Title: OPENSTEP 4.2 Release Notes
Entry Number: 2504
Last Updated: May 30, 1997

OPENSTEP 4.2 Developer Release Notes: An Introduction

With every release of its software, NeXT provides notes containing information pertaining to the release. These notes apply to Releases 4.2 and 4.1, and in some cases to earlier releases as well.

These on-line Release Notes list new features, fixed bugs, and known bugs that will be of interest to developers using OPENSTEP™ for Mach, OPENSTEP Enterprise, and PDO. These notes incorporate the latest known information about the release, and are thus more current than those found on the OPENSTEP Enterprise CD-ROM.

About NeXTanswers

NeXTanswers is an automated retrieval system that gives customers access to the latest product information, technical documents, drivers, and other software. You can access NeXTanswers through NeXT's website (**http://www.next.com**), or by:

- Electronic mail: Send requests to **nextanswers@next.com** with a subject line of HELP to receive instructions on how to proceed.

- Fax: Call (415) 780-3990 from a touch-tone phone and follow instructions (you'll need to know the ID numbers of the files you want).
- Anonymous **ftp**: Connect to **ftp.next.com** and read **pub/NeXTanswers/README** for further instructions.
- Modem: Call (415) 780-2965, log in as "guest", and go to the Files section. From there you can download NeXTanswers documents.

About This Release

Release 4.2 is primarily a bug-fix release. In addition, numerous fixes and enhancements have been made to the development tools, and the compilers have been synched-up so that they use the same source base across NeXT's entire product line (OPENSTEP for Mach, OPENSTEP Enterprise, and PDO). Release 4.2 also supports Windows 95 as a deployment platform (debugging is supported, but compilation is not).

The various files containing release notes for developers are listed below. These files are included in the ªDeveloper Release Notesº target of the NEXTSTEP Developer bookshelf (in the directory **/NextLibrary/Bookshelves**), and are located in **/NextLibrary/Documentation/NextDev/ReleaseNotes**.

This file is divided into the following major sections:

Section	**Subject of Notes**
Installation	Important notes regarding product installation
Compiler	The C, Objective-C, and C++ compilers
CompilerTools	Assembler, Linker, and other compiler-related tools

Converting Your Code	Converting your code from NEXTSTEP to OPENSTEP
Debugging	The GNU Source-level Debugger
Documentation	Locating on-line documentation
DriverDevelopment	Developing device drivers
D'OLE	The D'OLE ORB
EnterpriseObjects	The Enterprise Objects Framework, Release 2.1
Netware	Novell NetWare
ProjectBuilderSCM	Software Configuration Management extensions to Project Builder
Supplemental User Notes	Notes applicable to deployment systems as well as development systems
TextEdit	The TextEdit application

The following topics are release noted in separate NeXTanswers:

Section	**Subject of Notes**
AppKit	The Application Kit framework
FoundationPre40	Describes API changes that were made between OPENSTEP 4.0 and earlier releases. This note will be of interest only if you are upgrading an application to this release from a release of OPENSTEP, PDO, or D'OLE prior to 4.0.
Foundation	The Foundation Framework
ProjectBuilder	Project Builder

Installation

Upgrading from OPENSTEP 4.1 for Mach

Certain hardware configurations appear to cause upgrader to fail (installation hangs, and OPENSTEP is corrupted) when upgrading an OPENSTEP 4.1 system to OPENSTEP 4.2. This problem appears to occur on systems that have EIDE CD-ROM or hard drives, and affects upgrades performed either from a CD-ROM or over a network. It appears that you can work around this problem by placing a disk in the CD-ROM drive before attempting the upgrade.

If you encounter this problem, you'll need to install OPENSTEP 4.2 from scratch.

This problem does not occur when upgrading from NEXTSTEP 3.3.

Installing on Windows 95

While this release of OPENSTEP Enterprise now supports deployment on computers running Windows, 95, these same computers cannot be used to develop OPENSTEP software. (OPENSTEP software developed on Windows NT can be deployed on Windows 95.)

See the *OPENSTEP Enterprise Installation Guide* for instructions on installing the deployment packages (located in the OEDeploy directory on your *OPENSTEP Enterprise 4.2* CD-ROM) on computers running either Windows 95 or Windows NT.

PDO Installation

Due to a possible bug in the SPARC's quad-precision emulation software, this release cannot be used on SPARCstation 5-class machines.

Installing on Japanese Language Systems

Attempting to do a ªtypicalº installation of the NT Server Deployment (OEMinSys) package on a Japanese language system will cause the installer to enter an infinite loop. If you need to install this package on a Japanese language system, choose "custom" when asked whether you want a ªtypicalº or ªcustomº installation. When presented with the components to install, de-select ªIMEº and proceed with the installation.

This problem doesn't arise on non-Japanese systems; nor does it arise when installing either the NT Deployment (OEDeploy) or NT Development (OEDev) packages.

Known Problems in This Release

This section lists important problems that were discovered after the CD-ROM was duplicated.

Building Bundles With PDO on Solaris

Although NSBundles now work in PDO 4.2 on Solaris, due to a makefile problem bundles don't build properly (an error occurs when linking). This problem can be fixed by updating **bundle.make** (in **/NextDeveloper/Makefiles/pb_makefiles**) and **libtool** (in **/NextDeveloper/bin**). New versions of these files can be obtained through NeXTanswers; request NeXTanswer #2499 for the updated files.

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.

Compiler Tools

This file contains release notes for the 4.2 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 4.2

New Features

There are no new features for the 4.2 release of the compiler tools.

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

Converting Your Code to OPENSTEP

This document describes how to convert your code from NEXTSTEP Release 3.*x* to OPENSTEP for Mach Release 4.*x*.

Release 4.0 is NeXT's first OpenStep-compliant release. OpenStep is an API that enables platform-independent development of client/server applications. The OpenStep API includes the Application Kit, the DPSClient library, and a new kit called the Foundation Framework, which provides an operating system independence layer. The OpenStep Application Kit is functionally equivalent to the NEXTSTEP Release 3 Application Kit, but its API has been reworked to make use of the Foundation Framework. Because Release 4.0 is OpenStep compliant, converting your code to it is a good way to make your application an OpenStep application.

To convert your code, you run a series of scripts. These scripts use **tops**, a tool that performs in-place substitutions on source files according to a set of rules. The script files contain the rules that **tops** applies to your code. Most of the scripts are provided in the release, but you must generate some of them before you start converting because they work directly on the custom classes in your

code. OPENSTEP for Mach provides a tool that allows you to generate these scripts.

The scripts convert most of the NEXTSTEP API to the new OpenStep API. Some methods, classes, and functions have been altered in such a way that an automated conversion is not possible. For these, the conversion scripts produce an error message that identifies the obsolete code and tells you how to convert it.

You run the conversion in six stages. Each stage runs a different set of scripts. After each stage, you compile your code to identify places where the conversion was not automatic. It is recommended that you run some additional scripts to replace the Common classes with OpenStep API, making your code even more portable.

This document tells you how to set up your project for conversion and how to run the conversion scripts. A separate document, the *OpenStep Conversion Guide,* describes the changes made during each of the conversion stages and the reason for those changes. Its location on-line is **/NextLibrary/Documentation/NextDev/Conversion/ConversionGuide**. Read the first chapter of this guide (**00_Intro.rtfd**) for an overview of the differences between NEXTSTEP 3.X and OpenStep and for a discussion of the different strategies you might use when converting.

The Conversion Process

To convert your code, do the following:

1. **Convert your project to a 4.0 project.**
2. **Generate the conversion scripts.**
3. **Run the six-stage conversion process and any optional conversions.**
4. **Convert your nib files.**
5. **Debug your application.**

These steps are described further below.

Converting Your Project to a 4.*x* Project

Before you start the conversion process, you need to change your project and makefiles so that they use the new Release 4.0 development environment. The 4.0 development environment has many significant improvements over the 3.3 development environment. In particular, Project Builder has changed significantly. To convert your project, perform the following steps:

1. Read *OPENSTEP Development: Tools and Techniques* and the release notes for Project Builder and Interface Builder to learn about changes to the environment.

   The document you are reading now does not describe how to use the new Project Builder. The book *OPENSTEP Development: Tools and Techniques* is on-line in the directory **/NextLibrary/Documentation/NextDev/TasksAndConcepts/DevGuide**. The release notes are in the directory **/NextLibrary/Documentation/NextDev/ReleaseNotes**.

2. Make sure your code compiles cleanly without warnings.

   Perform this step so that any warnings that show up during conversion won't become mixed in with pre-existing warnings. Make sure the **-Wall** compiler option is being used.

   Before you compile your 3.3 code on a 4.0 system, you need to change the search path for header files to the directory **/NextDeveloper/OpenStepConversion/3.3Headers**. To do this in the new Project Builder:

   1. Choose Inspector from the Tools menu to bring up the project inspector.

2. From the inspector pop-up list, choose Build Attributes.
3. Choose Header Search Order from the pop-up list in the Build Attributes inspector.
4. Type **/NextDeveloper/OpenStepConversion/3.3Headers** and click Add.

3. Back up your project directory.

   The conversion scripts modify your code in-place, and they do not create backups for you. It is strongly recommended that you back up your project after each conversion stage in addition to backing up before you begin, provided you have enough space.

4. Delete the NeXT-provided libraries (such as **libNext**) from your project, and add the corresponding frameworks (such as **/NextLibrary/Frameworks/AppKit.framework**).

   All NeXT-provided libraries are replaced with frameworks in 4.0. A frameworks' executable code is a dynamic shared library. To add the Application Kit framework, in Project Builder choose Add File from the Project menu, select **/NextLibrary/Frameworks/AppKit.framework** from the Add File panel, and choose Frameworks from the file type pop-up list on the Add File panel. Add the Foundation Framework in the same way.

   Frameworks are bundled differently than shared libraries. All of the support files for a framework are contained in the same directory, so you no longer have to know that the library itself resides in one place, its header files another place, and its documentation still a third place.

   When you add a framework to your project, it doesn't appear in the Libraries suitcase. Instead, it appears in the Frameworks suitcase. You are able to see that framework's header files and documentation underneath that suitcase.

5. Change the application class in Project Builder.

   In OpenStep, all keywords are prefixed with "NS", so the Application class is now NSApplication. If you are converting an application project, you need to change the class in Project Builder so that it will use the appropriate class when updating your **main** function. To do this, bring up the project

inspector by choosing Inspector from the Tools menu. In the inspector, choose Project Attributes. Finally, type NSApplication in the field labelled Application Class.

6. Convert your makefiles.

   The new Project Builder comes with new **Makefile.postamble** and **Makefile.preamble** files. Convert your preamble and postamble files if you performed some customization on them. You can find copies of the new templates in the directory **/NextDeveloper/Makefiles/project**. Rename your existing preamble and postamble files, copy the new templates into your project directory, rename the template (remove the **.template** extension), and merge any customizations you would like to keep into the new templates.

   Most of the changes to the makefiles are additions made to support frameworks, however some makefile variables are now obsolete. The additions are documented in the comments in the preamble and postamble files. The following table lists the obsolete makefile variables and what you should use as a replacement.

Obsolete Makefile Variable	**Possible Replacement**
BUNDLELDFLAGS	OTHER_LDFLAGS
PALLETTELDFLAGS	OTHER_LDFLAGS
COMMON_CFLAGS	OPTIMIZATION_CFLAG, WARNING_CFLAGS
NORMAL_CFLAGS	OPTIMIZATION_CFLAG, WARNING_CFLAGS
DEBUG_CFLAGS	Use DEBUG_BUILD_CFLAGS.
PROFILE_CFLAGS	Use PROFILE_BUILD_CFLAGS.
DYLD_APP_STRIP_OPTS	LIBRARY_STRIP_OPTS (-S by default)
RELOCATABLE_STRIP_OPTS	DYNAMIC_STRIP_OPTS (-S by default)
OTHER_DEBUG_LIBS	Add libraries using Project Builder.

OTHER_PROFILE_LIBS	Add libraries using Project Builder.
OTHER_JAPANESE_DEBUG_LIB	Add libraries using Project Builder.
OTHER_JAPANESE_PROFILE_LIBS	Add libraries using Project Builder.
BUNDLE_LIBS	Add libraries using Project Builder.
PRECOMPS	Use Project Builder Inspector to mark headers for precompiling.

In addition, the following are some other changes to makefiles of which you should be aware:

- If your project is a library shared by several other projects, consider converting it to a framework. For more information, see the book *OPENSTEP Development: Tools and Techniques*.
- Many things that you used to have to set using the Makefile preamble and postamble files you can now set using Project Builder. For example, you can set search paths and simple compiler flags. You should minimize your use of the preamble and postamble files and use the Project Builder interface instead. For more information, see the development environment release notes.
- The build process now uses **gnumake** instead of **make**.
- If you wrote any top-level double-colon targets that are also implemented by Project Builder, such as **app::**, **install::**, **all::**, now is a good time to rename them. It is likely that the variables passed to these rules and the order in which they are executed has changed. Consider using **after_install::**, **before_install::**, **OTHER_INITIAL_TARGETS**, or **OTHER_PRODUCT_DEPENDS**.
- Once you save your project in 4.0, it uses the new makefile behavior. If your project must use the 3.3 makefile behavior, set **MAKEFILEDIR** in **Makefile.preamble** to **/NextDeveloper/Makefiles/app**.

Generating the Conversion Scripts

Some of the conversion scripts need to understand your application's class hierarchy and how you implemented certain methods. For this reason, you must generate some of the conversion scripts yourself.

To generate conversion scripts, do the following:

1. Add **/NextDeveloper/OpenStepConversion/UtilityScripts/shellscripts** to your **PATH** environment variable. The **convert** command, which you use to convert your code, is in this directory.

2. In a Terminal window, enter these commands to create files that will be used to generate the conversion scripts:

   ```
   % cd project_directory
   % convert -preprocess
   ```

3. Modify the files **StringMethods**, **StringDefines**, **RectMethods**, and **VoidMethods**, which are used to generate the conversion scripts. Carefully read "Modifying the Files Used to Generate Conversion Scripts" below for instructions on how to do this. Once you perform step 4, you cannot return to this step.

WARNING: Do not move on to step 4 until you have fully completed step 3.

4. Generate the optional conversion scripts with this command:

   ```
   % convert -makescripts
   ```

Modifying the Files Used to Generate Conversion Scripts

The **convert -preprocess** command creates a **CONVERSION** directory under your project directory and stores in it files that describe your application. These files and their contents are described in the table below.

File	Description
ClassHierarchy1	Describes the class hierarchy before conversion in a format **tops** can understand.
ClassHierarchy2	Describes the class hierarchy after conversion in a format **tops** can understand.
StringMethods	Lists methods that have **(char *)** or **(const char *)** as either a parameter type or return type.
StringDefines	Lists **#define**s that were determined to be of type **(const char *)**.
RectMethods	Lists methods that use pointers to NXRects, NXSizes, or NXPoints as either a parameter or a return type.
VoidMethods	Lists methods that did not specify a return type.

In order to generate conversion scripts that will convert your code properly, you need to modify all of the files produced by **convert -preprocess** except the class hierarchy files. The following sections provide instructions on this step.

After you modify these files and run **convert -makescripts**, some additional conversion scripts appear in the **CONVERSION** directory. These scripts are executed during the conversion process.

StringMethods

OpenStep provides a new object called NSString. NSString allows you to perform character manipulation on strings without requiring that you know which character encoding is being used. Using NSStrings, you can write truly portable and internationalized code, code that will work with any

writing system supported by the Unicode standard. The Application Kit now uses NSStrings where it used to use C strings in method and function arguments and return values. Because of this and because of the advantages of NSStrings, you may want to convert all of the C strings in your application so that they use NSStrings.

The **StringMethods** file contains a list of every method in your application that takes a C string as an argument or returns a C string. It does not list overrides of Application Kit methods; those are taken care of by the conversion scripts provided in the release. All of the methods listed in this file will have their C string arguments and return values converted to NSStrings.

Only methods that don't modify the string should have their C strings converted. Look at the implementation of every method listed in the **StringMethods** file. If the method modifies its C strings, remove its name from the file. To skip the optional conversion of C strings to NSStrings entirely, delete all of the methods from this file. (You may want to save them in a different file.) Before you decide, you may want to read about the conversion of C strings to NSStrings in the *OpenStep Conversion Guide*. See the chapter "Converting the Common Classes."

You can locate a method's implementation easily in Project Builder by doing the following (for more information, see the development environment release notes):

1. Choose Find from the Tools menu to display the Project Find panel.
2. In the Project Find panel, enter the name of the method in the text field.
3. Make sure Definition is selected in the pop-up list.
4. Click the Find button.

StringDefines

This **StringDefines** file contains **#define** macros that are string constants or **NXLocalizedString...** function calls. These macros will be converted so that they create NSString objects instead of C

strings. If this file lists macros that you want to remain C strings, delete the line naming that macro from the **StringDefines** file.

RectMethods

All Application Kit functions and methods that used to take the address of an NXRect or an NXSize now take the value of the structure. Similarly, all Application Kit functions and methods that used to return a pointer to an NXRect or NXSize now return the structure itself. This change was made to eliminate the aliasing problems that can occur when you pass pointers and to allow these methods to work better with the Distributed Objects system.

The **RectMethods** file contains a list of every one of your application's methods that takes a pointer to an NXRect, NXSize, or NXPoint structure or returns a pointer to one of these structures. (It does not list overrides of Application Kit methods.) All of the methods listed in this file will have their structure pointer arguments and return values converted to the actual structure.

Only methods that don't modify the structures should have their arguments and return types modified. Look at the implementation of every method listed in the **RectMethods** file. If the method modifies an NXRect, NXPoint, or NXSize structure, remove its name from the file. To skip the optional conversion of structure pointers entirely, delete all of the methods from this file. (You may want to save them in a different file.) Before you decide, you may want to read about the NXRect, NXPoint, and NXSize conversions in the *OpenStep Conversion Guide*. See the chapter "Converting the Common Classes."

VoidMethods

Previously, methods returned **self** by convention. Some methods return **self** to indicate success and **nil** to indicate failure. Returning **self** to indicate a Boolean value or returning **self** without any associated meaning made the API more confusing. In OpenStep, when a method has no real value to return, its return type is **void**. Where a method returned **self** or **nil**, it now returns **BOOL**. In addition

to being cleaner API, returning **void** and **BOOL** helps you avoid creating unnecessary proxies if you're distributing objects. The **VoidMethods** file contains a list of every method in your application that has no return type specified (except for overrides of Application Kit methods).

Look at both the implementation and the uses of each of the methods listed in the **VoidMethods** file and perform the action listed below. (To look at all of the uses of a method, enter its name in Project Builder's Project Find panel, choose References from the pop-up list, and click Find. The bottom half of the Find panel lists all of the places the method is invoked.)

- If the method returns either **self** or **nil** and invocations of the method test the return value, change the prefix for that method in the **VoidMethods** file to **SELFNIL-BOOL**. This will convert the method to return **BOOL**. (By default, all methods in the file are prefixed with **SELF-VOID**, which means they will be converted to return **void**.)
- Delete the method from the **VoidMethods** file if its return type is ever used in any of the places where it is invoked (and it should not be converted to return **BOOL**).
- **init...** methods do not appear in the **VoidMethods** file because it is correct for them to return type **id**. If you have a method that starts with **init...** but is not a initialization method for its class, add it to the **VoidMethods** file if it should return **void**.
- Leave all **+initialize** methods in the file. The **+initialize** method now returns **void**.
- If the method returns a value other than **self** and that value is never used anywhere the method is invoked, change its prefix to **OBJ-VOID**.
- To skip the optional void conversion, delete all of the methods from the **VoidMethods** file. (You may want to save them to a different file.)

For example, consider the methods shown in the following code fragment. All of these methods except **newCount** would be listed in the **VoidMethods** file because they don't specify a return type. However, the **countingObject** method has a meaningful return value because **countSomething**

expects it to return an object. **countSomething** is is the only method that should truly be converted to return **void** because it is invoked as if it already did return **void**. Thus, in the **VoidMethods** file, you would delete **countingObject** but leave **countSomething**.

```
- countingObject
{
   return countingObject;
}

- countSomething
{
   [[self countingObject] incrementCount];
   return self;
}

- (int)newCount
{
   [self countSomething];
   return [countingObject currentCount];
}
```

Before you decide which methods should be converted to return **void**, you may want to read about the **void** conversion in the *OpenStep Conversion Guide*. See the chapter titled "Global API and Style Changes."

Running the Conversions

The conversion process is organized into six stages. Each stage runs a series of scripts on your code and makes changes based on the information in those scripts. When converting code for the first time, you should perform the conversion in stages. Here's the recommended procedure:

1. Back up the project directory.

2. In Project Builder, close all of the source files.

   The `convert` script, which you run in the next step, changes your source files. If you have looked at these same source files in Project Builder before you run the script, the file displayed by Project Builder won't reflect the changes that `convert` makes. To make sure that you are always looking at the latest version of your files, close all of the files in Project Builder before you run the script.

   You can see which files you have loaded in Project Builder using the Loaded Files panel. Choose Loaded Files from the Tools menu to bring up the Loaded Files panel. Select a file in the Loaded Files panel, then choose Close from the File menu to close it.

   If you don't perform this step, `convert` will still work properly, but when you look at the source files after it is complete, they won't reflect the changes. To see the changes that `convert` made, press Command-u.

3. In a Terminal window, enter:

   `% cd` *project_directory*
   `% convert -stage`*x*

   where *x* is the number (1 through 6) of the conversion stage you want to perform. If you don't specify source files on the command line, they will be identified with the pattern:

   ```
   *.[hcmCM] *.psw* *.*proj/*.[hcmCM] *.*proj/*.psw*
   ```

   In other words, all code files in your current directory and in the first level of subproject directories will be converted. If this is not sufficient, specify the appropriate file list after the **-stage** option:

   `convert -stage`*x file1 file2* ...

   Each conversion stage takes several minutes to complete. Stage 1 is the longest stage.

WARNING: The conversion scripts should not be run out of order, and it is not recommended that

they be run on the same file more than once. In general, it's a good idea to save a copy of your files after each stage.

4. Once the conversion is complete, use FileMerge to compare your project with the backup of your project. This is a good way to learn about the differences between NEXTSTEP and OpenStep. (To learn how to use FileMerge, see the development environment release notes.)

5. Open the project in Project Builder and use the Inspector panel to add **/NextDeveloper/OpenStepConversion/IntermediateFrameworks***x* to the search path for frameworks, where *x* is the number of the conversion stage. (For stages 2 through 6, remove the **IntermediateFrameworks** directory for the previous step.)

   In Project Builder, you change the search paths for frameworks from the Build Options inspector. Choose Inspector from the Tools menu to bring up the inspector panel, and choose Build Options from the inspector's pop-up list. In the Build Options inspector, there is another pop-up list. Choose Framework Search Order from that list. Type the new search path for frameworks, then click Add.

   In between the first conversion stage and the last conversion stage, your code is in an interim state and will not compile successfully with either the NEXTSTEP 3.X headers or the OPENSTEP for MachOS Release 4.0 headers. The **IntermediateFrameworks** directories have NEXTSTEP headers at the intermediate stages of conversion so that your code will compile. Your code will not link successfully until after the sixth stage.

   There is no **IntermediateFrameworks** directory for stage 6. After stage 6, you should use the default search path for frameworks (**/NextLibrary/Frameworks**).

   NOTE: Be sure you change the Framework Search Order, not the Header Search Order.

7. Build your project and work through any error messages.

   The conversion process places **#error** messages in places where automated conversion was not possible and **#warning** messages in places where the conversion might not be correct. You will

see these messages when you compile. Once your code compiles successfully, back up your project again, and run the next stage.

If you need help deciding how to correct an error, see the *OpenStep Conversion Guide*. It describes how to correct most of the errors that occur. If you need more information about a particular class or function, look in the *Foundation Framework Reference* in **/NextLibrary/Frameworks/Foundation.framework/Resources/English.lproj/Documentation** or the *Application Kit Reference* in **/NextLibrary/Frameworks/Foundation.framework/Resources/English.lproj/Documentation**. If the class has no documentation yet, see the *OpenStep Specification* in **/NextLibrary/Documentation/OpenStepSpec**.

NOTE: If Project Builder is not showing you the updated source files (the source code does not match the warning you see), type Command-u. Be sure to close all files in Project Builder as described in Step 3 before you run the next conversion stage.

You may want to set the Continue After Error preference to Project Builder. If you do, Project Builder will continue to build even after it finds an error. You can find the Continue After Error preference in the Preferences panel, under Build.

8. If you're converting an application project, convert your nib files. To do this, use this command:

```
% convert -nib
```

Running All Conversions At Once

After you have converted at least one project and you are more familiar with the conversion, you may want to run all conversions at once. To do this, enter this command in the terminal window:

```
convert -all
```

This command runs all six conversion stages one by one, then converts any nib files. Before you enter this command, you still must generate the conversion scripts for the project as described earlier in this document. Remember to be very careful when modifying the files used to generate the conversion scripts. It will save you time in the end.

Running the Optional Conversions

After you have completed all of the conversion stages you may wish to run remaining optional conversions. The optional conversions are listed below.

Conversion Script	**Purposes**
CustomIBAPI.tops	Converts Interface Builder API for palettes.
ListToMutableArray.tops	Converts List objects.
HashAndStringTableConversion.tops	Converts HashTable and NXStringTable objects.
StringConversion2.tops	Converts more C strings to NSStrings.
StreamToMutableData.tops	Converts streams to NSMutableData objects.
StreamToString.tops	Converts streams to NSString objects.
TableView.tops	Converts NXTableView objects to NSTableView objects.
VMConversion.tops	Converts MachOS virtual memory functions to Foundation functions.

To run a single script on your source code, use this command:

```
tops -scriptfile scriptFile *.[hcmCM] *.psw* *.*proj/*.[hcmCM] *.*proj/*.psw*
```

Where *scriptFile* is the complete path for the script (**/NextDeveloper/OpenStepConversion/ConversionScripts/***scriptName*). See the *OpenStep Conversion Guide* for information on these scripts.

NOTE: Because the API changes between C strings and NSStrings and between NXTableViews and NSTableViews are significant, the usefulness of the scripts **StringConversion2.tops** and **TableView.tops** vary. You may want to run them on a copy of your code first to see if they help you.

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.

Notes Specific to GDB on Mach:

New GDB Version

The GDB debugger in OpenStep 4.0 for Mach (and later versions) 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

Debugging Apps that Use the Sybase Client Library

GDB hangs (actually, the new Sybase CT-Lib adaptor blocks) when you use the **next** command to step over a line of code which eventually causes a call to the Sybase client library.

More generally, when debugging a program that uses multiple threads it's possible to create a situation in which a deadlock will occur. Whenever the **next** or **step** commands are issued, GDB lets only the thread being debugged to execute. All other threads are suspended until the command is completed. Therefore, if you attempt to step over a line of code which tries to communicate with another thread, the program will deadlock.

To deal with problems like this, a "run-all-threads" option has been added to GDB. This controls whether or not all of the threads should execute while single-stepping. The default value for this option is "off", meaning the behavior is the same as in OPENSTEP 4.1. In order to prevent a deadlock like that described above, issue the following command in GDB:

set run-all-threads on

We recommend that you use this option only if you're experiencing a deadlock. Allowing other threads to execute while stepping through code can produce confusing results if, for example, the

other threads may be changing the values of global data.

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:

GDB Version

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. 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.

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 usually be the base address at which you linked the DLL plus 0x1000. However, if the address of a DLL that you build conflicts with another DLL in the process, it may automatically be assigned a new address. If you run the program under GDB (instead of attaching to it), GDB displays the address at which each DLL actually landed.

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 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). By default, GDB's ªcurrent languageº is Objective-C. At any time, you can find out what GDB's ªcurrent languageº is with **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, in very rare instances the program will be in the middle of executing Windows code that cannot be debugged. Occasionally you'll find that 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.

Documentation

This file contains release notes for the Documentation distributed with this release of OPENSTEP 4.2.

Locating the Developer Documentation

All of the documentation for OPENSTEP 4.2 can be found on-line (some of the documents are supplied in printed form, as well). On OPENSTEP for Mach, 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 4.2, 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**.

Driver Development

OPENSTEP 4.2 cannot be used to develop Mach device drivers. Because NEXTSTEP 3.3 device drivers work on OPENSTEP 4.2 systems, we recommend that you use NEXTSTEP 3.3 Developer to create device drivers.

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.

OPENSTEP Enterprise

On OPENSTEP Enterprise, double-click the BooksOnline entry in the NeXT Software program group to gain access to the complete on-line documentation set.

PDO

On PDO platforms, all documentation is located in **/NextLibrary/Documentation**.

Release Notes

/NextLibrary/Documentation/NextDev/ReleaseNotes contains important information about the release that was known at the time that the OPENSTEP 4.2 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.

The NeXT OLE ORB

This document lists all of the API changes and some of the bug fixes relating to the NeXT ORB for this release.

New Features in This Release

Supporting Localization in OLE Automation

OLE Automation offers support for localization by allowing the developer to specify a locale when looking up the names of properties, or of methods and their arguments. In release 4.1, the NeXT ORB would always specify the default system locale for the machine it was running on when matching an OLE method or property to a given Objective-C method. As a result, the Objective-C protocols a client used to talk to an OLE server would depend on the host on which the server was running. For this reason, the ORB now has new methods which allow you to specify the language to use when connecting to OLE Automation objects. From Objective-C, instead of using:

```
-(id)objectWithRegisteredName:(NSString *)name
     protocol:(NSString *)proto
     host:(NSString *)hostName
```

you can now use:

```
-(id)OLEObjectWithRegisteredName:(NSString *)name
     localeId:(unsigned long)localeId
     host:(NSString *)hostName
```

From Visual Basic, the alternative to this standard call:

```
orb.connectTo(name, protocol, host)
```

is:

```
orb.connectUsingOLE(name, localeId, host)
```

In the above, the *localeId* arguments are Win32 LCID's: 4-byte quantities which specify a primary language and sublanguage. (For more details see the Microsoft Win32 SDK documentation.) Common values are 0x9, for ªneutralº English, or 0x409 for North American English.

These new methods set the locale for the returned object. Also, the locale propagates recursively to all objects returned from that one.

Bugs Fixed in This Release

Supporting ªOldº DO

In OPENSTEP 4.1, OLE clients were unable to connect to Objective-C servers using ªoldº (NXConnection-based) DO. This has been fixed in release 4.2. However, this fix involves a change to the file **nxorb.m** (shipped on Windows systems in the directory **NextDeveloper/Libraries**). If you want Objective-C and OLE Automation objects to communicate using old DO, your Objective-C programs must be re-linked against the new version of this file.

Enterprise Objects Framework 2.1

This file describes new product features and explains how to install the examples. It does not contain complete release notes for release 2.1 of Enterprise Objects Framework. Complete release notes are

installed when you install EOF 2.1.

For the 2.0 release notes (which include a lot of information that's still relevant for 2.1), see NeXTanswer #2455. If you have access to the World Wide Web, you can look up the URL **http://www.next.com/NeXTanswers/HTMLFiles/2455.htmld/2455.html**.

The documentation supplied with this release is 2.0 documentation. The 2.1 documentation will be made available when the product is released.

The MachOS version of this release is 3-way fat; it can be used to develop software for NeXT, Intel, and SPARC. The OpenStep for Windows version of this release can be used to develop software for Intel machines running Windows NT.

New Features for 2.1

- You can add methods to your interface in Interface Builder to perform queries and calculations on a specified class property.
- You can associate a qualifier with a display group in Interface Builder.
- Interface Builder allows you to set a fetch limit.
- You no longer have to add key paths as strings in Interface Builder-- instead, you can traverse the object graph in the Inspector to connect to the desired property.
- There is a new EOComboBoxAssociation.
- There is a new EOArrayDataSource class.
- EOModeler has a new "Explorer" outline interface that makes model traversal easier.
- When you connect a control to a display group in Interface Builder, a formatter is automatically added to the control based on the associated property's data type.

- There is a new Enterprise Objects Framework wizard that automates the creation of simple Enterprise Objects Framework applications. When you create an application in Project Builder, you have the option of either creating a conventional Enterprise Objects Framework application or using the wizard to automate application creation.
- EOModeler provides an Inspector for editing the connection dictionary.
- The Diagram View in EOModeler, formerly just an example, is now a part of the product.
- The Enterprise Objects Framework examples have been updated.
- The Oracle and Informix adaptors link with the newest client libraries.
- The new Oracle login panel is designed to work with SQL*Netv2.

Installing the Examples

This release provides on-line examples to help familiarize you with Enterprise Objects Framework 2.1. These examples are located in /**NextDeveloper/Examples/EnterpriseObjects**. Installing the examples involves these steps:

- Setting up users and databases on your database server for the example databases.
- Installing the example directory.
- Populating your database server with example data.

Note: When installing the examples for use with the Informix adaptor, use EOModeler's Connection Dictionary Inspector to set "databaseEncoding" to "Non-lossy ASCII" for the Movies and Rentals models. This will allow Enterprise Objects Framework to insert records with non-English characters.

Setting up Database Accounts

The Enterprise Objects Framework 2.1 examples use two sets of tables: Movies and Rentals. Some

examples use just one of the these databases, while others use both. The multi-database support in Enterprise Objects Framework 2.1 makes it possible for you to install these databases in three different configurations:

· Both sets of tables together in a single user/database.
· Each set of tables in its own user/database on the same database server.
· Each set of tables on its own database server (for example, Movies on Informix, Rentals on Oracle).

Depending on your desired setup, you use the tools available with your database server to set up one or two new user/databases. For example. on Sybase you might create a new database on your server called ªMoviesº and login with the user ªsaº. On Oracle you might create a new user with the name ªMoviesº. Once you have set up these accounts, you're ready to install the examples.

Copying the Example Directory

To configure and build the examples you need to copy the example directory to a writable area in your file system. You can do this by copying the **/NextDeveloper/Examples/EnterpriseObjects** folder into your home directory (or any other directory writable by you).

Configuring the Example Models

The model files used by the examples must be configured to use your adaptor and server. To configure the examples, run the **configure_examples** program in your copy of the examples directory. It will ask you for the name of the adaptor you wish to use (Informix, Oracle, Sybase, and so on) and for the login information for your database. It will then convert the example models for your server.

Populating the Databases

Now that the examples are configured, you can fill your example databases with sample data. The **install_database** tool in the **DatabaseSetUp** directory will connect to your databases, add the example tables, and fill them with data. If you later wish to remove the data, simple run the **drop_database** tool.

Building the Example Programs

With your example projects installed and your database filled with data, you are ready to build and run the examples. To do this, in a command shell **cd** to your example directory and type **make all**. This performs a **make install** on **BusinessLogic.framework** and **EOExtensions.framework** to put them in **/LocalDeveloper/Frameworks**, where they are shared by many of the other examples. It then makes all of the examples applications.

Note: The **/LocalDeveloper/Frameworks** directory must be created and writable by you in order to build the examples. On Mach this can be accomplished by using **su** to become the superuser and then executing the following commands:

```
mkdirs /LocalDeveloper/Frameworks
chmod a+w /LocalDeveloper/Frameworks
```

Project Builder Software Configuration Management Extensions

Overview

Project Builder can now be extended to support interaction with third-party Software Configuration Management (SCM) systems. This is done by loading ªSCM Adaptor Bundlesº into Project Builder; these bundles plug into Project Builder's public SCM programmatic interface. OPENSTEP 4.2 on Windows includes a bundle that allows you to interact with INTERSOLV's PVCS Version Manager product. On both Mach and Windows, an unsupported bundle supports the GNU Concurrent Version

System (CVS). Both integrations make use of the same GUI extensions to Project Builder, but invoke different sets of SCM operations that are specific to the SCM system under which the current project lives.

INTERSOLV Polytron Version Control System (PVCS)

The **PVCS_Support** bundle requires that PVCS Version Manager be installed on each machine that uses it. This integration has been tested on the current version on PVCS, which is 5.2.20. It will also work with Intersolv's upcoming release of PVCS Version Manager 5.3 (it's also possible that later versions of PVCS will work using this integration bundle). PVCS can be obtained directly from Intersolv:

1700 NW 167th Place
Beaverton, OR 97006

1-800-547-4000 (General Phone)
1-800-443-1601 (Tech Support Phone)
1-503-645-6260 (Tech Support Fax)

pvcs_answerline@intersolv.com
http://www.intersolv.com

Other SCM Products

The **CVS_Support** bundle requires the public-domain CVS package that can be found at many popular ftp sites on the Internet. While it has proven useful in some organizations, it is not a supported configuration management product and it does have its limitations.

Support is planned for other popular SCM products that are accessible from Mach or Windows. Please contact NeXT Product Marketing regarding future plans and schedules of SCM integrations with Project Builder.

Using an SCM Adaptor Bundle

Before using an SCM Adaptor Bundle from Project Builder, you must configure Project Builder to load the bundle at startup and you must set up the defaults database to specify your integration preferences.

Configuring Project Builder

To configure Project Builder to load an SCM bundle at startup, launch Project Builder and select Preferences from the main menu. In the Bundles pane of the Preferences panel, click Add; this brings up a file browser that allows you to select the location of the bundle. Select the PVCS or CVS bundle, as appropriate, from **$(NEXT_ROOT)/NextDeveloper/PBBundles**. Then, re-launch Project BuilderÐthis causes the bundle to load and take effect.

Setup Defaults

You'll need to set some defaults in order to configure Project Builder to work with SCM. You may either use the SCM Preferences panel to set these defaults, or you may open a terminal and enter the defaults manually. If using a shell (such as the Bourne shell; defaults cannot be set from an MS-DOS prompt), enter the following commands to set the SCM defaults (replace any information in italics with your local information):

...If you're using PVCS:

```
defaults write ProjectBuilder PVCSDefaults '{path="PVCS_Executable_Path";
userName=unknown;}'
```

The following default should either be YES or NO, depending on whether or not you want to lock files automatically when you start to edit them in Project Builder.

```
defaults write ProjectBuilder lockOnEdit YES_or_NO
```

...If you're using CVS:

```
defaults write ProjectBuilder CVSDefaults '{path="CVS_Executable_Path";}'
```

Supported Operations

The SCM bundles support the following operations on files under SCM control:

- Create a new Work Directory from an existing Repository
- Add/Remove Files
- Lock/Unlock Files (PVCS only)
- Update Files
- Merge Files
- Tag Files
- Display History or Show Changes for Files

Some operations, such as creating a new repository or importing a project into it for the first time, must be performed using tools provided by the underlying SCM system.

PVCS User's Guide

Once you've configured Project Builder and the defaults database for use, you're ready to take your SCM system for a test drive. You may find the following guide helpful in getting started with Project

Builder and PVCS:

810820_SCMTutorial.rtf ¬

CVS Note

With the CVS Adaptor, you must have the CVSROOT environment variable set for Project Builder to inherit. The repository path you specify when creating a new CVS work area (which is currently necessary to use the integration) must be relative to the CVSROOT pathÐit cannot be absolute.

Known Bugs and Limitations in the SCM Integrations

Reference: 1463

Problem: File and Project RENAME not supported

Description: Renaming files or projects under SCM control is not supported. Attempting to do so will result in a confused SCM state.

Reference: 1647

Problem: Checking in files sometimes makes them write protected

Description: Occasionally the user may be prompted to overwrite a read-only file or revert a file to saved. These alerts occur mainly because the revision control system is writing to the files without Project Builder's knowledge. In general it is acceptable to revert or overwrite the files and continue working. This problem will be fixed in the next release.

Bugs Specific to PVCS Integration

Currently, there are no known bugs specific to the integration with PVCS.

Bugs Specific to CVS Integration

Reference: 1482

Problem: Updating a file that has been added (but not committed) changes status to up-to-date

Description: If a file is added to a project and then updated before it is checked in, the file status is incorrectly changed to "Up-To-Date" instead of remaining "Added".

Workaround: Check-in added files before you perform any other SCM operations on them.

Reference: 1606

Problem: If there is a lock on the repository SCM will hang forever

Description: If there is a lock on the CVS repository, the SCM adaptor (and hence Project Builder) will hang indefinitely.

Workaround: Manually remove the lock from the CVS repository and restart Project Builder if necessary.

Supplemental User Notes

Most information useful for users is located in the printed user release notes. Additional notes that are

relevant to both end-users and developers are listed here.

Sound.app Can Crash Workspace Manager

Sound.app has a bug that can cause Workspace Manager to crash. If Sound.app is the default application for a sound file, and you double-click that file, Sound.app will startÐwith an apparently blank sound (the title bar will show ªUntitledº). If you then drag the icon for the sound file from Sound.app's file well over the file viewer, Workspace Manager will report an error and you'll be automatically logged out.

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 hyphenation, 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: 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.