsilly.a: $(OBJS)
	$(LIBTOOL) -o $@ $(OBJS)

clean: ; /bin/rm -f *.o *.a *~
```

See Also:

Man Pages: **lipo**(1), **libtool**(1), **ar**(1), **ranlib**(1).
Release Notes: Compiler.rtf, FatFiles.rtf, ProjectBuilder.rtf

QA883

Valid for 3.1