OPENSTEP 4.2 Release Notes: Compiler

This file contains developer release notes for the 4.2 release of the compiler.

In the 4.2 OPENSTEP Enterprise, OPENSTEP for Mach, and PDO releases, the compilers on Mach, Windows and PDO are based on the GNU C compiler version 2.7.2.1 and are all built from a single source code base.

Notes From Release 4.2

Calling superclass methods from within a category (78005)

In the 4.2 prerelease, the Objective-C++ compiler would crash when processing code that called a superclass method from within a category. While this bug has been fixed in the final version of OPENSTEP 4.2 Developer for Mach, it still exists in the compilers included with OPENSTEP Enterprise 4.2. Thus, if you're using the Objective-C++ compiler and your category implementations call superclass methods, your code will compile on Mach but not on Windows NT or PDO. Note that this code will compile on Windows NT or on PDO if you use the Objective-C compiler.

Change in meaning of extern "C" to C++ compiler

The C++ and Objective C++ compilers no longer switch the list of valid keywords when they see the **extern "C"** construct. This may cause existing C++ and Objective C++ code to fail to compile. The **extern "Objective-C"** construct can still be used, as in the past, to switch to a mode in which C++-specific keywords such as "class" and "template" can be used as identifiers.

The header files on OPENSTEP and PDO 4.2 have been sanitized and no longer contain uses of C++ keywords as parameter names, struct field names, or function names. This should make C++ usage easier on OPENSTEP and more similar to other C++ development environments.

Recompiling C++ code with this new compiler may require the either the renaming of certain constructs in C header files you use or the use of **extern "Objective-C"** instead of **extern "C"**.

Cleanup of predefined symbols (62096)

The macros NEXT_OBJC_RUNTIME and NEXT_PDO are no longer predefined on the Windows NT compiler. You should no longer depend upon them.

Casting a receiver to conform to a protocol is now working (68626).

If, when invoking a method, you cast the receiver to conform to some protocol in order to make sure the compiler invokes the correct method in cases where there is more than one method with the same name, the Objective-C compiler will now pick up on the hint.

Notes From Release 4.1

Including Windows Header Files in Objective-C Code

In general, you should be able to include any Windows header file in an Objective-C source module without problems. The System framework contains Microsoft's header files, with slight modifications to make them compatible with **gcc**. For instance, slight changes have been made for unnamed unions, Microsoft assembly, and so on. If you have problems including any of the Windows header files, try including the file **winnt-pdo.h** before the Windows header file that's causing problems.

The -Wmost Compiler Flag

The **-Wmost** compiler flag is equivalent to FSF's **-Wall**, except that it doesn't turn on **-Wparenthesis**. **-Wmost** also suppresses warning messages about inline functions and static

constants that are not actually used. This flag is for internal use andits definition may change in a future release.

Notes From Release 4.0

- **Frameworks.** You can now specify frameworks on the linker and preprocessor command lines. The **-framework** flag is accepted by both the linker and the preprocessor, while the **-F** flag is accepted by the linker only. These flags are defined as follows:

  -framework *framework-name*
  Search the framework named *framework-name* when linking. The linker searches a standard set of directories for the framework. It then uses this file as if it had been specified precisely by name. The directories searched by the linker include a couple of standard system directories plus any that you specify with **-F**.

  -F *directory*
  Add the specified directory to the head of the list of directories to be searched for frameworks. If you use more than one **-F** option, the directories are scanned in left-to-right order; the standard framework directories (**LocalLibrary/Frameworks**, followed by **/NextLibrary/Frameworks**) come after.

  In your Objective-C code, include framework headers using the following format:

  ```
  #include <framework/include_file.h>
  ```

  Where *framework* is the name of the framework (such as ªAppKitº or ªFoundationº; don't include the extension) and *include_file* is the name of the file to be included.

- If the name of your source file ends in **.cc**, **.cxx**, **.cpp**, or **.C**, **gcc** will attempt to compile your program with the C++ compiler. Similarly, if the name of your source file ends in **.mm** or **.M**, **gcc** will attempt to compile your program with the Objective-C++ compiler.

- The Objective-C++ compiler is now much more useable than the ones included with OPENSTEP for Windows Prerelease 3 and with PDO 4.0.

- **Position-Independent Code Generation (PIC)**. The way the compiler generates code has changed. It now generates position-independent code by default when it builds libraries, bundles and executables. You can control the code generation style using the **-dynamic** and **-static** compiler flags; **-dynamic** specifies that position-independent code generation is to be used, whereas **-static** specifies position-dependent code generation. For related information, see the note on drivers and kernel servers below.

- **C++ Templates**. The compiler has been updated to support C++ templates or parametrized types. For example, consider the following code:

```
#include <stream.h>
#include <String.h>
#include <SLList.h>

typedef SLList<String> StringList;
main(){
 StringList listOfnames;

 listOfnames.append("hello world");
 cout <<listOfnames.remove_front() << "\n";
}
```

Then the above code is built and run:

```
%> cc++ template.cc -o test -lg++
%> test
hello world
%>
```

- **Debugging features.** The compiler includes a couple new options to assist in debugging. Specify the **-H** flag on the command to have the compiler emit a listing of included header files (indented to reflect where they are included). Specify the **-dM** option after the **-E** (preprocess) option to get a listing of all macros along with their full definitions.

- **Building drivers and kernel servers**. If you are building drivers and kernel servers, be sure to include **-static** on the command line so that position-dependent code is generated. Compilation with the **-dynamic** option assumes that the dynamic link editor (**/usr/lib/dyld**) is present in the running program, and that is not the case for modules to be loaded into the kernel.

Known Bugs and Limitations

The following bugs or limitations are worth noting for the GNU C and C++ Compilers for this release.

- **Mach programs may need to be recompiled**. Since release 4.1 on Mach, **libgcc** has changed in an incompatible way. It's possible that a small number of existing applications rely on the old **libgcc** and may crash because of the new **libgcc** in the base system libraries. We anticipate that this compatibility problem will be resolved in the final version of OPENSTEP 4.2 on Mach; at that time, it may be necessary for you to again recompile anything that now must be recompiled because of this fix.

- **Inline functions may not be properly expanded**. In some circumstances, inline C functions and C++ member functions may not be emitted properly, causing assembler errors. You can work around this problem by ordering your inline function or member-function definitions ahead of their references. Passing **-fkeep-inline-functions** to the compiler also works around this problem, but may sometimes cause other code-generation problems.

- **Keyword-switching for extern "Objective-C" may be delayed**. Although keyword-switching is no longer done inside a context tagged by the **extern "C"** linkage directive, the C++ keywords are turned off inside an **extern "Objective-C" { ...}** range. When entering and exiting this context, the actual switch in keyword sets may occurs a token or two late, meaning that you may get syntax errors on legal code. For example, if the first token following an **extern "Objective-C"** range is "class", this will be lexed as an identifier and not

the C++ class keyword. You can work around this by re-ordering your declarations or inserting a dummy declaration after a keyword-switching boundary.

- **Link errors on Windows NT** (69211). Programs on Windows NT must add explicit references to at least one class in each framework in order to avoid link errors at run time. For instance, you could add a function like that in the following code excerpt, which refers to classes in each of Enterprise Objects Framework's layers. Though never invoked, it forces the appropriate linking to occur.

```
#ifdef WIN32
#import <EOControl/EOControl.h>
#import <EOAccess/EOAccess.h>
#import <EOInterface/EOInterface.h>

void _referenceAllEOFrameworks()
{
    [EODisplayGroup new];          // EOInterface
    [EOEntity new];                // EOAccess
    [EOEditingContext new];        // EOControl
}
#endif
```

  If you create your project with the type "EOF Application," this code is automatically added to your project main file.

- **The -Wno-precomp flag is not supported** (63746). The precomp-related options are not yet supported on Windows NT.
- **Constant strings (both char * and NSString) should be 7-bit only**. Unless constant strings are 7-bit, your code will be non-portable as compilers will deal with 8-bit strings in a machine-dependent encoding.
- **Objective-C++ global constructors**. The Objective-C++ compiler sometimes ignores global constructors; they don't always get called.
- **Random name given to executable by default** (66861). If you create an executable and don't use the **-o** flag to explicitly tell **gcc** what to name it, **gcc** will most likely give it a random name.
- **Using pipes to communicate between compiler passes** (61306). The **-pipe** flag doesn't

work in the Windows NT compiler.

- **Using -pipe** (67853). Although the compiler rarely crashes, if it does, it may generate some assembly language output before doing so. If you use the **-pipe** flag, the assembler may not be able to detect the fact that its input is incomplete. As a result, the assembler may produce an incomplete, but valid **.o** file. If you use the **make** utility to build your application, **make** will detect the fact that there was a problem during compilation. However, when **make** is subsequently invoked, it might not recompile the source file that caused the problem, and the linker will most likely complain about unresolved external symbols.
- **-ObjC++ requires -lstdc++ when using C++ streams on PDO** (69156). When compiling C++ programs that use C++ streams with **gcc** on PDO platforms, if you specify the -**ObjC++** flag you must also specify the **-lstdc++** flag. So, for example, a program "foo" that uses **cout** (and therefore includes **iostream.h**) would be compiled using **gcc** as follows:

```
gcc -ObjC++ foo.cc -lstdc++
```

- **__declspec(dllexport) __stdcall doesn't work** (69194). On Windows NT, functions that are declared as **__declspec(dllexport)__stdcall** aren't handled properly. This may affect some Windows header files that you include in your programs.
- **The compiler sometimes tries to create a library instead of an executable** (69087). This happens on Windows NT when a function is declared as **__declspec(dllimport)** (perhaps in a header file), but the function is actually defined in the file being compiled. The workaround is to remove the offending **__declspec(dllimport)**.
- **Inconsistent function declarations involving stdcall produce unexpected results** (69506). On Windows NT, if a function is forward-declared to be **stdcall** but not declared to be **stdcall** in the actual function definition, the compiler will emit code to pop the arguments off the stack, but won't adjust the function name.
- **The linker complains about objects exported as CONSTANT** (70212). On Windows NT, if you're building a framework and you create your own DEF file for it, defining exported objects as CONSTANT will produce a warning from the linker advising you to use the word DATA instead. If you substitute the word DATA for CONSTANT in your DEF file, some or all of your objects won't be exported correctly; the linker will be unable to find them. As a workaround, simply leave the declarations CONSTANT and ignore the linker warnings.

- **-static option causes linking to fail** (70326). The **-static** and **-dynamic** compiler flags are meaningless on Windows NT, and shouldn't be used.
- **Static constructors can't be used for run-time class initialization** (54831). The PDO compiler can't apply a user-defined constructor to a global or static C++ object and send an Objective-C message in the same file. To work around this problem, eliminate the constructor, the global, or the Objective-C code.
- **wchar_t string literals** (38759). The compiler generates wide-character literals for the host endian-ness only. For example, if you are cross-compiling the string L"x" from m68k to i386, it will be a big endian wide string. The only workaround is not to cross-compile modules which depend on wide characters.