Title: OPENSTEP 4.1 Release Notes
Entry Number: 2473
Last Updated: February 20, 1997

OPENSTEP 4.1 Release Notes

This file contains a complete set of release notes for the following products:

- OPENSTEP Enterprise Release 4.1 (including PDO)
- OPENSTEP for Mach Release 4.1

Information specific to only one of the above products are identified in the individual note.

This file is divided into the following major sections:

- Important Notes (contains extremely important information about the release that should be read by all OPENSTEP developers and installers)
- Compiler Tools
- Compiler (OPENSTEP Enterprise)
- Compiler (OPENSTEP for Mach)
- Debugger

- Documentation
- Driver Development
- Interface Builder
- Netware
- NEXTIME Services
- PDO
- Printing
- Project Builder
- TextEdit
- True Type
- Window Server

The following topics are release noted in separate NeXTanswers:

- Application Kit Framework (2469)
- Foundation Framework (2470)
- Enterprise Objects Framework

Important Notes

The following notes are of particular importance, and should be reviewed by all OPENSTEP developers and installers.

Installing OPENSTEP Enterprise

- **Cannot install under directory names with spaces** (73170). Installation under any directory names or paths containing spaces will not work. For example, trying to install under **C:\Program Files\NeXT** will leave the installation unusable. Install OPENSTEP Enterprise only in a directory the complete path to which contains no spaces.

- The automated uninstaller included with OPENSTEP Enterprise doesn't clean up your System environment variables after a successful uninstallation and reboot. The system environment still contains OPENSTEP-specific entries for LIB and PATH. While this isn't necessarily harmful, successive install-uninstall cycles will cause your system environment to grow. To work around this problem, after uninstallation manually edit your system environment variables as described in the uninstallation section of the *OPENSTEP Enterprise Installation Guide*. (72486)

- **The new Daylight Savings Time (DST) rule in Germany isn't accounted for** (67469). In Germany the rules for ªdaylight saving timeº have changed. From now on, Daylight Savings Time ends on the last Sunday of October, instead of the last Sunday in September. Because Foundation doesn't account for this, unless you supply a new file for **/etc/zoneinfo,** all NeXT computers in Germany will use the incorrect time (including timestamps) between September 29th and October 27th.

  This bug affects all customers in Europe running NEXTSTEP 3.3, NEXTSTEP 3.3risc, 3.3Patch, OPENSTEP for Mach (all versions), and OPENSTEP Enterprise (all versions). NeXT's Technical Support department has issued a patch to address this particular problem; consult NeXTanswers for more

information.

Installing OPENSTEP for Mach

- **Upgrading SPARC Systems from NEXTSTEP 3.3 to OPENSTEP 4.1** At the conclusion of the upgrade process you're told to reboot your computer. On SPARC, systems, this reboot will fail and Workspace Manager will appear to be locked up. Since your system *has* been successfully upgraded, however, it's safe to reboot your SPARC system manually.

  To manually reboot your SPARC, first wait for a minute or two (to allow your disk to sync up) and then press Command-NumLock. When prompted, select "r" to restart your system.

Binary Incompatibility

PDO 4.2 on Solaris is binary-incompatible with PDO 4.0 and previous PDO releases. You must re-link all executables on a PDO 4.2 system before they will run on PDO 4.2.

Known Problems in Deployed Systems

The following two problems are known to have occured in OPENSTEP Enterprise systems configured for deployment as well as for development.

Reference: 72130

Problem: Text could display incorrectly.

Description: Depending on the length of the document you have and how fast you are typing, displayed text might sometimes have problems. For instance, you might get overlapping lines or large blank areas between two lines.

Workaround: Typing will usually fix this right away. Note that in all cases, even though the display is garbled, your document is still correct, so there's no need to exit or restart the application.

Reference: 71671

Problem: "Flash active caption bar" visual warning option might cause applications to hang.

Description: If you have enabled SoundSentry (under Accessibility Options in the Control Panel under NT 4.0), and turned on the "Flash active caption bar" visual warning option, applications could hang when they attempt to beep.

Workaround: None. Use one of the other visual warning options.

New Linker

The Microsoft linker is now the default for project builds on Windows with OPENSTEP Enterprise. This linker is the same one that ships with Visual C++. For information on this linker, consult the Visual C++ documentation.

Project Builder

- **Pasting text into a .m file can cause Project Builder to crash** (73263). Pasting text starting with a tab after an unterminated comment in a source file can cause Project Builder to crash. In addition, typing an unterminated commentÐa slash, an asterisk (*), and a spaceÐand then pressing return can cause Project Builder to crash on Mach systems.

Uninstalling OPENSTEP Enterprise

The uninstaller supplied with OPENSTEP Enterprise doesn't remove OPENSTEP-specific references from your system environment variables. This can be done manually after uninstallation by simply deleting the NEXT_ROOT environment variable and removing all invalid references from both your **Path** and **Lib** environment variables.

If you would like more detailed instructions on how to make these changes, refer to the relevant step in "Uninstalling OPENSTEP for Windows" in the *OPENSTEP Enterprise Installation Guide*. Note that this guide is supplied in printed form only, and is packaged with the OPENSTEP Enterprise CD-ROM.

Compiler Tools

This file contains release notes for the 4.1 release of the Compiler Tools. It contains information about the following topics:

- The NeXT Mach-O GNU-based assemblers
- The NeXT Mach-O static link editor
- The NeXT Mach-O dynamic link editor
- Mach-O object tools (**nm**, **otool**, and so on)

Notes Specific to Release 4.1

New Features

There is only one new feature for the 4.1 release:

- The dynamic linker now has the environment variable DYLD_FRAMEWORK_PATH to better support the development of frameworks. See the man page for details.

Notes Specific to Release 4.0

New Features

Dynamically Linked Shared Libraries

The compiler tools now allow you to build and develop dynamic shared libraries and support the programs that use these libraries, including programs that use bundles. The tools to build or use dynamic shared

libraries are in the 4.1 updates for the m68k, i386, sparc, and hppa target architectures.

All object files that are part of a dynamic shared library or that are to be in an executable should be compiled with the **-dynamic** flag. The **-dynamic** flag is now the default. Executables using shared libraries must also be linked with this flag when using **cc**(1) or **ld**(1) and must be using **crt1.o** (this is done automatically with **cc**(1) and the **-dynamic** flag). To build a dynamic shared library, use **libtool**(1) with the **-dynamic** option; typically you also specify the **-install_name** *library_name* option as well as other options (see the **libtool**(1) man page).

The dynamic linker is becoming more fully-featured. It contains some programmatic support for runtime loading (but as of yet no unloading or replacing).

The static link editor uses the symbol tables of dynamically-linked shared libraries to cause modules for undefined symbols to be pulled in from static libraries and to check for undefined symbols. The **ld**(1) flags for undefined checking **-undefined** {**error**, **warning**, **suppress**} can be used; the default is to treat undefined symbols as errors. This default holds for dynamically-linked shared libraries and bundles. In these cases the dependent libraries should be listed if available. If not available then **-U** _*symbol* or -**undefined** {**warning**, **suppress**} can be used until they are.

Using and Building Dynamic Shared Libraries from Project Builder

The 4.1 version of Project Builder supports a library project type which by default is a dynamic shared library and a framework project type which by default contains a dynamic shared library. Project Builder supports the use of dynamic libraries in other project types. To build static libraries, un-comment the LIBRARY_STYLE definition in the project makefile (LIBRARY_STYLE is set to STATIC and commented out).

Tools that are complete

- The assember, link editor, and **otool** support position-independent code for the m68k, i386, sparc, and hppa architectures. The link editor, assember and **otool** fully support indirect undefined references using symbol pointers and symbol stubs. And the assembler now fully supports Mach-O files.
- The dynamic link editor now has the first level of support for runtime loading. Currently only loading of MH_BUNDLE files is supported. Comments in the header file **/NextDeveloper/Headers/mach-o/dyld.h** describe what is not yet implemented.

Tools that are not complete

- The dynamic link editor now has the first level of support for runtime loading. Currently only loading of MH_BUNDLE files is supported. Comments in the header file **/NextDeveloper/Headers/mach-o/dyld.h** describe what is not yet implemented.
- The tool **segedit**(1) does not have support for dynamic shared libraries.

The NeXT Mach-O GNU-based Assemblers

The NeXT assembler now a fully supports Mach-O files with the ability to create arbitrary sections with the **.section** and **.zerofill** directives. Contents of sections now correctly reflect the assembly code with respect to the section aligment and no unnecessary padding is added.

Major New Features

Support for position-independent code through the use of a new relocatable form of an expression: "*add_symbol* - *subtract_symbol* + *offset*" where *add_symbol* and *subtract_symbol* can be defined in different sections.

Documentation

Assembler Manual

The assembler manual has been updated to reflect support for dynamically-linked shared libraries. It contains appendices of the instructions for the i386, M68K, and PA-RISC processor architectures. An RTF version of the Assembler manual is in:

/NextLibrary/Documentation/NextDev/Reference/DevTools/Assembler

Compiler (OPENSTEP Enterprise)

This file contains developer release notes for the Windows and PDO compilers shipped with OPENSTEP Enterprise Release 4.1.

In this release, the compilers are based on the GNU C compiler version 2.7.2.

New Features

- 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.

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 and is not officially supported. Its definition may change in a future release.

Known Bugs and Limitations

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

- **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.

- **Don't depend on NEXT_OBJC_RUNTIME being predefined** (62096). Although the macro NEXT_OBJC_RUNTIME is predefined on the Windows NT compiler, your code shouldn't depend on this being the case, since this macro won't be predefined in future releases of the compiler.
- **Casting a receiver to conform to a protocol doesn't work** (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 not pick up on the hint. The work around is to either compile your source file with the Objective-C++ compiler (use the **-x objective-c++** switch) or assign the object returned by the method invocation to some (temporary) variable that has the same type as that object.

- **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 does not work.
- **-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)**.

- **-Wno-precomp flag isn't supported** (63746). On Windows NT and on PDO platforms, **gcc** refuses to do anything when the **-Wno-precomp** flag is specified. Currently, the precomp option isn't supported.
- **Reporting bugs** (68122). In some cases, when the compiler detects an internal error, it prints a message requesting that you send a bug report to some electronic mail address. Please disregard this message. All compiler bugs should be reported to the appropriate channels at NeXT Software.
- **cc -MM Foo.mm produces Foo.mm.o** (40491). Compiling a file that has either a **.M** or **.mm** extension with **-M** or **-MM** will produce a file whose name is the same as the source file with the addition of **.o**. The original extension is not stripped off before constructing the name of the **.o** file.
- **-traditional-cpp acts differently on Mach than on NT or PDO** (60175). On Windows NT and on PDO, **-traditional-cpp** changes the behavior of the preprocessor in the same way that **-traditional** does. This can result in compilation errors when recompiling an existing NEXTSTEP project on Windows NT or on PDO.
- **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.
- **Programs that use posing sometimes crash** (72283). If a program compiled with the Objective-C compiler contains an object that accesses its superclass while posing on a category, it will crash. To workaround this problem, either rewrite your code so that it doesn't do this, or compile the program with the Objective-C++ compiler (to do this, use the **-x objective-c++** switch).
- **Objective-C++ is broken in PDO on Solaris** (69089). In this release of PDO, **gcc** doesn't compile Objective-C++ code correctly on Solaris machines. However, you can compile both Objective-C and C++ codeÐproviding that they are in separate source filesÐwith **gcc**.
- **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.

Compiler (OPENSTEP for Mach)

This file contains developer release notes for the 4.1 and 4.0 releases of the compiler. Items specific to or introduced in this release of OPENSTEP Release 4.1 are listed first, followed by items noted in earlier releases.

Notes Specific to Release 4.1

Known Bugs and Limitations

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

- **Constant strings (both char * and NSString) should be 7-bit only** (52695). 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.

- **Missing precomps cause compiler warnings** (73313). If you only install the DeveloperLibs and EO2Developer packages for a single architecture (ªthinº), precomps won't be generated and the compiler will generate numerous warnings. While these warnings can safely be ignored, there are a couple of workarounds: either install the DeveloperLibs and EO2Developer packages for all three supported architectures (ªfatº), or run fixPrecomps manually to generate the missing precomp files. To do this, simply log in as root and type the following in a terminal window:

  ```
  /usr/bin/fixPrecomps
  ```

Notes Specific to Release 4.0

In this release, the Mach compiler is based on the GNU C compiler version 2.5.8.
The compilers Windows and PDO compilers shipped with the OPENSTEP Enterprise release are based on

the GNU C compiler version 2.7.2.

New Features

- **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 ªFoundationºÐdon't include the extension) and *include_file* is the name of the file to be included.

Known Bugs and Limitations

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

- **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.

- **Reporting bugs** (68122). In some cases, when the compiler detects an internal error, it prints a message requesting that you send a bug report to some electronic mail address. Please disregard this message. All compiler bugs should be reported to the appropriate channels at NeXT Software.

Notes Specific to Release 4.0 PR2

In this release, the compiler is based on the GNU C compiler version 2.5.8.

New Features

- **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.

Changes

- **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 pre-release, since they were not mentioned in earlier releases.

- **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.

The GNU Source-Level Debugger

This file contains information about GDB, the GNU Debugger. For more information, see the debugging chapter in the *OPENSTEP Development Tools Reference* manual. On Mach, you may also refer to the **gdb**(1) manual page. On PDO platforms, see the **gdb** reference in

/NextLibrary/Documentation/NextDev/DevTools/Debugger.pdf.

Notes Specific to GDB on Mach:

The **gdb** debugger in OpenStep 4.1 for Mach is based on the version 4.14 release from GNU/FSF. This brings with it many bug fixes and new features, many of which are mentioned in the debugging chapter of the *OPENSTEP Development Tools Reference* manual.

New Features

Dynamic Link-Editor Support

GDB supports debugging of dynamic shared libraries (sometimes known as Frameworks or Bundles in the OPENSTEP world). The presence of dynamic shared libraries has some impact on debugging, which is described in this section.

Debugging symbols for dynamic shared libraries are not present in the program itself. GDB obtains them when the running program attaches and links to the shared library. This means that before the program is actually running, GDB has no information about the contents of the dynamic shared libraries that it uses.

Because of this, it is not possible to set ordinary breakpoints in a shared library before that library has been attached. GDB provides a new command for this purpose, **future-break** or **fb**. If GDB cannot find the necessary symbols to resolve a **future-break** command, it defers the breakpoint and attempts to resolve it later, when new symbols from a shared library become available. Caveat: since the **future-break** command

deals with names and symbols that are as yet unknown to the debugger, it cannot check spelling for you; if you make a spelling mistake, it will never be detected and the breakpoint will never take effect.

There is also an environment variable, **DYLD_LIBRARY_PATH**, which tells the dynamic link-editor where to search for dynamic libraries. This variable can be used to cause a library with debugging symbols to be linked, even though the library on the default path has no symbols. This environment can be set from within GDB by using the **setenv** command. In order to affect the program being debugged, it should be set before running the program.

The "view" interface

In prior releases, GDB supported a GUI interface that used the NEXTSTEP Edit application as a source file viewer (invoked by the **view** command). Edit has been replaced by Project Builder as the source file viewer for GDB, and the **view** command now connects GDB to Project Builder. You must start Project Builder yourself before giving GDB the **view** command (GDB will not start Project Builder automatically). Project Builder has its own user interface for interacting with GDB (see the Project Builder documentation).

Methods with Variable Number of Arguments

GDB now understands the syntax for calling a method with a variable number of arguments (for example, **[MyClass myMethod: 1, 2, 3, 4]**).

Known Problems

Known Problems with Dynamic Link-Editor Support

It has been observed that GDB sometimes hangs or crashes if you run a program that uses a dynamic shared library (Framework or Bundle), then recompile the dynamic shared library, and run the program again. If this happens to you, we recommend that you quit GDB and start a new debugging session every time you rebuild the library. You can use the **.gdbinit** file to help re-establish things such as breakpoints that you need in your debugging session.

Interrupting with ^C during Dynamic Symbol Loading

Immediately after you start your program running under GDB, the program will start to load dynamic shared libraries, and GDB will begin reading symbols from these libraries. If you attempt to interrupt GDB by typing ^C (control-C) during this process, the debugger will be left in a confused internal state from which the only known recovery is to quit the debugger and start over.

Notes Specific to GDB on Windows:

The GDB debugger for OPENSTEP for Windows is based on the version 4.15.1 release from GNU. This brings with it many, if not most of the features of debugging on UNIX and Mach, although there are inevitably some differences.

New Features

Dynamically Loaded Library (DLL) Support

GDB supports debugging of dynamically loaded libraries (DLLs). In OPENSTEP for Windows, Frameworks

and Bundles are implemented as DLL's. The presence of DLLs has some impact on debugging, which is described in this section.

Debugging symbols for dynamically-loaded libraries are not present in the program itself. GDB obtains them when the running program attaches and links to the DLL. This means that before the program is actually running, GDB has no information about the contents of the DLLs that the program uses.

Because of this, it's not possible to set breakpoints or access data in a DLL before the DLL has been attached by the program. If you need to debug code in a DLL, or set breakpoints on a function or method in a DLL, you should first set a breakpoint on **main** (or some place in the program after the DLL has been attached but before the code you want to debug has been executed). Run your program up to that point. The program will load the DLL, and GDB will gain access to it's symbols, after which you should be able to set breakpoints in the DLL. If the DLL was built with the OpenStep compiler (**gcc**), and with **ª-gº** debugging symbols, you should be able to step into functions or methods in the DLL just like ordinary functions or methods in your program.

Attach and Detach

GDB's **attach** command works pretty much as it does on Mach. The process ID to attach to can be obtained from PVIEW, or by having the attachee call **getpid()** and output the result. However, when GDB attaches to an already-running process, it won't learn about symbols from DLLs that the process has already linked to.

The **detach** command is only partially useful at this time. **detach** will let the attached process continue to run, but a parent/child relationship continues to exist between the debugger and the detached process. Because of this, if you then quit the debugger, the detached process will also die.

When you attach to a running process, you will frequently find yourself in a non-debuggable area of code from which you cannot even get a backtrace. See the section on known problems, "Non-Debuggable Code" below for more information on this topic.

Add-Symbol-File

When GDB attaches to a running process, it does not read any symbols from DLLs that the process has previously linked itself to. If you need those symbols for debugging, you can explicitly cause GDB to load them by using the **add-symbol-file** command. This command takes two arguments: the fully-qualified filename of the DLL file, and a base address. For a DLL that you build, this base address will be the base address at which you linked the DLL plus 0x1000. Here are the base addresses to give to the **add-symbol-file** command for the major OPENSTEP DLLs:

- 0x30001000 System
- 0x31001000 nextpdo
- 0x32011000 Foundation
- 0x34021000 AppKit
- 0x38031000 NeXTApps
- 0x3A041000 DevKit
- 0x3C051000 ProjectBuilder
- 0x3C051000 InterfaceBuilder
- 0x40001000 Message
- 0x42011000 WebObjects
- 0x44021000 EOControl
- 0x46031000 EOAccess
- 0x48041000 EOInterface

- 0x4A051000 Sybase
- 0x4C061000 Oracle
- 0x4E071000 Informix
- 0x50081000 nextorb
- 0x52091000 EOModeler

Command Editing

GDB's command line history can be accessed by using the up and down arrow keys, or by using the EMACS key bindings (^P and ^N to scroll thru previous commands, ^B, ^A, ^E to move around on a line, and so on). Also, **set history expansion on** enables C-shell-like command history within GDB (**!!**, **!print** and so on). As usual, an empty newline repeats the previous command (except where specifically disabled, as with the **run** command).

Debugging Objective-C: Differences from Mach GDB

The Windows version of GDB is more of a multi-language debugger than the older GDB on Mach (especially the older version of GDB available with previous releases of NEXTSTEP). This GDB has separate features for many different languages, including Objective-C. It attempts to guess the source language by looking at the extension of the source file name ("**.m**" or "**.M**" for Objective-C). To find out what GDB's ªcurrent languageº is, type **show language**. To force the current language to Objective-C, type **set language objective-c**.

Calling Methods from GDB

To call a method in your program from GDB, use the **print**, **set**, or **call** commands with an argument that looks just like a method call in Objective C, as shown here:

```
(gdb) print [myClass showValue: 12]
```

If the method comes from a Category, you must include the category name, like this:

```
(gdb) print [myClass(myCategory) showValue: 12]
```

Listing and Setting Breakpoints on Methods

To refer to a method in a **list** or **break** command, you can give the full class and method name, including a `+' or `-' to indicate a class method or instance method. If there is a category name, you must give that too:

```
(gdb) list +[myClass init]
(gdb) break -[myClass(myCategory) showValue]
```

You can also set breakpoints or list a method just by giving a selector. If the selector is implemented by more than one class, gdb will list the corresponding methods and ask you to choose one or more:

```
(gdb) break init
[0] cancel
[1] all
[2] -[Change init] at Change.m:20
[3] -[DrawApp init] at DrawApp.m:130
[4] -[Graphic init] at Graphic.m:139
>
```

You would then enter your choice or choices at the ">" prompt.

Getting Information about Classes and Methods

GDB for Windows now has the **info classes** and **info selectors** commands. These commands accept the same regular expression language as GDB's **info type** and **info function** commands (ie. the Unix style regular expression language). This is a change from the Mach gdb, where **info classes** and **info selectors** accept a slightly different regular expression language. For instance, to learn about class names beginning with NS (using the '^' character to designate ªbeginning withº):

```
(gdb) info classes ^NS
```

To learn about selectors, you can use the **info selectors** command. To find every selector containing the string "withObject:" you could enter:

```
 (gdb) info selector withObject:
```

To learn about methods, you can use the **info function** command, which also takes a regular expression. Since the square bracket characters `[' and `]' are significant in regular expressions, you can quote them with a backward slash to prevent their being treated as special characters. To list all the methods of a class, you might say:

```
 (gdb) info function \[MyClass
```

To list all the methods whose selector ends with ªcount:º, you might say:

```
(gdb) info function count:\]
```

If you want to know about a specific method of a specific class, but you are not sure if it belongs to a category, you could use the ª.*º wildcard sequence to stand for ªany number of any charactersº:

```
(gdb) info function MyClass.*mySelector:
```

Debugging Threads on Windows

The GDB for Windows is thread-aware. If you are debugging a multi-threaded program, you can use the **info threads** command to see a list of the currently active threads. Use the **thread** command to switch to one of the threads listed by **info threads**, giving it the number of the thread you want to debug (a small integer such as 1 or 2, not the thread ID).

When you switch threads, you will frequently find yourself in a non-debuggable area of code from which you cannot even get a backtrace. See the section on known problems, ªNon-Debuggable Codeº below.

Known Problems

Non-Debuggable Code

When you interrupt a program with ^C, attach to a running program, or switch threads in a running program, sometimes the program will be in the middle of executing Windows code that cannot be debugged. Often GDB cannot even give you a backtrace, because Windows has done something with the program stack. When this happens, if you don't want to simply let the program continue (for instance, you need to know exactly how you got to where you are), you can use the **stepi** command to step by single machine instructions until the Windows code cleans up the stack and returns, at which time you will suddenly be able to see symbols and backtraces again. GDB will notify you when you have returned to a symbolic region (say, a

function or a method) that it knows about. Usually this does not take too long; on the order of a few dozen instruction steps.

Notes Specific to GDB on PDO:

The **gdb** debugger for OPENSTEP Enterprise is based on the version 4.15.1 release from GNU. This brings with it many, if not most of the features of debugging on Mach, although there are inevitably some differences.

Debugging Objective-C: Differences from Mach gdb

The PDO version of **gdb** is more of a multi-language debugger than the older **gdb** on Mach (especially the older version of **gdb** available with previous releases of NEXTSTEP). This **gdb** has separate features for many different languages, including Objective-C. It attempts to guess the source language by looking at the extension of the source file name (**.m** or **.M** for Objective-C). To find out what **gdb**'s current language is, type **show language**. To force the current language to Objective-C, type **set language objective-c**.

Calling Methods from gdb

To call a method in your program from **gdb**, use the **print**, **set**, or **call** commands with an argument that looks just like a method call in Objective-C. For example:

```
(gdb) print [myClass showValue: 12]
```

If the method comes from a category, you must include the category name, as shown here:

```
(gdb) print [myClass(myCategory) showValue: 12]
```

Listing and Setting Breakpoints on Methods

To refer to a method in a **list** or **break** command, you can supply the full class and method name, including a + or - to indicate a class method or instance method. If there is a category name, you must give that too:

```
(gdb) list +[myClass init]
(gdb) break -[myClass(myCategory) showValue]
```

You can also set breakpoints or list a method just by giving a selector. If the selector is implemented by more than one class, **gdb** will list the corresponding methods and ask you to choose one or more:

```
(gdb) break init
[0] cancel
[1] all
[2] -[Change init] at Change.m:20
[3] -[DrawApp init] at DrawApp.m:130
[4] -[Graphic init] at Graphic.m:139
>
```

You would then enter your choice or choices at the ">" prompt.

Getting Information about Classes and Methods

gdb for PDO has the **info classes** and **info selectors** commands. These commands accept the same regular expression language as **gdb**'s **info type** and **info function** commands (that is, the UNIX-style regular expression language). This is a change from the Mach **gdb**, where **info classes** and **info selectors** accept a slightly different regular expression language. For instance, to learn about class names beginning with "NS" (using the "^" character to designate ªbeginning withº):

```
(gdb) info classes ^NS
```

To learn about selectors, you can use the **info selectors** command. To find every selector containing the string "withObject:" you could enter:

```
(gdb) info selector withObject:
```

To learn about methods, you can use the **info function** command, which also takes a regular expression. Since the square bracket characters ('[' and ']') have special significance in regular expressions, you can quote them with a backward slash to prevent their being treated as special characters. To list all the methods of a class, you might say:

```
(gdb) info function \[MyClass
```

To list all the methods whose selector ends with "count:", you might say:

```
(gdb) info function count:\]
```

If you want to know about a specific method of a specific class, but you aren't sure if it belongs to a

category, you can use the ".*" wildcard sequence to stand for any number of any characters:

```
(gdb) info function MyClass.*mySelector:
```

Known Problems

Problems with Underscores

gdb doesn't correctly handle methods for which either the class, the category, or the selector has underscores in the name, except when the underscore is the first character of the class or selector name.

Thread Support

gdb has no support for debugging multiple threads on either Solaris or HP-UX.

Random SIGTRAPs

gdb occasionally throws random SIGTRAPs. If you verify that the instruction at the $PC is not a trap instruction, you should be able to continue by saying "signal 0" (that's a zero). This tells **gdb** to continue the child without passing any signal to it.

Documentation

This file contains release notes for the Documentation distributed with OpenStep 4.1.

Locating the Developer Documentation

All of the documentation for OPENSTEP 4.1 can be found on-line (some of the documents are supplied in printed form, as well). The easiest way to access this documentation is to open the NextDeveloper bookshelf using Digital Librarian (this bookshelf can be found in **/NextLibrary/Bookshelves**). Although the NextDeveloper bookshelf can be used directly to access the documentation, most developers prefer to create their own custom bookshelf containing just those documents of interest. Select the first entry in the NextDeveloper bookshelf, then click ªList Titlesº; the documents that are then listed detail how to create your own custom bookshelf.

In NEXTSTEP Release 3.3, as well as in prior releases, all developer documentation was located in a subdirectory of **/NextDeveloper/Documentation/NextDev**. In OPENSTEP 4.1, framework documentation has been moved to a location inside of the relevant framework. All remaining developer documentation can still be found under **/NextDeveloper/Documentation/NextDev**.

In keeping with the philosophy that all framework resources be located within the framework, framework reference documents are located in the directory **Resources/English.lproj/Documentation**, relative to the ª**.framework**º directory. So, for instance, reference documentation for the Application Kit Framework can be found in **/NextLibrary/Frameworks/AppKit.framework/Resources/**

English.lproj/Documentation.

Release Notes

/NextLibrary/Documentation/NextDev/ReleaseNotes contains important information about the release that was known at the time that the OPENSTEP 4.1 CD-ROM was manufactured. Additional important information, which was discovered too late to be put onto the release, can be found on NeXTanswers. Request NeXTanswers #2455 for the location of the most up-to-date release notes for a variety of NeXT's software products.

Driver Development

OPENSTEP 4.1 cannot be used to develop Mach device drivers. Because NEXTSTEP 3.3 device drivers work on OPENSTEP 4.1 systems, we recommend that you use NEXTSTEP 3.3 Developer to create device drivers.

Interface Builder

This file contains release notes for Interface Builder distributed with OPENSTEP Release 4.1.

Notes Specific to Release 4.1

New Features

This release of Interface Builder supports these new features:

- *Context Menus*. Interface Builder on Windows NT has a context menus within editable windows and panels. A context menu is activated by clicking the right mouse button within a window or panel. In this way, operations such as Cut, Copy, and Group can be performed.

- *Interface Styles*. Interface Builder has a new preference which permits nib files to be viewed and edited in either Mach or Windows user interface style. Nib files opened after a change in the interface style preference will have the new interface style.

- *Control mnemonics.* Interface Builder enables users to set mnemonics for user-interface controls. In order to set a character in the title of a control to be a mnemonic, double-click the character while holding down the Alternate key.

- *OpenStep Compliance*. Interface Builder is OpenStep-compliant, and you can use it to create OpenStep nib files.

- *Custom Class Inspector*. You now use a new inspector display, the Custom Class Inspector, to specify a

custom subclass of an object. For example, suppose you have defined a class MyButton to be a subclass of NSButton. To add an object of class MyButton to your interface, you do the following:

1. Define MyButton in the Classes view of the nib file window.
2. Drag the button from the Views palette to your interface.
3. Choose Custom Class from the pop-up list in the Inspector panel.
4. Select MyButton in the browser to set the class of the button.

Currently, this works only for NSView and NSWindow subclasses.

- *Connection Locking*. Connection locking prevents connections from being deleted when a nib file is edited. This feature is designed primarily for localization. To turn connection locking on or off, use the Preferences panel.

- *Nib Dumping.* A new tool, **nibTool**, prints the contents of a nib file in an ASCII format. **nibTool** is located in **/usr/bin**. You can easily compare two versions of a nib file if you use this tool to create ASCII representations of each. Currently, **nibTool** does not support nib files that use objects from custom palettes.

- *Typed Outlets and Actions.* The parser in Interface Builder has been improved, and many parsing bugs have been fixed. In addition, you can now specify typed outlets. Interface Builder considers any of the following to be an outlet when it appears in an interface:

```
id outlet;
id outlet1, outlet2, ...;
IBOutlet SomeClass *outlet;
IBOutlet SomeClass *outlet1, *outlet2, ...;
```

You can also specify the type of the argument to an action, and you can specify the return type to of an action to be **void**. Interface Builder considers any of the following to be an action:

- `action:sender;`
- `(void)action:sender;`
- `action:(id)sender;`
- `(void)action:(id)sender;`
- `action:(`*SomeClass* `*)sender;`
- `(void)action:(`*SomeClass* `*)sender;`

The declarations of IBOutlet are in <AppKit/NSNibDeclarations.h>.

· IB supports the new NSComboBox object. It appears on the DataViews palette.

· The date and number formatters now live on the DataViews palette, not the EOF palette.

Known Problems

These problems exist in this release of Interface Builder:

· If you are running with 8-bit color, testing your interface may cause Interface Builder to crash. After entering test interface mode, if you exit test interface mode by clicking on the close box of the last open window, Interface Builder will be `hidden' - none of your document windows will reappear. At this point, you can restore Interface Builder to its active state by restarting it from the start menu. Unfortunately, Interface Builder will be in an unstable state, and may crash after a few operations. The workaround is to always exit test interface mode via the `Exit' menu item. (72599)

- After displaying the inspector the first time, deleting a menu item or submenu may cause Interface Builder to crash. This can happen even if you close the inspector and then delete a menu item. The workaround is to either delete the menu item before you display the inspector the first time, or to select the `Custom Class' inspector from the inspector popup before you delete the menu item. (72727)

- If you select BooksOnline from Interface Builder's Help menu on a system running Windows NT version 3.51, you'll discover that the links on the BooksOnline page appears to be broken. BooksOnline works correctly, however, if you invoke it from the OPENSTEP Enterprise program group.

- NEXTSTEP Release 2 nib files containing bitmaps unknown to the Application Kit are not loaded correctly. As a workaround, open the nib file in a Release 3 Interface Builder and save it before opening it in 4.1. For a list of bitmaps that the Application Kit supports, see the file **/NextLibrary/Frameworks/AppKit.framework/Headers/NSBitmapImageRep.h**.

- Interface Builder does not allow you to change the size of the document view when editing a ScrollView.

- The context-sensitive help links don't work for anything in the palettes window, and they don't work in the ScrollView Attributes display.

- Model files copied from a mail message are incorrectly referenced. In Interface Builder, if you drag a model file in from a mail message, it turns into an entity controller. Don't make a copy, and save the nib file. Then drag the model file from the mail message into **~/Library/Models**. Go back into the nib file and delete the combined controller/data source object. Then drag the model from **~/Library/Models** into Interface Builder. The resulting entity controller still references the model file in your mailbox.

Novell Netware

Novell Netware is not included in this release of OPENSTEP 4.1.

NEXTIME Framework

New Features in 4.0

OpenStep

The NEXTIME framework was introduced with release 4.0. It differs considerably from the architecture shipped with NEXTIME 1.0. The framework makes extensive use of the OPENSTEP Foundation classes.

NTMovieScreen

The new NTMovieScreen class provides a simple way to add movie support to applications. An

NTMovieScreen is an NSControl containing a NTMovieScreenCell in which you can display a NEXTIME movie. The user can then start and pause the movie's playback by clicking anywhere in the NTMovieScreen.

NTMovieScreenCell

NTMovieScreenCell is a subclass of NSActionCell for displaying the video data of a NEXTIME movie. Clicking the mouse within the cell's frame normally starts playback of the movie. If the movie is already playing, the click pauses the playback, so that the current video frame is displayed as a still image. The NTMovieScreenCell is normally associated with a NTMovieDocument or other object conforming to the NTMovieDocument and NTMovieDrawing protocols.

NTMovieControlView

The NTMovieControlView class provides a basic horizontal control bar to play, position, and step through movies. The view incorporates a pop-up volume control on systems with sound hardware. The class is intended to communicate with a NTMovieDocument or similar class.

NTMovieDocument

The NTMovieDocument manages a single movie, usually by displaying it in an NTMovieScreen and receiving control messages from an NTMovieControlView. Operations supported include play, pause, reset, fast forward or reverse, and step forward or backward. Notifications from the NTMovieDocument report changes in the movie's playback status and its position in time. The class provides support for editing movies from

the pasteboard.

/usr/etc/showmacresources

The **showmacresources** program slightly modifies the state of the loaded Macintosh filesystem to enable or disable reporting of resource forks by the functions used to report directory contents. NEXTIME users can use this program to make finding and copying Macintosh Quicktime movie resource and data forks much easier.

showmacresources
showmacresources +
Enable reporting of resource forks on disks mounted after the command is run.

showmacresources -
Disable reporting of resource forks on disks mounted after the command is run.

The Macintosh filesystem must be loaded prior to running **showmacresources**. There are two ways to ensure that the Mac filesystem is loaded.

1) You can load the Macintosh filesystem by inserting a floppy. Simply insert a Macintosh CDROM or floppy, and, on non-NeXT hardware, click on the ªDisk->Check for Disksº Workspace menu item. Once the Workspace has found and displayed the disk, eject it. The Macintosh file system is now loaded.

2) You can load the Macintosh filesystem from a shell. To do so, run the following command as root:

```
kl_util -a /usr/filesystems/mac.fs/macfs_reloc
```

The Macintosh file system will be loaded.

If you always want to see resource forks for Macintosh files, you can modify your **/etc/rc.local** file to always load the Macintosh file system and modify it to show resource forks:

```
#
# Load Macintosh filesystem and modify to report resource forks
#
if [ -f /usr/filesystems/mac.fs/macfs_reloc -a -f /usr/etc/showmacresources ]; then
        /usr/etc/kl_util -a /usr/filesystems/mac.fs/macfs_reloc
        /usr/etc/showmacresources +
fi
```

Known Problems in This Release

Cursor during Movie Playback

If the cursor is placed in the movie screen area during movie playback, the cursor will disappear until moved again. When the cursor is moved across the playback area while the movie is playing, it may leave behind small squares of ªstaleº movie images which will disappear when the next key frame is drawn (usually within 1 second).

PDO

Known Problems in This Release

This section documents some of the known problems with this release of PDO. Known problems with specific portions of PDO (such as Foundation, the compiler, and so on) can be found in their own release note files.

Reference: 66188

Problem: Redefinition of MIN and MAX on HP/UX generates a warning

Description: On HP/UX, **param.h** doesn't guard against the redefinition of MIN and MAX. Since **NSString.h** indirectly includes **param.h**, if you've already imported **NSObjCRuntime.h** (which defines MIN and MAX, properly guarded against redefinition), you get a warning when **param.h** overrides these.

Workaround: Edit the copy of **param.h** that's located in **NextDeveloper/lib/gcc-lib/hppa1.1-nextpdo-hpux/2.7.2/include/sys** (relative to the location where PDO is installed), and place a guard around the MIN and MAX defines, as shown here:

```
#ifndef MIN
#define MIN(a,b) (((a)<(b))?(a):(b))
#define MAX(a,b) (((a)>(b))?(a):(b))
#endif
```

Reference: 68675

Problem: Hang while looking up connection after fork

Description: If you have a DO client which tries to look up a connection using NSConnection's **connectionWithRegisteredName:host:**, then forks a process, and then either exits or tries to look up the connection again, the client will hang. If the fork is omitted, the client won't hang.

Workaround: None.

Reference: 69152

Problem: HP-UX nmserver has problems when passing large structures

Description: On HP-UX systems, passing large structures over DO without having first defined a protocol can cause the nmserver to become non-functional.

Workaround: Use DO protocols when passing large structures over DO on HP-UX.

Reference: 69056

Problem: Multi-threaded DO example occasionally hangs on HP-UX

Description: Compiling and running the multi-threaded DO example in **/NextDeveloper/Examples/Foundation/MultiThreadedDO** occasionally results in a hang.

Workaround: None.

Deviations from the OpenStep Specification

The following items are known departures from the OpenStep specification:

- **[NSString stringByStandardizingPath]** raises when presented an absolute path ending in "**..**".
- PDO's implementation of dynamic loading of code from bundles doesn't work on either HP-UX or Solaris. This includes related functionality such as **+bundleForClass:**.

Printing

The following notes relate to printing, and augment those found in the Application Kit Release Notes.

Known Problems in This Release

Reference: 59276

Problem: Default printer in NEXTSTEP 3.3 vs. OpenStep apps

Description: Different default mechanisms are used for NEXTSTEP 3.3 and OpenStep 4.*x* applications. As a result, if you select a printer in TextEdit, it will not be the default printer in Edit.

Windows-Specific Notes

Third-party software is required to print to non-PostScript printers from OPENSTEP Enterprise applications.

Project Builder

This file contains release notes for the Project Builder application distributed with OpenStep 4.1. It includes information specific to OPENSTEP for Mach and to OPENSTEP for Windows as well as information general to both platforms.

Project Builder has undergone a radical redesign since NEXTSTEP 3.3. Its main window integrates a project browser with a full-featured code editor. You can bring up panels for building, debugging, finding information in the project, and setting project attributes. To familiarize yourself with the new Project Builder's capabilities and features, you can do one of the following.

- On Mach, choose the Help command from Project Builder's Info menu. This launches a Digital Librarian bookshelf containing the on-line book *OpenStep Development: Tools and Techniques*.

- On Windows, choose the How To command from Project Builder's Help menu. This launches a Windows

Help version of the same book.

New Features for 4.1

New Makefiles

Project Builder includes new makefiles that offer several advantages over the previous ones. They:

- Build projects with lower overhead.
- Fix a number of extensibility problems.
- Allow a higher degree of parallelism with **gnumake -j**.
- Make it easier to diagnose problems.

When you first open a project that uses the old makefiles, Project Builder displays a panel that lets you :

- Convert immediately to the new makefiles.
- Keep your current makefiles.
- Defer the decision to later.

If you convert to the new makefiles, the project defines this path to them in MAKEFILEDIR:

```
$NEXT_ROOT/NextDeveloper/Makefiles/pb_makefiles
```

The original 4.0 Makefiles are located in this directory:

```
$NEXT_ROOT/NextDeveloper/Makefiles/project
```

Currently, you can override the MAKEFILEDIR attribute for your project in **Makefile.preamble** by setting it to the desired directory. Or you can edit the MAKEFILEDIR attribute in **PB.project**, but not through the user interface of Project Builder. You are strongly encouraged to convert your projects to the new makefiles if that is feasible. For instance, a new facility that automatically generates **.def** files (see below) depends on these makefiles. However, you might keep older makefiles intact for those exceptional projects that depend upon unique implementation details in the project makefiles.

Microsoft Linker

The Microsoft linker is now the default for projects on Windows built with Project Builder. This is the same linker that ships with Visual C++. For information on using the linker, consult the Visual C++ documentation. See the following item, "Automatic Generation of .def Files," for a related issue.

Automatic Generation of .def Files

On Windows, the Microsoft linker requires symbols exported from a framework DLL to be explictly specified. One way of doing this is in a .**def** file in the project (for example, "MyFramework.def"). In prior releases, this file had to be manually created. Project Builder now generates a **.def** file for frameworks and dynamic libraries (that is, library projects producing DLLs). It re-creates the file the first time a project is built and, thereafter, updates it every time an object file changes. To get this feature, you must update to the new makefiles (See "New Makefiles," above).

If you currently have a **.def** file for a framework, and it is not highly customized, delete it and let Project regenerate it for you. Otherwise, leave it where it is. Project Builder creates a **.def** file only if there isn't one already in the project.

The **.def** file that Project Builder creates is comprehensive, so if you want to create a customized .**def** file that is a subset of it, first build your project. Then add the generated **.def** file in **derived_src** to Supporting Files in your project and edit it as necessary.

Note 1: You should use the CONSTANT keyword instead DATA in your **.def** file. The export of framework functions also requires a little extra code. For more on both of these items, see "Known Problems in this Release, " below.

Note 2: If you have prerelease versions of OPENSTEP 4.1 in which you used **Makefile.preamble-winnt** and **Makefile.postamble-winnt** for manually generating **.def** files, you can delete these files and now use Project Builder's automatic **.def** generation facility. Remember to remove all references to items involved in manual **.def** generation, particularly where your Makefiles include the platform-specific makefiles. Do this by disabling the option in your respective pûeamble and postamble files. For example:

```
#-include Makefile.preamble-$(PLATFORM_OS)
```

Specifying Installation Directories for Dynamic Libraries

On Windows, the makefiles for dynamic-library projects (that is, projects with "LIBRARY_STYLE = DYNAMIC") put the two files generated by the project in different locations. The **.dll** file is installed in

INSTALLDIR and the **.lib** file is installed in the location for libraries that are imported into executable. You can specify this latter location in **Makefile.preamble** by setting IMPORT_LIBRARY_DIR to the correct path.

New Features for 4.0

Some or all of these new features may exist in the Windows version of Project Buiilder depending on its state at the time of the release:

- The new Launcher Panel allows you to run or debug executables built by the project.
- Printing is now supported.
- The Build Attributes Inspector allows you to specify OS specific values for compiler flags, linker flags, installation and build directories, and build tool.
- The Build Options panel values persist across invocations of Project Builder. The values you specify on this panel are saved per-user, per-project.
- In the Project Attributes Inspector for an application project, you can specify a help file and icons for both NT (**.ico**, **.ICO**) and Mach (**.eps**, **.tiff**).
- Project Builder by default sets tab stops at regular intervals of eight spaces each. There is now a defaults write, ªtabStopCharsº, that allows you alter this interval to any desired number of spaces, as long as that number is greater than zero.

4.1 Conversion Note

The _**main.m** files of 3.*x* applications have to be manually converted to the new _**main.m** content. To do this, copy the **main()** code from existing 4.1 projects or create a new project and copy its **main()** code. You can also copy and paste this code directly from here:

```
#import <AppKit/AppKit.h>

int main(int argc, const char *argv[]) {
   return NSApplicationMain(argc, argv);
}
```

Known Problems in This Release Ð Windows

These problems are known to exist with this release on Windows:

Reference: 72980

Problem: Automatic **.def** file generation doesn't handle subprojects.

Description: The Project Builder feature that generates **.def** files only inspects the object (**.o**) files at the top level and overlooks the object files in subprojects. Complex projectsÐthose with one or more subprojectsÐwill therefore not build correctly. To work around this problem, add the following code

to the **Makefile.postamble** of the library or framework.

```
ifeq "WINDOWS" "$(OS)"
    $(WINDOWS_DEF_FILE): $(OFILES) $(OTHER_OFILES)
    $(SILENT) if [ ! -r "./$(WINDOWS_DEF_FILE)" ] ; then \
        $(ECHO) -n "Generating $(notdir $@)...." ; \
        (cd $(OFILE_DIR); $(DUMP_SYMBOLS) $(OFILES)            $(OTHER_OFILES) `$(CAT)
$(NAME).ofileList` | $(EGREP)
    "(SECT).*(External).*(\|)" | $(EGREP) -v "(__GLOBAL_$I)|(_OBJC_)"    | $(SED) "s/
_/ /" > $(TMPFILE)) ; \
        $(ECHO) "LIBRARY $(NAME).dll" > $(GENERATED_DEF_FILE) ; \
        $(ECHO) "EXPORTS" >> $(GENERATED_DEF_FILE) ; \
        $(CAT) $(TMPFILE) | $(EGREP) "(SECT2)|(.objc_c)" | $(AWK)   '{printf "\t%s
CONSTANT\n", $$NF}' >> $(GENERATED_DEF_FILE) ; \
        $(CAT) $(TMPFILE) | $(EGREP) -v "(SECT2)|(.objc_c)" | $(AWK) '{printf
"\t%s\n", $NF}' >> $(GENERATED_DEF_FILE) ; \
        $(RM) $(TMPFILE) ; \
        $(ECHO) "done"; \
fi
endif
```

Reference: 72815

Problem: Multiple Control-N's in rapid succession causes Project Builder to lock up

Description: Typing a series of Control-N characters one after the other causes Project Builder to display the Open Project panel. Clicking Cancel causes Project Builder to freeze. At this point, it can only be killed from Windows NT's Task Manager.

Reference: 72715

Problem: BooksOnline doesn't work when accessed from Project Builder on NT 3.51

Description: If you select BooksOnline from ProjectBuilder's Help menu on a system running Windows NT version 3.51, you'll discover that the links on the BooksOnline page appears to be broken. BooksOnline works correctly, however, if you invoke it from the OPENSTEP Enterprise program group.

Workaround: Access BooksOnline from the OPENSTEP Enterprise program group.

Reference: 72554

Problem: Source code versioning is error-prone with derived project files.

Description: Sometimes it might be difficult to keep derived project files (**Makefile** and *app***.iconheader**) and original project files (**PB.project**) synchronized in source-control situations. You can face a similar problem when you modify **Makefile** separately from **PB.project** and need to "back out" your changes.

Project Builder contains in its Resources directory a command-line tool that generates derived files from **PB.project**. The proper invocation of this tool, called FileWriter, is:

```
/NextDeveloper/Apps/ProjectBuilder.app/Resources/FileWriter -project [path/]PB.project
-makefile [path/]Makefile [-iconheader iconHeaderFile]
```

Note that the **-iconheader** option is meaningful only on Mach.

Reference: 72542

Problem: LIBRARY_PATHS makefile variable not passed to linker

Description: If you add libraries to a project using Project Builder's Project->Add Files command, or if you add entries to the Library Search Order list in the project inspector, the LIBRARY_PATHS variable is created in the Makefile. However, this variable is not passed to the linker.

To work around this problem, assign the following value to the OTHER_LDFLAGS variable in **Makefile.preamble**:

```
OTHER_LDFLAGS = $(LIBRARY_PATHS)
```

Reference: 72344

Problem: Installation of **.lib** file in static library projects

Description: In static library projects (LIBRARY_STYLE=STATIC), the **.lib** file gets installed twice, once in the INSTALLDIR and once in the location specified by IMPORT_LIBRARY_DIR in **Makefile.preamble**. To have the **.lib** file installed only in the desired location (INSTALLDIR), set IMPORT_LIBRARY_DIR to INSTALLDIR.

Reference: 71975

Problem: The **libtool** utility does not recognize the **-read_only_relocs** flag.

Description: A serious side effect of this bug is that this option causes **libtool** to ignore all subsequent flags.

Currently there is no workaround except for not using this flag.

Reference: 69061

Problem: Builds will fail if NEXT_ROOT has been modified to contain backward slashes.

Description: The installer specifies NEXT_ROOT with forward slashes, for example:

```
C:/NeXT/
```

Do not edit this variable in the Environment section of the System control panel, even if to change the forward slashes to backward slashes.

Reference: 72307

Problem: The Microsoft linker emits a bogus warning about CONSTANT keywords.

Description: When you build framework projects, you will see messages complaining about the use of CONSTANT keywords in your **.def** file. Because of problems with the linker, CONSTANT should be used in place of DATA. You can ignore the warnings.

Reference: 72308

Problem: Automatically generated **.def** files do not include public exported functions and variables.

Description: In creating the **.def** file for frameworks and dynamic libraries (DLLs), Project Builder handles Objective-C symbols and data but not functions or variables. If you have functions in your framework

that need to be callable from the outside, or variables that should be externally accessible, define certain macros in a header file and then use them in your function declarations. These macros allow you to export non-static functions and variables from the DLL (*FRAMEWORK*_EXTERN) or to declare them as **extern**, but not exported (*FRAMEWORK*_PRIVATE_EXTERN).

Make a header file for each framework with functions and variables you wish to export (*Framework*Defines.h), and in it write the following macros (substitute your own framework name for *FRAMEWORK*):

```
#ifndef _FRAMEWORKDEFINES_H
#define _FRAMEWORKDEFINES_H
#endif

#if defined(WIN32)

//
// For Windows
//

#ifndef _FRAMEWORK_BUILDING_DLL
#define _FRAMEWORK_WINDOWS_DLL __declspec(dllimport)
#else
#define _FRAMEWORK_WINDOWS_DLL __declspec(dllexport)
#endif

#ifdef __cplusplus
#define FRAMEWORK_EXTERN _FRAMEWORK_WINDOWS_DLL extern "C"
#define FRAMEWORK_PRIVATE_EXTERN extern "C"
#else
```

```
#define FRAMEWORK_EXTERN _FRAMEWORK_WINDOWS_DLL extern
#define FRAMEWORK_PRIVATE_EXTERN extern
#endif

#else

//
// For MACH and PDO
//

#ifdef __cplusplus
// This isnt extern "C" because the compiler will not
// allow this if it has seen an extern "Objective-C"
#define FRAMEWORK_EXTERN extern
#define FRAMEWORK_PRIVATE_EXTERN __private_extern__
#else
#define FRAMEWORK_EXTERN extern
#define FRAMEWORK_PRIVATE_EXTERN __private_extern__
#endif

#endif
```

You need to import this header in all your other headers and declare your function prototypes accordingly. Then, before building your framework, specify "-D_*FRAMEWORK*_BUILDING_DLL" in the Compiler Flags field of the Build Attributes inspector.

Additionally on Windows, Project Indexing doesn't work and, because of this, the ªDefinitionsº and ªReferencesº searches in the Project Find Panel don't work either.

Known Problems in This Release Ð Mach and Windows

Other problems with this release on both Mach and Windows are the following:

- When Escape is defined as the Meta key in Preferences (and Tab is bound to indentation), Escape-/ (slash) does not cycle through all possible expansions (50288).
- Queries in the Project Find panel for macro references do not find existing uses of macros (55187).
- Comparing autosaved and last-saved versions of files in FileMerge does not work (56892).
- Header files in Framework projects do not become public by default (56924).
- Renaming a subproject doesn't rename the directory the subproject lives in (57211).
- You cannot switch to a second view when no files have been loaded into the code editor (57698). To workaround this problem, after selecting a text file in the upper browser, type Command-1 followed by Command-2. This switches back to a single view, and then splits it again so it can double the text file in the upper view.
- Indexing generates an error if a header file is empty (57790).
- Build errors in tear-off windows appear in the main editing view (58863).

- An exception may be raised when running a new executable (that is, after selecting a different executable from the Launcher Options panel) with no arguments selected when the previous executable had arguments. The workaround is to deselect the argments from the old executable before selecting the new one.

- Breakpoints on functions or methods without debugging information aren't displayed in the breakpoint inspector. (67785)

- Moving a breakpoint icon doesn't update the line number in the Breakpoint inspector. However, the location of the breakpoint is updated in **gdb**. (67712).

TextEdit

TextEdit is a new application based on the new text system. In addition to the standard text editing features in the 3.3 Edit application, it provides:

- "Wrap to Page" mode. This mode causes the document to be wrapped to the current page size (as opposed to the window size), and thus shows you a simple WYSIWYG representation of your pages. This mode also allows zooming.

- Support for new text object features such as kerning, ligatures, and baseline offsets.

- Ability to read/write plain text files in different 8-bit character encodings and Unicode. Coupled with the Kanji support in the new text object, you can read/write/edit Kanji files created by 3.3J or 4.1J. (You will need Kanji fonts and input manager.)

  The alternate encoding to apply to a file is specified by a popup in the Open or Save panels. The default item, "Default," tells TextEdit to use either the default 8-bit encoding of the system or Unicode. Other choices let you select different encodings. You can also set the default setting of this popup via the Preferences panel.

Sources to TextEdit are available with the developer software, in **/NextDeveloper/Examples/AppKit/TextEdit**.

Known Problems in This Release

Reference: 58895

Problem: Tabs in plain text mode

Description: Tabs in a plain text window are by default set to a multiple of the font width. However this setting isn't readjusted when the font is changed, so the tabs get out of sync.

Workaround: Either don't change the font of a plain text window on the fly, or change it via defaults and restart the application.

Reference: 58741

Problem: Displaying unrenderable characters too slow

Description: When given characters which are not valid for the current font, the text system might spend too much time trying to find a font to render them. This means that files with random binary data could take a long time to load.

Workaround: Avoid binary files. A few binary characters are probably fine, and will be displayed as boxes, but a total binary file (for instance, an executable), could keep TextEdit busy for a while.

Reference: None

Problem: Encoding popup in Open/Save panels not available under NT3.51

Description: Because accessory views aren't supported under NT3.51, there is no encoding popup in the Open/Save panels.

Workaround: You can use the popup in the Preferences panel to set the default choice for the popup, which will then be used when files are opened or saved.

In addition, please refer to the Application Kit release notes for the new text system bugs.

TrueType Fonts

This file contains release notes for the 4.1 TrueType font support. TrueType font support was introduced with

the 4.0 release.

ttf2font

True Type font files can be converted to font files which are usable by the OpenStep Window Server and the Application Kit. The **ttf2font** command line utility converts a TrueType font file to a font directory which contains the Postscript Type 42 font definition as well as the Adobe Font Metrics file. The command line syntax is as follows

ttf2font [-o *outputPath*] *trueTypeFont.ttf*

where the **-o** option is used to identify the directory where **.font** files will be placed

As an example, the following command line converts the TrueType font file TIMES.TTF and places the resulting **.font** directory into the **~/Library/Fonts** directory using standard encoding:

ttf2font -o ~/Library/Fonts TIMES.TTF

Advanced options are also available and are as follows:

-v Verbose mode. All errors and messages are printed to standard output.

-e *encoding* Use a specific encoding scheme, where *encoding* is one of:

nextstep	Nextstep encoding
standard	Abobe standard encoding
isolatin1	ISO latin 1 encoding

symbol Symbol encoding
font Font-specific encoding

Encoding schemes stored within the font file are font-specific encoding schemes. The font vendor is responsible embeding a suitable encoding scheme for your platform.

Specifiying an encoding scheme other than **font**, causes the program to ignore the font specific encoding schemes stored within the file.

-p *platformID*
-s *platformSpecificID*

Within a TrueType font file there are several tables which contain font specific encoding schemes. These encoding schemes are organized by platform; there can be more that one encoding scheme stored within a file, each identifying a different platform.

These options allow you to override the default platform. The *platformID* and *platformSpecificID* pair must identify a valid encoding scheme store within the file. The **-v** option allows you to observe the platform encoding schemes stored within the file. See *The TrueType Font Format Specification* for more details.

If no encoding option is specificed and the font does not contain alpha-numeric characters, the best font-specific encoding scheme is used. Otherwise, the default encoding is **standard** encoding.

If no encoding schemes yield any glyph-to-character mappings, then all glyphs are linearly mapped to character codes 32 through 255.

Known Problems in This Release

- Printing of TrueType fonts can only performed on printers with Postscript interpreter versions greater that 2013 which have Type 42 font support.

- The TrueType fonts MS Line Draw and Symbol MT do not work.

Window Server

Notes Specific to Release 4.0

New Features in 4.0 (General)

A new release of DPS from Adobe (2015) has been integrated.

Better Font Support

This release includes support for CID fonts and TrueType (type 42) fonts. You can use the **ttf2font** conversion program to convert TTF files to a format suitable for use with OPENSTEP. (see TrueType release notes for details).

New Features in 4.0 (Mach Only)

Native 16-bit color support

We have implemented native support for RGB:555/16 framebuffers on Intel platforms . In previous releases, rendering to these framebuffers actually only used 4 bits per channel of precision (12 bits total). This new rendering implementation gives much improved color fidelity on RGB:555/16 displays. The only difference developers might notice is that **readimage** will now return 24 bit images from RGB:555/16 displays.

Backing Store Compression

The WindowServer now uses compression to conserve memory for window backing stores. Windows which have not been recently referenced are compressed. This leads to a slight increase in CPU usage by the WindowServer with the gain of significantly reducing memory requirements for window backing stores.

NXHosting

NXHosting is supported from 3.3 applications to the 4.0 WindowServer. NXHosting from 4.0 to 3.3 is not supported.

New Features in 4.0 (Window NT Only)

On Windows NT the WindowServer plays a different role in OPENSTEP than it does under Mach. It is not involved in Window region management or event distribution. Instead, it serves as a "drawing server" which interprets PostScript and provides backing stores for Windows windows. OPENSTEP applications still

rendezvous with the WindowServer to do their drawing, but instead of creating windows via the WindowServer, they use WIN32 to create windows directly. The application then passes the Windows window number to the WindowServer and a backing store is created. All drawing is occurs in the WindowServer, and when **flushgraphics** is called, the backing store is transferred to the display via GDI's BitBlt.

NXHosting

NXHosting is not supported (inbound or outbound) on Windows NT.

Pixel Formats

We support drawing into the following DIB formats for backing stores:

8bit color (includes System Palette entries)
555 16bit color
565 16bit color
32 bit color

In Windows 16 color VGA mode, we use 8bit color backing stores. This mode is supported for diagnostics, but is not recommended and has some really bad color artifacts.

In Windows 32bit color mode, our rendering code currently does not gamma correct. This means that OPENSTEP applications will look dark, unless you display adaptor supports gamma correction in hardware.