OPENSTEP 4.2 Copyright1997 by Apple Computer, Inc. All Rights Reserved.

OPENSTEP 4.2 Release Notes: Project Builder

This file contains release notes for the Project Builder application distributed with OpenStep 4.2. It includes information specific to OPENSTEP for Mach and to OPENSTEP for Windows as well as information general to both platforms.

Project Builder provides integrated code development capabilities for the sophisticated developer. Its main window combines 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 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 4.2

Click-to-Find Button

A new context help button has been added to the right of the toolbar. It activates the context help mode, denoted by the question-mark cursor. In context help mode, you can get information on UI items by clicking on them. You can also use this button for "Click-To-Find" operations to access symbols in the project's index. To get definitions for source code symbols, move the cursor over the source file and click when the symbol has been highlighted. This will bring up the finder panel to display the results.

grab2.tiff ¬

Fuzzy Indexing

A new lighter-weight style of project indexing debuts in this release on both Mach and Windows. It provides a more reliable and low-cost index that is not based on the strict pre-processing and formal parsing methodology employed in 4.0 and 4.1 (which is only available on Mach via a defaults setting). In addition to using less virtual memory and CPU time, the fuzzy parsing system can index symbols inside **#ifdef**s that are not currently turned on in your builds.

Information about frameworks is shared across projects that link against them and can be accessed from the project find panel associated with each such projects. Information about other header files you may import is not currently stored. All existing preferences, indexing panel operations, and find panel queries now apply to the new Fuzzy Indexing feature.

Limitations for 4.2 pre-release: The symbol index is not built for source code modules that contain C++ code or for **.h** files that appear to contain C++-specific declarations. Additionally, reference queries do not work in the project find panel. These should be corrected in the final 4.2 release.

Framework Documentation Access

Although the Find panel can be used to access documentation about a particular symbol, all of

the documentation associated with a framework can now be browsed. The Frameworks suitcase will now show for each framework the on-line available documentation structure. This works as a direct link to the file system with the use of filtering. On Mach, the user can browse the documentation and display any file directly from the project browser. On Windows, the WinHelp system can be launched on a particular framework's documentation.

grab1.tiff ¬

Syntax Coloring

It is now possible to use syntax coloring on source files (C, C++, Obj-C, Java). Syntax coloring is a feature that allows Project Builder to show parts of a source file (tokens) using different text attributes depending on their meaning. For instance, comments can appear in italic text and string constants in red.
grab3.tiff ¬

A new "Syntax Coloring" preferences pane has been added to allow you to switch the syntax coloring off or change the font and color for each lexical category of text.

pref.tiff ¬

The current set of lexical categories include: Comments, Keywords, Strings, and Numbers. It is also possible to use the normal text attributes for a lexical category inherited from the existing "Fonts, Sizes & Colors" preferences pane. Tokens that don't appear in one of the lexical categories (such as identifiers) will always inherit these default attributes. Switching syntax coloring on and off takes place right away. Setting attributes for one or more tokens "dirties" the window; these settings are applied and saved to the defaults when Set is pressed. Revert takes you back to the previously-saved syntax-coloring preferences.

Improved Debugging

A number of bugs have been fixed in the Launcher/Debugger facilicities of Project Builder. New features include the ability to set the directory paths needed for dynamically loaded libraries and, on Windows, better integration between building and debugging an executable. When a path is entered in the Launcher Options Panel environment section, the Launcher sets the appropriate environment variable so that launched and debugged programs use dynamically-loaded libraries located in the specified directory. When building a project on Windows, the final link of the executable will fail if the executable is being run or debugged. Project Builder will now ask you if you want to stop a launched executable when building. If the executable is being debugged, Project Builder will notify **gdb** to release the executable and then reload its symbols when the build finishes. While the build is in progress, you cannot run the executable from within the debugger, but you can manipulate breakpoints.

New Features for 4.1

New Makefiles

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

- 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 do one of the following:

- 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/Project Builder_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 `Project Builder`**.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 the current 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 Builder 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: 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.

In creating the **.def** file for frameworks and dynamic libraries (DLLs), Project Builder handles Objective-C symbols and data but not functions. If you have functions in your framework that need to be callable from the outside, define certain macros in a header file and then use them in your function declarations. These macros allow you to export non-static functions from the DLL (*FRAMEWORK*_EXTERN) or to declare them as **extern**, but not exported (*FRAMEWORK*_PRIVATE_EXTERN). The following example shows how (substitute your own framework name for *FRAMEWORK*):

```
#ifndef _FRAMEWORKDEFINES_H
#define _FRAMEWORKDEFINES_H

#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 extern "C" _FRAMEWORK_WINDOWS_DLL
#define FRAMEWORK_PRIVATE_EXTERN extern
#else
#define FRAMEWORK_EXTERN _FRAMEWORK_WINDOWS_DLL extern
#define FRAMEWORK_PRIVATE_EXTERN extern
#endif
```

You need to import this header in all your other headers and declare your function prototypes accordingly. Then make sure you define *FRAMEWORK*_BUILDING_DLL when building your framework.

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 dwrite, ªtabStopCharsº, that allows you alter this interval to any desired number of spaces, as long as that number is greater than zero.

Converting 3.x applications to 4.x

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 The 4.2 Release Ð Mach and Windows

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

Reference: 76321

Problem: Project Builder edit caches file changes in memory between close and reopen of a project

Description: If you choose not to save changes to a file when closing a project, when the project is opened again Project Builder reuses the file buffer containing the unsaved changes for that file, instead of the version of the file from the disk

Reference: 76133

Problem: File renaming doesn't work properly

Description: Using the Inspector to change the name of a file causes exceptions to be raised inside Project Builder.

Reference: 76114

Problem: Yellow arrows are not displayed in the browser

Description: If you open a file and then index the project, the yellow arrow, which signifies the file is indexed, is not displayed for that file. The arrows are displayed for other files.

Reference: 77979

Problem: LIBRARY_STYLE=STATIC excludes frameworks

Description: If your builds terminate with link errors claiming that imported frameworks are undefined, check your **Makefile.preamble** to see if the following line is uncommented.

```
LIBRARY_STATIC = STYLE
```

If it is uncommented, prepend a hash character (#) to comment it out. Due to a change in the make process, "LIBRARY_STYLE=STATIC" tells the linker not to link against frameworks (such as AppKit and Foundation), even if they are added to the project.

Known Problems in The 4.2 Release Ð Windows

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

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: 76192

Problem: Find results don't show the documentation icon for methods.

Description: After indexing the project, a find on a method name does not show the documentution icon
(a book) for methods which have documentation.

Reference: 75239

Problem: Project Builder thinks some **.nib** files have been modifed

Description: Open a project and then start Interface Builder. Open a **.nib** file with Interface Builder, not by double-clicking on the nib inside Project Builder. Build the project in Project Builder and Project Builder will warn that the **.nib** file is modified, when it is not. The workaround is to start Interface Builder by double-clicking the nib inside Project Builder.

Known Problems in The 4.2 Release Ð Mach

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

Reference: 67785

Problem: Breakpoints on functions or methods without debugging information aren't displayed in the breakpoint inspector.

Reference: 67712

Problem: Moving a breakpoint icon doesn't update the line number in the Breakpoint inspector. However, the location of the breakpoint is updated in **gdb**.