Q: I need to use the method **setName:** for both a Sound object and an NXImage object in the same file. However, I always get this warning:

```
warning: multiple declarations for method `setName:'
```

at compilation time. In addition, occasionally the wrong method is called at run-time. How can I get rid of the warning and make sure the correct method is always used?

A: The problem is caused by the fact that NXImage's **setName:** method returns a BOOLEAN, while Sound's **setName:** returns an **id**. You can avoid the problem by using static typing, instead of declaring the objects to be of type **id**. Static typing enables the compiler to do better type checking. The following code snippet shows how to do it:

```
Sound *mySound;                    /* not: id mySound */
NXImage *myImage;    /* not: id myImage */

mySound = [[Sound alloc] init];
myImage = [[NXImage alloc] init];
[mySound setName:"The Beatles"];
[myImage setName:"The Stars"];
```

If static typing is not possible, you can use type-casting instead:

```
id mySound, myImage;

mySound = [[Sound alloc] init];
myImage = [[NXImage alloc] init];
[(Sound *)mySound setName:"The Beatles"];
[(NXImage *)myImage setName:"The Stars"];
```

Note that to prevent similar problems when using duplicate method names for different classes, you should keep their argument types and return value types identical, unless you plan on using static typing to differentiate them.

Why does the compiler not do better type checking by default? The reason is that Objective-C uses dynamic binding (also known as "run-time binding") to give you greater flexibility in deciding which object will be sent a given message. Dynamic binding means that the class of the message's recipient is determined by the Objective-C run-time system rather than by the compiler. This facility lets you send the same message to any of several different classes of object (each declared as type **id**), depending on the current state of the application. A good example of the potential of dynamic binding is InterfaceBuilder, which can manipulate many different types of objects, not all of which existed when InterfaceBuilder was compiled.

The compiler doesn't care if several different classes use exactly the same method name, because the class of the recipient won't be determined anyway until run-time; and as long as the

class implements the method, all is well. In your problematic case, however, the compiler can't figure out which of two similar method names is being invoked by your code, and it may decide on the wrong name.

For more on dynamic binding and Objective-C, see ../Objective-C/why_objective-C.rtf.

QA793

Valid for 2.0, 3.0