The Cross-Platform Developing Skills for Mac Applications

Xiao Hanyu[1]

October 17, 2009

[1]Computer Science and Technology 0706, Zhejiang University

Contents

Chapter 1

Basics of Mac Platform

1.1 Mac OS and iPhone

1.1.1 Introduction to Mac OS

Mac OS is the trademarked name for a series of graphical user interface-based operating systems developed by Apple Inc. (formerly Apple Computer, Inc.) for their Macintosh line of computer systems. The Macintosh user experience is credited with popularizing the graphical user interface. The original form of what Apple would later name the "Mac OS" was the integral and unnamed system software first introduced in 1984 with the original Macintosh, usually referred to simply as the System software. It was a trimmed-down version of the operating system underpinning Apple's earlier Lisa product.

1.1.2 Introduction to iPhone OS

The iPhone OS, known as OS X iPhone in its early history, is the operating system developed by Apple Inc. for the iPhone and iPod touch. Like Mac OS X, from which it was derived, it uses the Darwin foundation.iPhone OS has four abstraction layers:

1. the Core OS layer
2. the Core Services layer
3. the Media layer
4. the Cocoa Touch layer

The operating system takes less than 240 Megabytes of the device's total memory storage.

The central processing unit used in the iPhone and iPod Touch is an ARM-based processor instead of the x86 (and previous PowerPC or MC680x0) processors used in Apple's Macintosh computers, and it uses OpenGL ES rendering by the PowerVR 3D graphics hardware accelerator co-processor. **Mac OS X applications cannot be copied to and run on an iPhone OS device. They need to be written and compiled specifically for the iPhone OS and the ARM architecture**. However, the Safari web browser supports "web applications" as noted below. Authorized third-party native applications are available for devices with iPhone OS 2.0 and later through Apple's App Store.

iPhone SDK

On October 17, 2007, in an open letter posted to Apple's "Hot News" weblog, Steve Jobs announced that a software development kit (SDK) would be made available to third-party developers in February 2008. The SDK was released on March 6, 2008, and allows developers to make applications for the iPhone and iPod Touch, as well as test them in an "iPhone simulator". However, loading an application onto the devices is only possible after paying an iPhone Developer Program fee. Since the release of Xcode 3.1, Xcode is the development environment for the iPhone SDK.

Developers are able to set any price above a set minimum for their applications to be distributed through the App Store, of which they will receive a 70% share. Alternately, they may opt to release the application for free and need not pay any costs to release or distribute the application except for the membership fee.

SDK contents

As the iPhone OS uses a variant of the same XNU kernel that is found in Mac OS X, the tool chain used for developing on the iPhone OS is also based on Xcode.[4]

The SDK is broken down into the following sets:

1. Cocoa Touch
   (a) Multi-touch events and controls
   (b) Accelerometer support
   (c) View hierarchy

(d) Localization (i18n)

(e) Camera support

2. Media

(a) OpenAL

(b) audio mixing and recording

(c) Video playback

(d) Image file formats

(e) Quartz

(f) Core Animation

(g) OpenGL ES

3. Core Services

(a) Networking

(b) Embedded SQLite database

(c) Core Location

(d) Threads

4. OS X Kernel

(a) TCP/IP

(b) Sockets

(c) Power management

(d) File system

(e) Security

Along with the Xcode toolchain, the SDK contains the iPhone Simulator, a program used to emulate the look and feel of the iPhone on the developer's desktop. Originally called the Aspen Simulator, it was renamed with the Beta 2 release of the SDK. Note that the iPhone Simulator is not an emulator and runs code generated for an x86 target.

The SDK requires an Intel Mac running Mac OS X Leopard or later. Other operating systems, including Microsoft Windows and older versions of Mac OS X, are not supported.

1.1.3 iPhone and iPod Touch

The iPhone is an Internet and multimedia enabled smartphone designed and marketed by Apple Inc. Because its minimal hardware interface lacks a physical keyboard, the multi-touch screen renders a virtual keyboard when necessary. The iPhone functions as a camera phone (also including text messaging and visual voicemail), a portable media player (equivalent to a video iPod), and an Internet client (with email, web browsing, and Wi-Fi connectivity). The first-generation phone hardware was quad-band GSM with EDGE; the second generation added UMTS with 3.6 Mbps HSDPA; the third generation adds support for 7.2 Mbps HSDPA downloading but remains limited to 384 Kbps uploading as Apple had not implemented the HSPA protocol.

The iPod Touch(trademarked and marketed as iPod touch) is a portable media player, personal digital assistant, and Wi-Fi mobile platform designed and marketed by Apple Inc. The product was launched on September 5, 2007 at an event called The Beat Goes On. The iPod Touch adds the multi-touch graphical user interface to the iPod line. It is the first iPod with wireless access to the iTunes Store, and also has access to Apple's App Store, enabling content to be purchased and downloaded directly on the device.

The iPod Touch and the iPhone, a smartphone by Apple, share the same hardware platform and run the same iPhone OS operating system. The iPod Touch lacks some of the iPhone's features such as access to cellular networks and a built-in camera (and microphone on older models); as a result, the iPod Touch is slimmer and lighter than the iPhone.

1.2 Before Developing

1.2.1 Darwin (operating system)

Darwin is an open source POSIX-compliant computer operating system released by Apple Inc. in 2000. It is composed of code developed by Apple, as well as code derived from NEXTSTEP, BSD, and other free software projects.

Darwin forms the core set of components upon which Mac OS X and iPhone OS are based. It is compatible with the Single UNIX Specification version 3 (SUSv3) and POSIX UNIX applications and utilities.

1.2.2 Cocoa API

Cocoa is one of Apple Inc.'s native object-oriented application program environments for the Mac OS X operating system. It is one of five major APIs available for Mac OS X; the others are Carbon (deprecated), POSIX(for the BSD environment), X11 and Java.

Cocoa applications are typically developed using the development tools provided by Apple, specifically Xcode(formerly Project Builder), and Interface Builder, using the Objective-C language.

1.2.3 POSIX

POSIX ["Portable Operating System Interface for Unix"] is the name of a family of related standards specified by the IEEE to define the application programming interface (API), along with shell and utilities interfaces for software compatible with variants of the Unix operating system, although the standard can apply to any operating system.

1.3 Developing Tools

1.3.1 Xcode

Xcode is a suite of tools for developing software on Mac OS X, developed by Apple. Xcode 3.2, the latest major version, is bundled free with Mac OS X v10.6, but is not installed by default. Because version 3.2 is not supported on older Mac OS versions, more dated versions of Xcode are free from the Apple Developer Connection.

The main application of the suite is the integrated development environment (IDE), also named Xcode. The Xcode suite also includes most of Apple's developer documentation, and Interface Builder, an application used to construct graphical interfaces.

The Xcode suite includes a modified version of free software GNU Compiler Collection (GCC, apple-darwin9-gcc-4.2.1 as well as apple-darwin9-gcc-4.0.1, with the former being the default), and supports C, C++, Fortran, Objective-C, Objective-C++, Java, AppleScript, Python and Ruby source code with a variety of programming models, including but not limited to Cocoa, Carbon, and Java. Third parties have added support for GNU Pascal, Free Pascal, Ada, C#, Perl, Haskell, and D. The Xcode suite uses GDB as the back-end for its debugger.

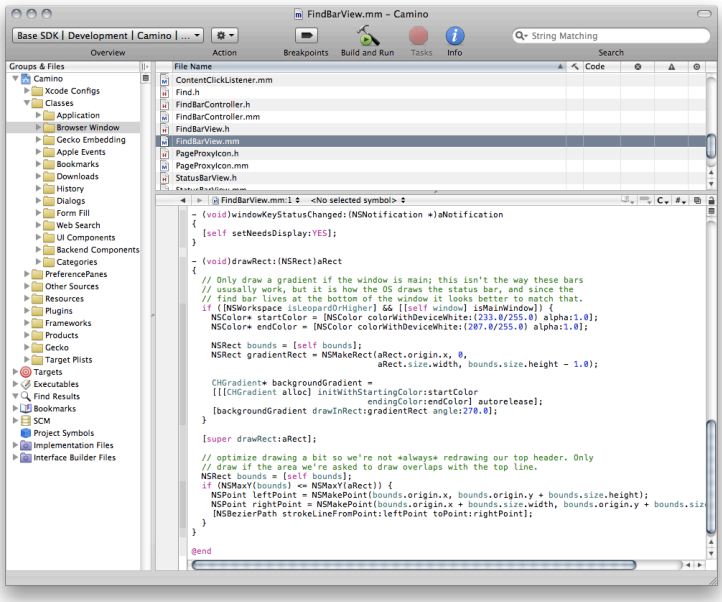

Figure 1.1: Xcode

1.3.2 Interface Builder

Interface Builder is a software development application for Apple's Mac OS X operating system. It is part of Xcode (formerly Project Builder), the Apple Developer Connection developer's toolset. Interface Builder allows Cocoa and Carbon developers to create interfaces for applications using a graphical user interface. The resulting interface is stored as a .nib file, short for NeXT Interface Builder, or more recently, as a .xib file.

Interface Builder is descended from the NeXTSTEP development software of the same name. A version of Interface Builder is also used in the development of OpenStep software, and a very similar tool called Gorm exists for GNUstep. On March 27, 2008, a specialized iPhone version of Interface Builder allowing interface construction for iPhone applications was released with the iPhone SDK Beta 2.

1.4 Objective-C

1.4.1 Introduction to Objective-C

Objective-C is a reflective, object-oriented programming language, which adds Smalltalk-style messaging to the C programming language.

Today it is used primarily on Apple's Mac OS X and iPhone OS: two environ-

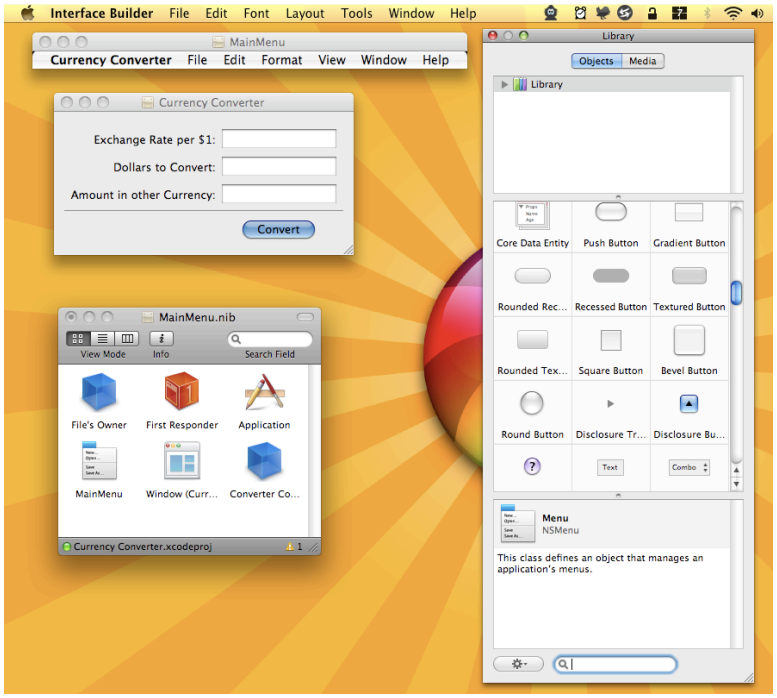

Figure 1.2: Interface Builder

ments based on, although not compliant with, the OpenStep standard. Objective-C is the primary language used for Apple's Cocoa API, and it was originally used as the main language on NeXT's NeXTSTEP OS. Generic Objective-C programs which do not make use of these libraries can also be for any system supported by gcc, which includes an Objective-C compiler.

More precisely, Cocoa is the implementation by Apple, for MacOS X, of the OpenStep standard, originally published in 1994. It consists of a developer framework based upon Objective-C. The GNUstep project is another implementation, which is free. Its goal is to be as portable as possible on most Unix systems, and is still under development [5].

1.4.2 A Short History of Objective-C

A rough history is given in Figure 1.3 to get a quick look at Objective-C amongst its ancestors and "challengers".

Brad J. Cox designed the Objective-C language in the early 1980s. The language was based on a language called SmallTalk-80. Objective-C was layered on top of the C language, meaning that extensions were added to C to create a new programming language that enabled objects to be created and manipulated.

NeXT Software licensed the Objective-C language in 1988 and developed its libraries and a development environment called NEXTSTEP. In 1992, Objective-

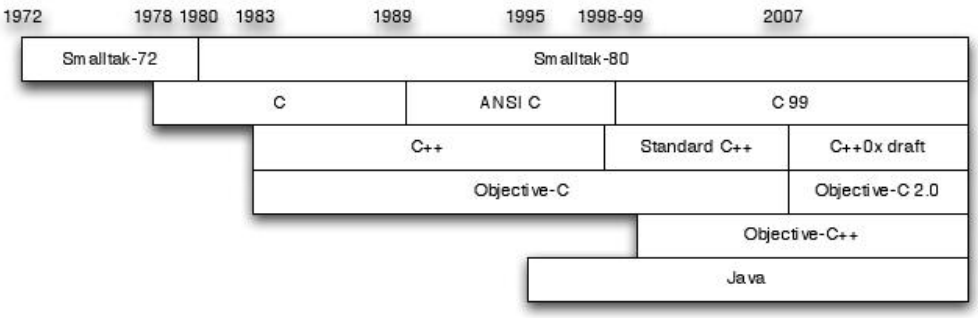

Figure 1.3: Timeline of Java, C, C++ and Objective-C

C support was added to the Free Software Foundation's GNU development environment.This software is in the public domain, which means that anyone who wants to learn how to program in Objective-C can do so by downloading its tools at no charge.

In 1994, NeXT Computer and Sun Microsystems released a standardized specification of the NEXTSTEP system, called OPENSTEP.The Free Software Foundation's implementation of OPENSTEP is called GNUStep.A Linux version, which also includes the Linux kernel and the GNUStep development environment, is called, appropriately enough, LinuxSTEP.

On December 20, 1996,Apple Computer announced that it was acquiring NeXT Software, and the NEXTSTEP/OPENSTEP environment became the basis for the next major release of Apple's operating system, OS X.Apple's version of this development environment was called Cocoa.With built-in support for the Objective-C language, coupled with development tools such as Project Builder (or its successor Xcode) and Interface Builder,Apple created a powerful development environment for application development on Mac OS X.

In 2007,Apple released an update to the Objective-C language and labeled it Objective-C 2.0 [6].

1.4.3 Syntax Overview

Chapter 2

The Cross-Platform Solution

2.1 Cygwin

Cygwin is a Linux-like environment for Windows. It consists of a DLL (cygwin1.dll), which acts as an emulation layer providing substantial POSIX1(Portable Operating System Interface) system call functionality, and a collection of tools, which provide a Linux look and feel. The Cygwin DLL works with all x86 versions of Windows since Windows 95. The API follows the Single Unix Specification2 as much as possible, and then Linux practice. Two other major differences between Cygwin and Linux are the C library (newlib instead of glibc) and default /bin/sh, which is ash on Cygwin but bash on most Linux distributions.

With Cygwin installed, users have access to many standard UNIX utilities. They can be used from one of the provided shells such as bash or from the Windows Command Prompt. Additionally, programmers may write Win32 console or GUI applications that make use of the standard Microsoft Win32 API and/or the Cygwin API. As a result, it is possible to easily port many significant UNIX programs without the need for extensive changes to the source code. This includes configuring and building most of the available GNU software (including the development tools included with the Cygwin distribution) [1].

2.2 Cross-compiling

A compiler is a program that turns source code into executable code. Like all programs, a compiler runs on a specific type of computer, and the new programs it outputs also run on a specific type of computer.

The computer the compiler runs on is called the host, and the computer the new programs run on is called the target. When the host and target are the same

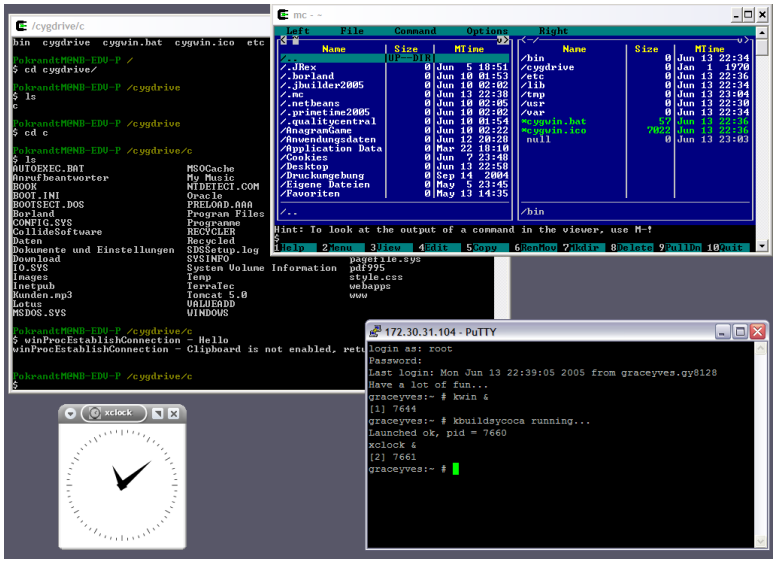

Figure 2.1: Cygwin

type of machine, the compiler is a native compiler. When the host and target are different, the compiler is a cross compiler [3].

To develop an application running on iPhone OS in linux or windows environment, we need cross-compiling, here, the windows or linux platform is the host, and iPhone OS is the target.

The gnu toolchains is one of the best cross-compiling tools including gcc, which is one of the most powerful compiler, supporting various languages, cross-platform, free, and open-source.

Here gives a solution to build an cross-compiling platform in ubuntu linux and windows xp environment.

Figure 2.2: iPhone toolchain, Cross-compiling environment

Make sure you have the following directory structure in you $HOME directory [2]:

You can also build iPhone application through some IDE, such as Eclipse [4].

If you are lucky, you are already got an application that can run in iPhone OS.

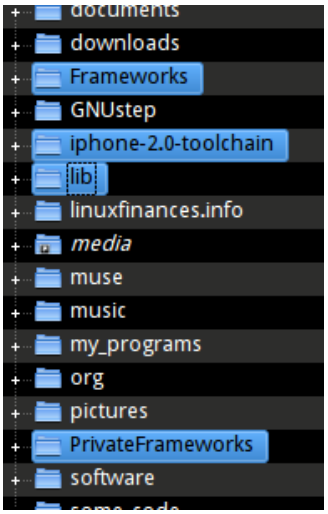

Figure 2.3: the directory structure of our cross-compiling environment

```
cd ~/iphone-2.0-toolchain/examples/GUI/HelloWorldiPhone
make
```

Then copy the HelloWorld.app to /Applications directory of your iPhone OS, type the following commands through SCP terminal:

```
chmod -R 755 /Applications/HelloWorld.app
ldid -S /Applications/HelloWorld.app/HelloWorld
```

Restart the iPhone, run the HelloWorld application, if you are lucky, we'll see the following exciting snapshot:

Now let's start to analyse the source code of HelloWorld.app. First we must know something about GNU make and makefile.

2.3 Makefile

The purpose of the make utility is to determine automatically which pieces of a large program need to be recompiled, and issue the commands to recompile them.

To prepare to use make, you must write a file called the makefile that describes the relationships among files in your program, and the states the commands for updating each file. In a program, typically the executable file is updated from object files, which are in turn made by compiling source files.

In general, makefile defines a dependency tree and some targets, including

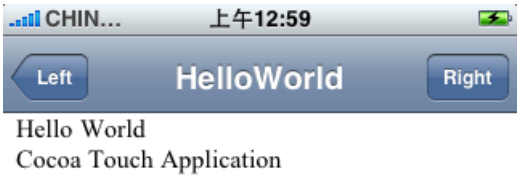

Figure 2.4: HelloWorld

a final target, thus you can type make to finish automatical compiling, make tarballs, delete the cache, and so on [7].

The Makefile for HelloWorld.app is as following:

```
1  CC=/usr/local/bin/arm-apple-darwin9-gcc
2  CXX=/usr/local/bin/arm-apple-darwin9-g++
3  LD=$(CC)
4
5  CFLAGS=-I/usr/local/lib/gcc/arm-apple-darwin9/4.2.1/include
       \
6          -isysroot /usr/local/iphone-sysroot
7
8  LDFLAGS=-framework CoreFoundation -framework Foundation -
       framework UIKit \
9          -lobjc -bind_at_load -isysroot /usr/local/iphone-
               sysroot
10
11 all:    HelloWorld.app
12
13 HelloWorld.app: HelloWorld Info.plist
```

```
14          mkdir -p HelloWorld.app
15          cp Info.plist HelloWorld Default.png icon.png
                HelloWorld.app/
16
17  HelloWorld:     HelloWorld.o HelloWorldApp.o
18          $(LD) $(LDFLAGS) -o $@ $^
19
20  %.o:    %.m
21          $(CC) -c $(CFLAGS) $(CPPFLAGS) $< -o $@
22
23  clean:
24          rm -rf *.o HelloWorld HelloWorld.app
```

The first three line defines three varibles which refers to the c compiler, c++ compiler, and linker, respectively. We can see that the compiler and linker defined in this Makefile is based on darwin foundation from it's name: arm-apple-darwin9.

Later, line 5 and 8 defined the options for compiling and linking.

Line 10, defines the default target: all, which depends HelloWorld.app.

Here $@, $^ and $< is the automatic variable [7].

Clean is a phony target, namely, an action, but not a file. "make clean" defines how to clean the corresponding files in order to rebuild the system.

2.4 GNUstep

2.4.1 Introduction to GNUstep

GNUstep is a free software implementation of NeXT's OpenStep Objective-C libraries (called frameworks), widget toolkit, and application development tools not only for Unix-like operating systems, but also for Microsoft Windows. It is part of the GNU Project.

GNUstep features a cross-platform, object-oriented development environment based on and completely compatible with the OpenStep specification developed by NeXT (which has since been bought by Apple Inc.) and Sun Microsystems. Like Apple, GNUstep also has a Java interface to OpenStep, as well as Ruby and Scheme bindings. The GNUstep developers track some additions to Apple's Cocoa to remain compatible. The roots of the GNUstep application interface are the same as the roots of Cocoa: NeXT and OpenStep. GNUstep

predates Cocoa.

2.4.2 The Correlative OpenStep

OpenStep

OpenStep was an object-oriented application programming interface (API) specification for an object-oriented operating system that uses any modern operating system as its core, principally developed by NeXT with Sun Microsystems. OPENSTEP (all capitalized) was a specific implementation of the OpenStep API developed by NeXT. While originally built on a Mach-based Unix (such as the core of NEXTSTEP), versions of OPENSTEP were available for Solaris and Windows NT as well. The software libraries that shipped with OPENSTEP are a superset of the original OpenStep specification.

2.4.3 Gorm

Gorm(Graphical Object Relationship Modeller)is a graphical interface builder application. It is part of the developer tools of GNUstep.

Gorm is the equivalent of Interface Builder that was originally found on NeXTSTEP, then OPENSTEP, and finally on Mac OS X. Interface Builder was believed by many who have used it to be far ahead of its time. Interface Builder, together with Project Builder (now rewritten as Xcode in Mac OS X versions after Mac OS X v10.3), formed one of the most productive, clean and coherent programming environments and was surely part of the success of the various operating systems that used it.

Gorm and Project Center represent the heart of the suite for GNUstep. Gorm follows Interface Builder so closely that using tutorials written for the latter is possible without much hassle and thus brings the power of Interface Builder to the open source world being part of the GNU project.

Gorm allows developers to quickly create graphical applications and to design every little aspect of the application's user interface. The developer can drag and drop all types of objects like menus, buttons, tables, lists and browsers into to the interface. With the mouse it is possible to resize, move or convert the objects or connect them to functions as well as to edit nearly every aspect of them using Gorm's powerful inspectors.

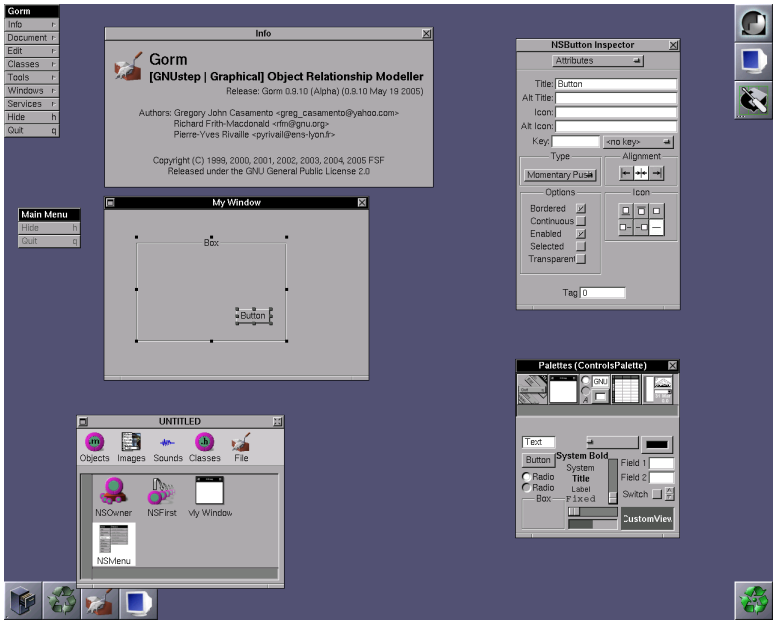

Figure 2.5: Gorm

2.4.4 ProjectCenter

ProjectCenter is GNUstep's integrated developement environment (IDE). It is based in part on NeXT's original Project Builder. It assists you in starting new projects and lets you manage your project files using a intuitive and well ordered graphical user interface.

Supporting the project types 'Application', 'Bundle', 'Library', 'Tool', and 'Aggregate', ProjectCenter automatically creates the project makefiles and aids you in the process of editing, project compilation, package building and debugging. In the future, built-in CVS support will be available, too.

2.4.5 Window Maker

Window Maker is an X11 window manager originally designed to provide integration support for the GNUstep Desktop Environment. In every way possible, it reproduces the elegant look and feel of the NEXTSTEP[tm] user interface. It is fast, feature rich, easy to configure, and easy to use. It is also free software, with contributions being made by programmers from around the world.

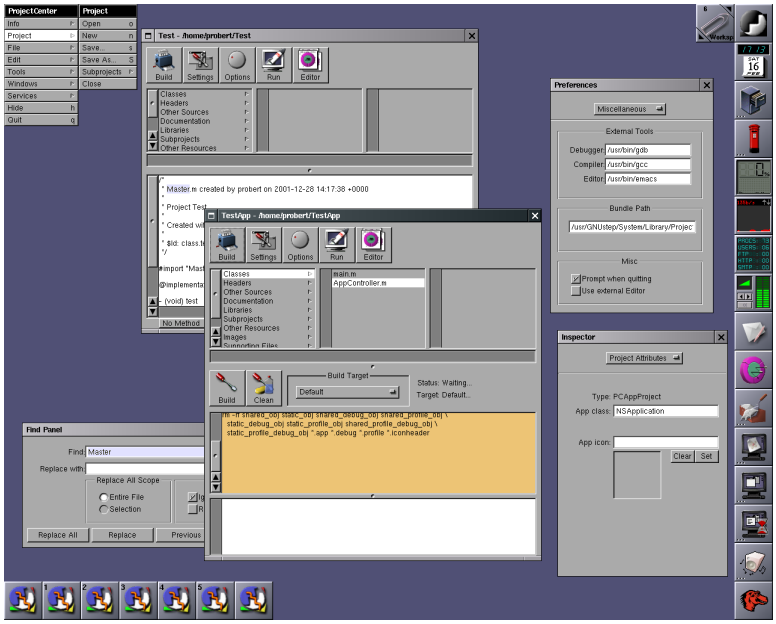

Figure 2.6: Project Center

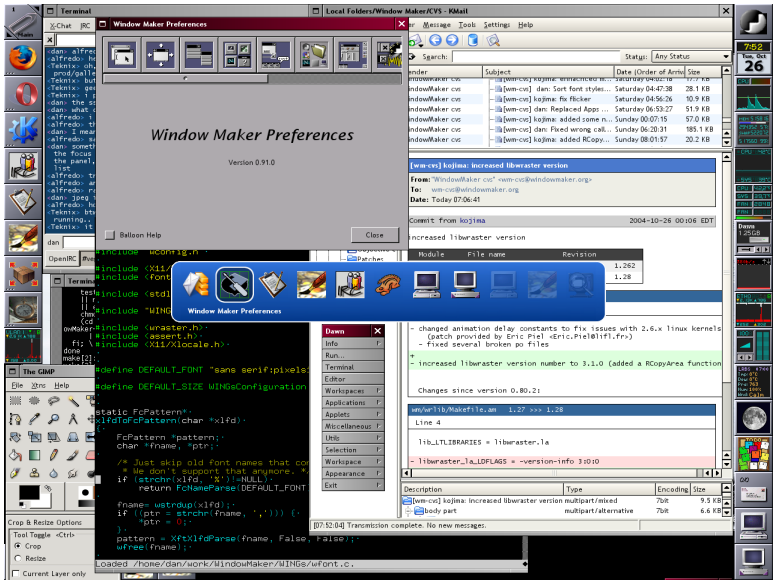

Figure 2.7: Window Maker

Chapter 3

Our First "Hello World" Application on iPhone OS

Now we show the source code of our first "Hello World" application on iPhone [4].

HelloWorldApp.h:

```
//
//  HelloWorldApp.h
//  HelloWorld
//
//  Created by PJ Cabrera on 08/18/2008.
//  Copyright PJ Cabrera 2008. All rights reserved.
//

#import <UIKit/UIKit.h>

@interface HelloWorldApp : UIApplication {
    UIWindow *window;
    UIView *contentView;
    UINavigationBar *nav;
    UITextView *text;
}

@end
```

HelloWorldApp.m:

```
//
```

```
2  //  HelloWorldApp.m
3  //  HelloWorld
4  //
5  //  Created by PJ Cabrera on 08/18/2008.
6  //  Copyright PJ Cabrera 2008. All rights reserved.
7  //
8
9  #import "HelloWorldApp.h"
10
11 @implementation HelloWorldApp
12
13 - (void)applicationDidFinishLaunching:(UIApplication *)
       application {
14         // Create a "full-screen" window to hold the UI.
15         window = [[UIWindow alloc] initWithContentRect:
16                   [UIHardware fullScreenApplicationContentRect
                        ] ];
17
18         // Create a view to hold the window contents.
19         contentView = [[UIView alloc] initWithFrame:
             CGRectMake(0.0f, 0.0f, 320.0f, 480.0f)];
20
21         // Create a navigation bar for the top of the view,
             with two buttons.
22         // The buttons do not do anything in this example.
23         nav = [[UINavigationBar alloc] initWithFrame:
             CGRectMake(0.0f, 0.0f, 320.0f, 48.0f)];
24         [nav pushNavigationItem:[[UINavigationItem alloc]
             initWithTitle:@"HelloWorld"]];
25         [nav showButtonsWithLeftTitle: @"Left" rightTitle: @
             "Right" leftBack: YES];
26         [nav setBarStyle: 0];
27         [contentView addSubview: nav];
28
29         // Create a text view, this is a basic text editor,
             with incorporated keyboard.
30         text = [[UITextView alloc] initWithFrame: CGRectMake
             (0.0f, 48.0f, 320.0f, 480.0f)];
31         [text setText: [[NSString alloc]
```

```
32                  initWithString: @"Hello World\nCocoa Touch
                        Application"]];
33          [contentView addSubview: text];
34
35          // UIWindow con only hold one view, the contentView.
                The contentView can hold many
36          // navigation controllers, which control the
                different views in your application.
37          window.contentView = contentView;
38
39          // These three instructions effectively show the
                window.
40          [window orderFront: self];
41          [window makeKey: self];
42          [window _setHidden: NO];
43  }
44
45  - (void)dealloc {
46          // Release the UI elements as they were allocated
47          [text release];
48          [nav release];
49          [contentView release];
50          [window release];
51
52          // And don't forget to call the parent class dealloc
53          [super dealloc];
54  }
55
56  @end
```

HelloWorld.m:

```
1  //
2  //  HelloWorldApp.h
3  //  HelloWorld
4  //
5  //  Created by PJ Cabrera on 08/18/2008.
6  //  Copyright PJ Cabrera 2008. All rights reserved.
7  //
8
```

```
9  #import <Foundation/Foundation.h>
10 #import <UIKit/UIKit.h>
11 #import "HelloWorldApp.h"
12
13 int main(int argc, char *argv[]) {
14         // This autorelease pool is managed by the UI's
               event dispatcher. It autoreleases
15         // any allocated objects that fall out of scope. The
               iPhone's implementation of
16         // Objective-C 2.0 does not have garbage collection
               yet, but autorelease pools and
17         // proper release of allocated objects is still a
               good practice.
18         NSAutoreleasePool *pool = [[NSAutoreleasePool alloc]
               init];
19
20         // This is where UI execution gets started. This
               instantiates the UIApplication
21         // subclass and UIApplicationDelegate subclass
               specified as string parameters.
22         // The name of an UIApplication subclass can be
               passed as both UIApplication and
23         // UIApplicationDelegate.
24         UIApplicationMain(argc, argv, @"HelloWorldApp", @"
               HelloWorldApp");
25
26         // Force release of the autorelease pool and all
               objects still allocated.
27         [pool release];
28         return 0;
29 }
```

Bibliography

[1] Cygwin user's guide. http://cygwin.com/cygwin-ug-net/.

[2] Develop iphone application in linux. Website. http://www.iphonetoolchain.cn/viewthread.php?tid=18&extra=page%3D1.

[3] Introduction to cross-compiling for linux. Website. http://landley.net/writing/docs/cross-compiling.html.

[4] PJ Cabrera. 使用 eclipse cdt 编写本机 iphone 应用程序. Website, 2008. http://www.ibm.com/developerworks/cn/edu/os-dw-os-eclipse-iphone-cdt.html.

[5] Pierre Chatelier. From c++ to objective-c, 2005.

[6] Stephen G. Kochan. *Programming in Objective-C 2.0*. Addison-Wesley, 2009.

[7] 陈皓. 跟我一起写 makefile.

List of Figures

Index