Q: In C++, how can I override the error handler for the new operator? I can do this with other compilers, but it doesn't work with the NeXT C++ compiler.

A: On the NeXT, you can override the error for the **new** operator in the following way. Define a routine, say, **newErrorHandler**:

```
static void newErrorHandler()
{
        fprintf(stderr, "new: Failed to acquire sufficient memory.");
        fflush(stderr);
}
```

Then, somewhere in your initialization routines call **set_new_handler**:

```
set_new_handler(newErrorHandler);
```

newErrorHandler is called whenever an error is encountered during the **new** operation.

QA776

Valid for 1.0, 2.0, 3.0

Q: When compiling the following snippet of code with the C++ compiler, I get the error message "Internal Compiler Error 89", and the compiler crashes.

```
extern "C"
{
#include <bsd/libc.h>
#include <mach/cthreads.h>
}

class Test {
        struct condition aCondition;
        struct mutex aMutex;
public:
        void wait()   { condition_wait(&aCondition, &aMutex); }
};

main()
{
        Test   test;
        test.wait();
        exit(0);
}
```

What's happening?

A: This error is caused by a name clash of the symbol **wait** between the definition in the header file **/NextDeveloper/Headers/g++/sys/wait.h** and your member function **wait()**. To work around this bug,

you can rename your **wait()** function to something else, such as **wait_on()**.

QA893

Valid for 3.1, 3.2

Q: When I try to compile my C++ program which contains this header file:

```
typedef enum {
    aaa,
    bbb,
    ccc
} Type;

class myClass {
private:
    Type myType;
public:
    void setType(Type newType);
};
```

I get the link error:

```
/bin/ld:Undefined symbols:_setType__7myClass3$_1
```

and the compilation stops. The same program would compile fine on a non-NEXTSTEP c++ compiler. What's the problem?

A: There is indeed a bug in the current C++ compiler, but the workaround is easy.

The problem is with the declaration:

```
typedef enum {
   aaa,
   bbb,
   ccc
} Type;
```

This causes problems with C++'s name mangling, since this enum has no tag. The current compiler does not know how to encode this type, so it simply uses an integer for the number of the unencodable type. There is no reason to think that this matches the encoding used in some other file which may have had a different number of unencodable types or the same types but in a different order. This is indeed what was happening in your case, and explains why the link failed.

You should just use this declaration instead:

```
enum Type {
   aaa,
   bbb,
   ccc
};
```

In C++, a typedef is automatically published for each struct, union, or enum with the same name as the tag, so you can continue to use "Type" rather than "enum Type".

This bug will be fixed in a future release of the C++ compiler.

QA780

Valid for 2.0, 3.0, 3.1

Q: What version of the C++ compiler is used in NEXTSTEP Release 3.1?

A: The Release 3.1 of the Objective C/C++ compiler is based on version 2.2.2 of the GNU compiler. The GNU compiler has been extended to recognize Objective C constructs within C++ source files. Note that the language g++ accepts is very similar in semantics and syntax, but not identical to that defined by AT&T. Both products are tracking the ongoing ANSI X3J16 standardization work and are converging on the C++ draft standard, due in late 1994.

Q: What development tools are provided with your C++ compiler? Can I use GDB?

A: Yes, GDB, nm, and gprof can be used with C++ source code. With Release 3.1, if your code is compiled with the **-gg** switch, GDB will be able to show C++ methods with more readable names. The user can then insert breakpoints using these names. The **-gg** switch is an abbreviation for the **-ggdb** flag to load the gdb debug symbols.

Q: Do you provide any C++ libraries?

A: libg++ for NEXTSTEP Developer is included in Release 3.2, on the CD-ROM. NeXT does not distribute nor support any previous versions of libg++ for any previous releases of NEXTSTEP Developer.

Q: How do I integrate C++ and Objective C code?

A: Release 3.1 provides on-line documentation on how to write mixed code (Objective C/C++). Here is the list of the on-line reference material:

- Chapter 5 of *Object-Oriented Programming and Objective C* (/NextLibrary/Documentation/NextDev/Concepts/ObjectiveC/5_Programming) discusses how to use C++ with Objective C.

- Chapter 11 of *Development Tools and Techniques* (/NextLibrary/Documentation/NextDev/DevTools/11_Compiler) discusses C++ compiler options.
- The C++ release notes for Release 3.1 (/NextLibrary/Documentation/NextDev/ReleaseNotes/C++.rtf).

QA889

Valid for 3.1

Q: How do I integrate .psw files with C++ files? In my Makefile.preamble, I changed the compiler to cc++. Since *pswrap* generates C code, the C++ compiler will not work with .c files generated by *pswrap*. What can I do?

A: You need to create a **Makefile.postamble** and tell **make** to use **cc** when compiling pswrap'ed files. See the example below:

For NEXTSTEP Release 2.0:

```
.psw.o:
	$(PSWRAP) $(PSWFLAGS) -a -h $*.h -o $*.c $*.psw
	cc $(CFLAGS) -c $*.c -o $(OFILE_DIR)/$*.o
	rm $*.c
```

For NEXTSTEP Release 3.0/3.1:

```
.psw.o:
	$(PSWRAP) $(PSWFLAGS) -a -h $(SYM_DIR)/$*.h -o $(SYM_DIR)/$*.c $*.psw
	cc $(DEFAULT_CFLAGS) -c $(SYM_DIR)/$*.c -o $(OFILE_DIR)/$*.o
	rm $(SYM_DIR)/$*.c
```

You have to remove the **.c** files generated by **pswrap** after the compilation, so that in subsequent makes, the **c++** compiler will not try to compile these files.

In general, you can override the default **make** rules to suit your environment. For example, if you only have one C++ file (e.g. foo.c), add the following to your Makefile.postamble rather than overriding the CC directive in the Makefile.preamble:

for NEXTSTEP Release 2.0:

```
.c.o:
	cc++ $(CFLAGS) -c $*.c -o $(OFILE_DIR)/$*.o
```

for NEXTSTEP Release 3.0/3.1:

```
.c.o:
	cc -ObjC++ $(CFLAGS) -c $*.c -o $(OFILE_DIR)/$*.o
```

See also:
 /usr/lib/nib/Makefile.common for NeXT's default rules for NEXTSTEP Release 2.0, or
/NextDeveloper/Makefiles/app/basicrules.make for NEXTSTEP Release 3.1,
and the ../Objective_C/integrating_C++_code for how to integrate C++ and Objective-C files.

QA795

Valid for 2.0, 3.0, 3.1