Write native iPhone applications using Eclipse CDT

How Windows and Linux developers can bypass the iPhone SDK and write iPhone apps using open source tools

Skill Level: Intermediate

PJ Cabrera (pjcabrera@pobox.com)
Freelance Software Developer and Writer
Freelance

30 Sep 2008

Learn how to use the Eclipse C Development Toolkit (CDT) to program native applications for the Apple iPhone, using open source tools to enable iPhone OS development on any Eclipse-supported platform.

Section 1. Before you start

About this tutorial

In July 2008, with the introduction of the Apple iPhone 3G and the iPhone OS V2.0, Apple Computer released production versions of an application development environment for the creation of native iPhone and Apple iPod Touch applications by third parties. But the application development environment is for Apple Mac OS X only, leaving developers using the Microsoft® Windows® and Linux® operating systems out of the loop with regard to developing native applications for this exciting platform.

This tutorial explores the Eclipse C Development Tooling (CDT) project and how it allows you to work with C- and C++-based projects. In this tutorial, you install open source tools to help develop native applications for the iPhone platform on Windows and Linux, and learn about the source code of a basic iPhone application in Objective-C.

Objectives

This tutorial shows how you can use Eclipse and the CDT plug-ins to develop for the iPhone platform, using the open source GNU Compiler Collection (GCC). You use these tools to compile an Objective-C example application for the iPhone.

Prerequisites

This tutorial is written for C and C++ programmers whose skills and experience are at a beginning to intermediate level. You should have a general familiarity using a UNIX® command-line shell and a working knowledge of the C language. The iPhone Cocoa Touch native application development frameworks are written in Objective-C and are based on the Cocoa frameworks used in Mac OS X. Familiarity with Objective-C and Mac OS X Cocoa application development frameworks is not assumed, but such skills are helpful in making sense of the iPhone platform. Further study of Cocoa development is recommended after following this introductory tutorial.

System requirements

To follow along with this tutorial, you need:

- A computer running Windows or Linux with at least 1.5 GB of free disk space.
- An installation of Eclipse.
- Permission to write to the /usr/local directory in the file system (C:/cygwin/usr/local for Windows users).
- Permission to modify your Eclipse configuration by installing plug-ins.

Section 2. The iPhone and iPod Touch platform

The iPhone platform and its cousin, the iPod Touch, are a WiFi-enabled mobile phone and a mobile Internet device, respectively, with a state-of-the-art graphical user interface (GUI), a powerful integrated Web browser, and various other applications, such as e-mail, calendaring, contacts, note-taking, basic Geographic Information System (GIS) functions through its integration of Google Maps, as well as entertainment functions, such as movie and music playing.

The iPhone platform is based on the same Darwin operating system and Objective-C Cocoa frameworks as Mac OS X, modified to fit on an Advanced RISC Machines (ARM)-based embedded device with 128 MB of RAM, several gigabytes of flash storage, and a multi-touch touchscreen. The platform also features an accelerometer for detecting device orientation and movement, and 3-D graphics that rival the capabilities of the Sony PSP and PS2 consoles.

Darwin, Objective-C, and the Cocoa Touch frameworks

Objective-C is a superset of the C language that adds object-oriented features to C, but with a syntax much different than that of C++. Objective-C was used by NeXT Computer in the late-1980s and early 1990s as the development language of choice for its NextStep and OpenStep operating systems. In the mid-1990s, Apple acquired NeXT Computer's technology and assimilated much of NextStep and OpenStep into the Mac OS operating system, producing Darwin and Mac OS X out of the mix. Darwin is an open source UNIX-based operating system created by Apple as the core of the Mac OS X platform.

As with NextStep and OpenStep, the development tools for Mac OS X and the iPhone are based on Objective-C. Apple created a development framework based on Objective-C for Mac OS X: *Cocoa.* The iPhone platform is based on the same Darwin operating system core and Objective-C Cocoa framework as the Mac OS X platform.*Cocoa Touch* is the development framework for the iPhone.

C++ and Objective-C are object-oriented extensions of the C language. But there are key differences between C++ and Objective-C. Objective-C is a language and an object-lifetime management runtime, whereas C++ is only a language, and the programmer manages object lifetime entirely. The other main difference is the syntax of object-oriented operations between the two. C++ extends the C `struct` manipulation syntax, whereas Objective-C invents a new syntax with scoped square brackets. Table 1 presents the main syntactic differences between the languages.

Table 1. Main syntactic differences between C++ and Objective-C

Object-oriented concept	C++	Objective-C	Explanation
Method invocation	`object_or_class.method(2, 4);`	`[object_or_class methodWithParam1: 2 param2: 4];`	C++ extends the C language syntax for accessing elements of a `struct` and reuses it for method invocations

			on classes or objects. Objective-C invented a different syntax with scoped square brackets and named parameters terminated by colons.
Object instantiation	`UIView contentView = new UIView(x, y, width, height);`	`UIView contentView = [[UIView alloc] initWithFrame: CGRectMake(x, y, width, height)];`	In C++ the `new` keyword allocates memory for the object and calls one of the declared constructors that most closely matches the parameters you pass in the code. In Objective-C, you call the `alloc` method of the class, which returns an uninstantiated object of that class. Then, you call an `init` method on the object returned by `alloc` to set initial values for the instance.

Class definitions	class MyClass
	{
	private:
	    int myVariable;
	    void privateMethod();
	public:
	    void setMyVariable(int param);
	}
		@interface MyClass : NSObject {
		    int myVariable;
		}
		
		@private
		- (void)privateMethod;
		@public
		- (void)setMyVariable: (int)param;
		@end
			C++ extends the C `struct` definition syntax by adding the ability to define methods within a `struct` or `class`. It also adds the `private` and `public` keywords (as well as the `protected` keyword not used in this example) to provide the object-oriented feature of encapsulation. In Objective-C, the `@interface` and `@end` keywords are used to define the attributes of a class. Objective-C uses the keywords `@private`, `@public`, and `@protected` with equivalent meaning to the similarly named keywords in C++. Classes in Objective-C have to inherit from another class. In this example, the class NSObject, whereas in

			C++, a class can be the head of its own class hierarchy.

Class and instance methods	`class MyClass`
	`{`
	`private:`
	`    static int classVariable;`
	`    int instanceVariable;`
	`public:`
	`    static void classMethod();`
	`    void instanceMethod(int param);`
	`}`
		`@interface MyClass : NSObject {`
		`    int instanceVariable;`
		`}`
		`+ (void)classMethod;`
		`- (void)instanceMethod: (int)param;`
		`@end`
			C++ uses the `static` keyword within the class body to mean the method or variable is a class method. Objective-C uses a prepended + in front of a class method and a prepended - in front of an instance method.

Method declaration	`MyClass::privateMethod() {`
	`    myVariable++;`
	`}`
	`MyClass::setMyVariable(int param) {`
	`    myVariable = param;`
	`}`
		`@implementation MyClass`
		`- (void)privateMethod {`
		`    myVariable++;`
		`}`
		`- (void)setMyVariable: (int)param {`
		`    myVariable = param;`
		`}`
		`@end`
			C++ adds the scoping operator `::` to declare that a method belongs to a specific class. Objective-C uses the `@implementation` and `@end` keywords to mark a group of methods as belonging to a specific class.

Section 3. Removing application restrictions from your iPhone

Unravel the background behind iPhone application development and discover how to create your own iPhone applications.

Jailbreak

When Apple announced the iPhone, the company downplayed the lack of an unfettered application development environment, citing the need to protect the cell-provider data network from malicious applications running on hacked phones. When introducing the iPhone software development kit (SDK) in March 2008, one of the key features was that the iPhone would only be allowed to run applications signed by Apple using unique keys handed out to iPhone developers with whom Apple has a relationship.

Since September 2007, a group calling itself the iPhone Dev Team developed methods for removing the restrictions Apple devised to limit what could be done on the iPhone. The group helped create the open source tools used to compile programs for the device.

The iPhone Dev Team also created tools that remove restrictions from the iPhone and iPod Touch. These tools permit iPhone owners to use their devices on networks other than AT&T in the United States or to use the iPhone in countries without an iPhone carrier. This process is called *unlocking.*

This tutorial covers "jailbreaking" only, which consists of allowing applications not blessed by Apple to run on the device. Note that the use of these tools is not supported by Apple, and their use may void the device's warranty if Apple has evidence of third-party software modification.

Before you begin
The information presented here is believed to be accurate as of this writing. The steps needed to jailbreak a device may vary with later versions of iPhone OS, QuickPwn, XPwn, Pwnage, and WinPwn. Read the QuickPwn, XPwn, Pwnage, and WinPwn documentation carefully before you begin.

As when using any other unfamiliar software, take your time before you start. Read ahead to understand where you are going and be certain you understand the possible outcomes to using these tools.

Jailbreak tools: QuickPwn, XPwn, Pwnage, and WinPwn

The iPhone Dev Team wrote a series of tools to jailbreak and unlock the iPhone and iPod Touch, each with increasing user-friendliness and sophistication. Installation details and pointers to online documentation are available in Resources.

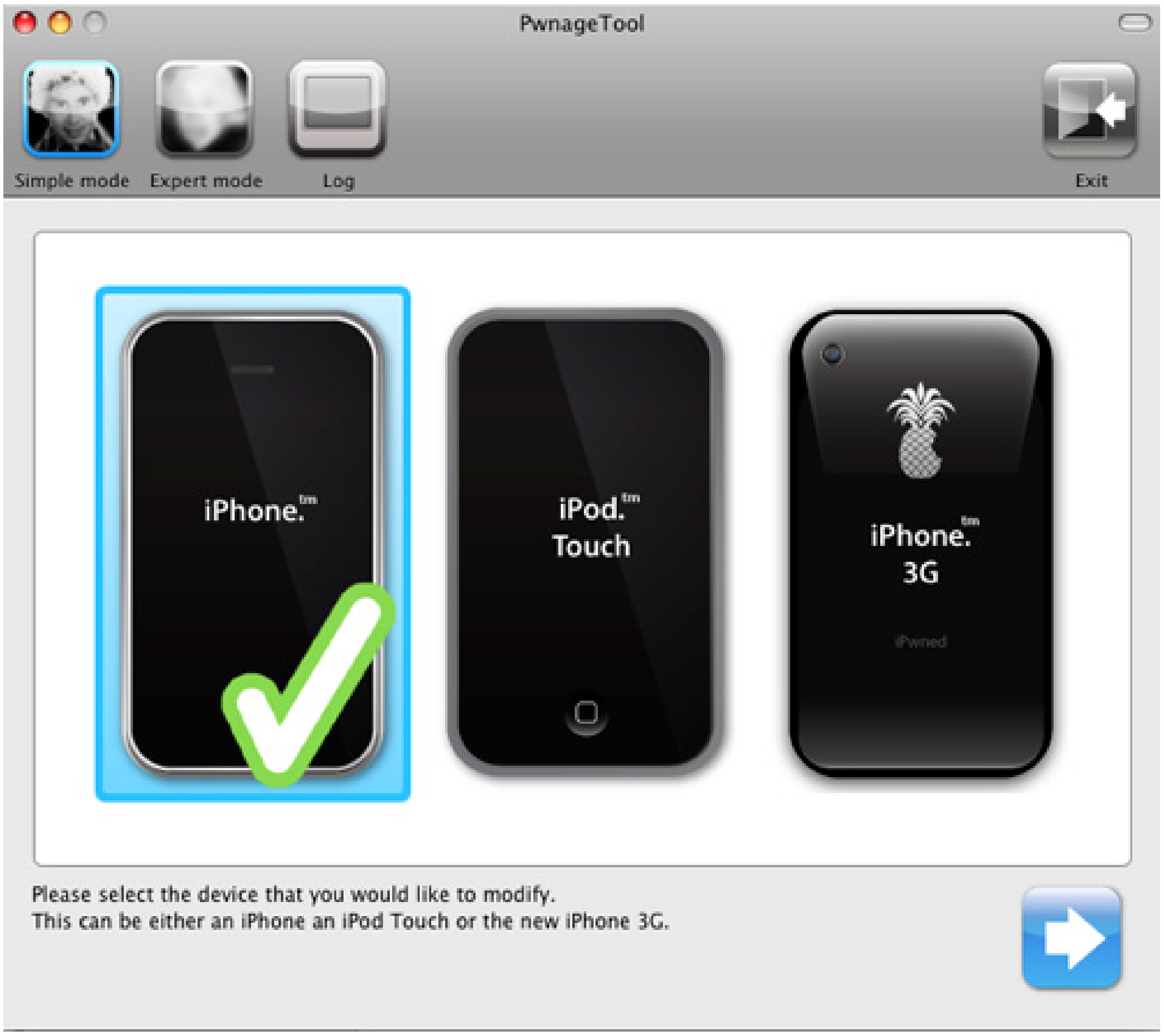
If you own an iPhone, already use it with an Apple-approved service provider account for your country, and only want to run applications of your own development on the device, you need a quick and easy jailbreak solution. Similarly, if you have an iPod Touch, a simple jailbreak tool is all you need.

Jailbreaking your device

QuickPwn performs the jailbreaking task quickly. All that's required is to download the correct version of the iPhone OS for your device (only iPhone OS versions 2.0 through 2.0.2 are supported at the time of this writing). Then, connect it to your computer with the USB charging/data cable and put your iPhone in what is called *DFU mode*, as per the instructions in the software. In less than 10 minutes, your device is *jailbroken.*

If you are using an iPhone in a country where there is currently no Apple-approved cell service provider, or you want to use an iPhone with a provider other than the designated iPhone service provider for your country, the iPhone Dev Team offers Xpwn, Pwnage (see Figure 1), or WinPwn.

Figure 1. Pwnage is flagship tool for jailbreaking and unlocking iPhones



Please select the device that you would like to modify.
This can be either an iPhone an iPod Touch or the new iPhone 3G.

Installation details and pointers to online documentation are in the Resources section. Basically, Xpwn, Pwnage, and WinPwn allow you to create a jailbroken version of the iPhone OS on your computer. Then you use iTunes to download the jailbroken OS into your iPhone. This sounds more complicated than it really is. The tools do all the difficult work and use the same automated mechanisms Apple uses in iTunes to upgrade the iPhone OS.

Preparing for development

After you have jailbroken your device and restarted it, you have a device with a full UNIX command line and several utilities preinstalled to make development easier.

But to tap that command-line and enable development, you must install on the device the OpenSSH secure remote shell client and server, the Linker Identity Editor, and the Respring utility to restart the dashboard. These enable you to transfer your programs to the iPhone and log in to execute commands on the device, sign your code with a bogus key to make the iPhone run it, and reset your dashboard to show your newly copied applications.

Software installation is made simple on jailbroken iPhones through the use of Cydia, a package manager front end for APT. Linux users are probably giddy right now as the previous sentence sinks in: APT package management, the one used by Debian-based Linux distributions including Ubuntu? Yes, indeed, my friends. Yes, indeed.

Before running Cydia, make sure you're connected to either the cell data network or WiFi — preferably the latter, as OpenSSH is a bit hefty. Run Cydia, wait for it to download package manifests from the built-in sources, then click **Search** at the bottom of the Cydia window. Search for **openssh** and install it. Then search for **ldid** and install it. Finally, search for **respring** and install it.

Now that you have installed OpenSSH on your iPhone, you need to secure it. All iPhones come with two user accounts: root and mobile. Both have the same password: **alpine**. Log in to your device and change the password for these two accounts. Make them different from each other — and memorable. Connect your device to your WiFi network if you haven't done so already and find its IP address using the **Settings** application.

Note: The first time you connect to the iPhone through OpenSSH, the device can take a minute or so to appear to respond because it is generating unique system keys. Subsequent connections should be nearly instantaneous.

You're ready to get started creating your own software for the device. The next section shows how to install the toolchain that makes programming the iPhone possible.

Section 4. Installing prerequisite tools

When developers refer to a *toolchain,* they mean the compilers, assemblers, linkers, and other tools that work together to build an executable file for a particular platform from nothing but sources and libraries. The iPhone toolchain used by Apple for its iPhone SDK is actually based on the open source GCC — the same toolchain used for Linux, Sun Solaris, and Berkeley Software Distribution (BSD) development. Because of its open source origins, the base source code for the iPhone SDK is actually available. Members of the iPhone Dev Team have produced instructions and patches to enable developers in other platforms to build and use it. That's what we do next.

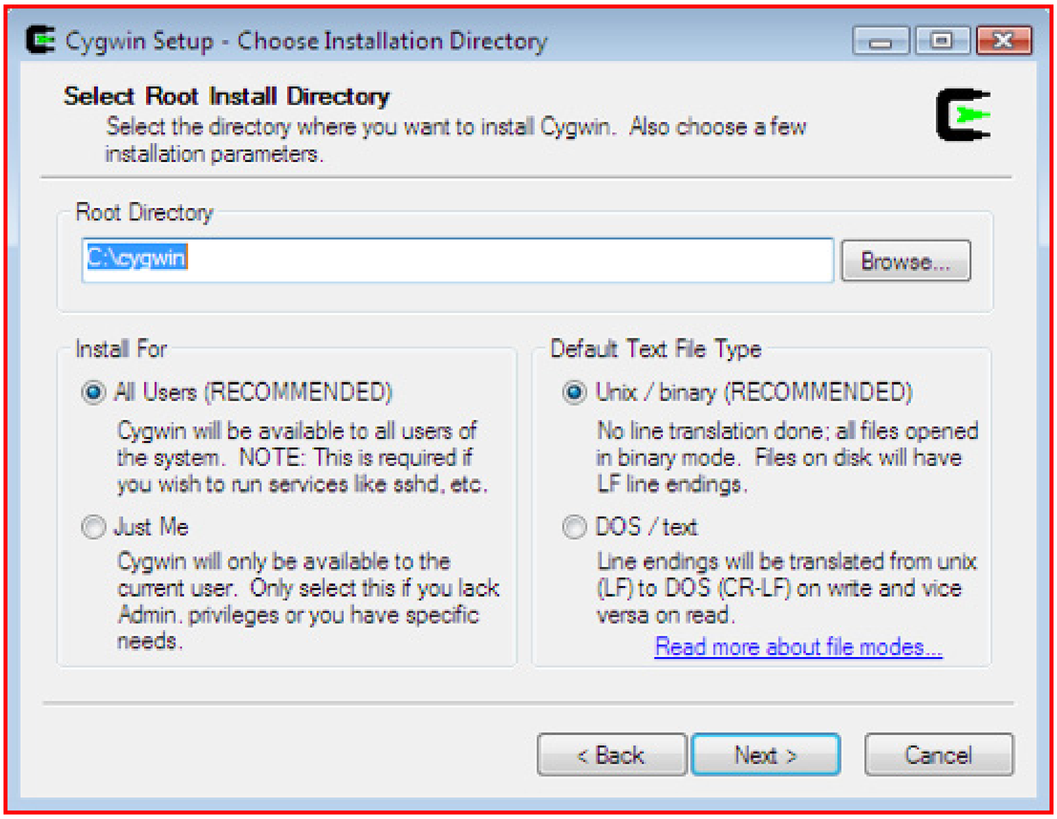
There are a couple of requirements before we begin. You must already have a toolchain installed for the platform you're running. You need tools installed that can convert the GCC source code into executables for your operating system. Depending on your platform, you can install tools for Windows or Linux.

Install prerequisite tools in Windows: Cygwin

Cygwin is a set of Windows programs that creates a UNIX-like environment for Windows. Rather than use virtualization or emulation, the Cygwin team created a Windows dynamic loading library that implements a good deal of the UNIX application program interface (API), then proceeded to patch the source code of many popular UNIX tools until they compiled and ran with sufficient stability on Windows. They have been perfecting that environment for more than 13 years. See Resources for links to the Cygwin site, where you can download the Cygwin setup program. The Cygwin setup program is actually a rather small download because you download the rest of the programs that make up the Cygwin distribution during the last part of the Cygwin installation.

To install and configure Cygwin:

1. Downloaded the Cygwin setup program, run it and click **Next** at every prompt until you come to the window shown below.
   Figure 2. Cygwin root directory selection part of the installation



Cygwin Setup - Choose Installation Directory

Select Root Install Directory

Select the directory where you want to install Cygwin. Also choose a few installation parameters.

Root Directory: C:\cygwin

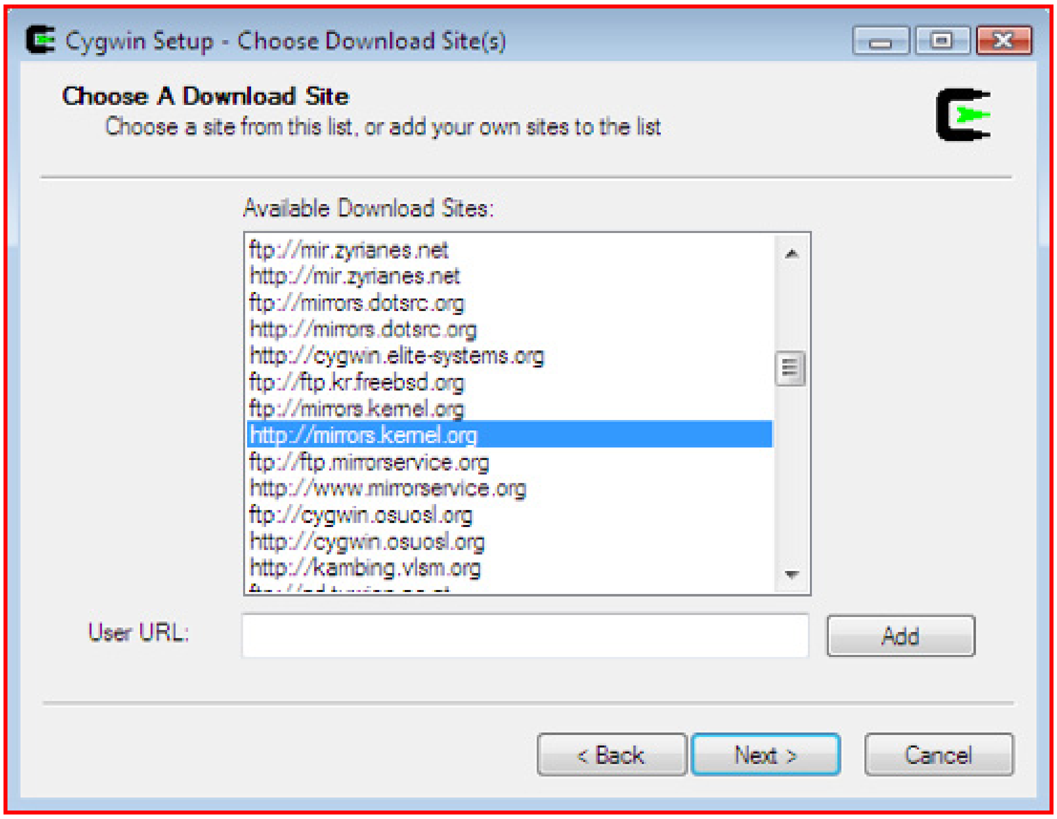
Install For
- (●) All Users (RECOMMENDED)
  Cygwin will be available to all users of the system. NOTE: This is required if you wish to run services like sshd, etc.
- ( ) Just Me
  Cygwin will only be available to the current user. Only select this if you lack Admin. privileges or you have specific needs.

Default Text File Type
- (●) Unix / binary (RECOMMENDED)
  No line translation done; all files opened in binary mode. Files on disk will have LF line endings.
- ( ) DOS / text
  Line endings will be translated from unix (LF) to DOS (CR-LF) on write and vice versa on read.

Read more about file modes...

2. Make sure the options at the bottom of this window match those in Figure 2 or you'll have trouble working with the example code and toolchain sources you download in the next sections of this tutorial. If you install Cygwin to a different directory from the one in Figure 2, you must keep that directory in mind and make modifications to the steps given in the other sections of this tutorial to match your installation choice. I suggest you keep things simple and follow the example.

3. Continue through the Cygwin installation windows, leaving the default choices selected, until you come to the mirror selection screen, shown below.
Figure 3. Cygwin mirror selection part of the installation



Cygwin Setup - Choose Download Site(s)

Choose A Download Site

Choose a site from this list, or add your own sites to the list

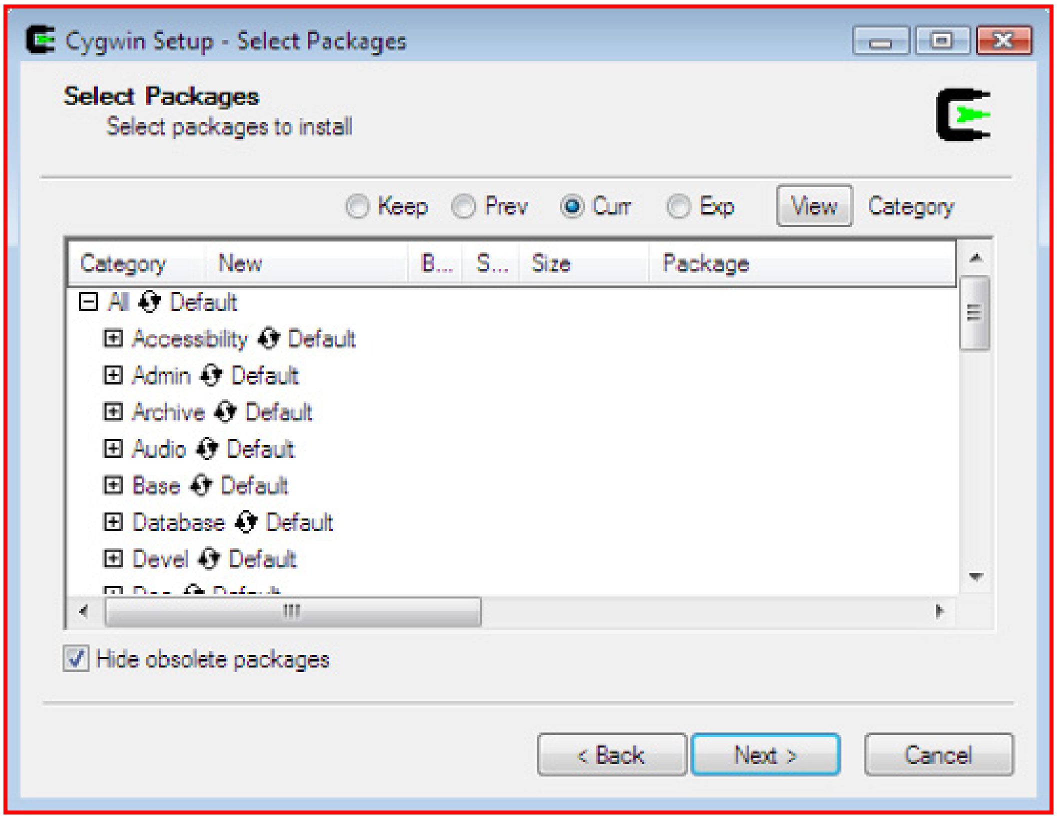
Available Download Sites:

ftp://mir.zyrianes.net
http://mir.zyrianes.net
ftp://mirrors.dotsrc.org
http://mirrors.dotsrc.org
http://cygwin.elite-systems.org
ftp://ftp.kr.freebsd.org
ftp://mirrors.kernel.org
http://mirrors.kernel.org
ftp://ftp.mirrorservice.org
http://www.mirrorservice.org
ftp://cygwin.osuosl.org
http://cygwin.osuosl.org
http://kambing.vlsm.org

User URL:

4. Select a mirror site close to you to distribute the bandwidth use geographically. Try using the mirror at a university in your state or province, for example. If you don't recognize any of the URLs as "local," you can't go wrong choosing either of cygwin.osuosl.org or mirrors.kernels.org. Choosing URLs that begin with HTTP rather than FTP might help if you're having problems with firewalls or proxies within your network.

5. Continue through the installation until you come to the package-selection window, shown below.
Figure 4. Cygwin package-selection part of the installation


Cygwin Setup - Select Packages

Select Packages
Select packages to install

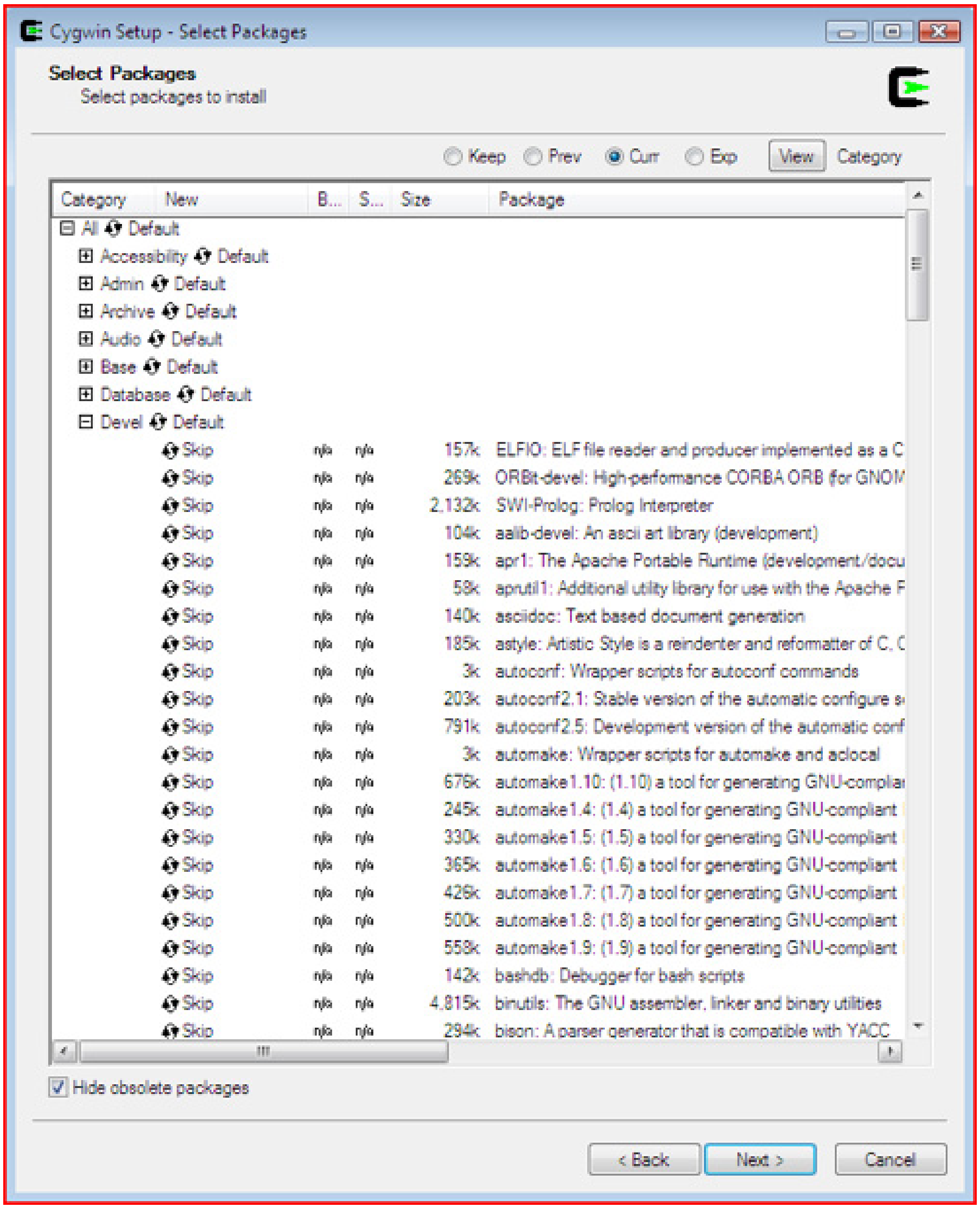
Keep Prev [✓ Curr] Exp View Category

Category	New	B...	S...	Size	Package
All Default					
Accessbility Default					
Admin Default					
Archive Default					
Audio Default					
Base Default					
Database Default					
Devel Default					

[✓] Hide obsolete packages

< Back Next > Cancel

6. Expand the Devel category, shown below.
Figure 5. Expand the Devel category to show the packages within that category



Cygwin Setup - Select Packages

Select Packages
Select packages to install

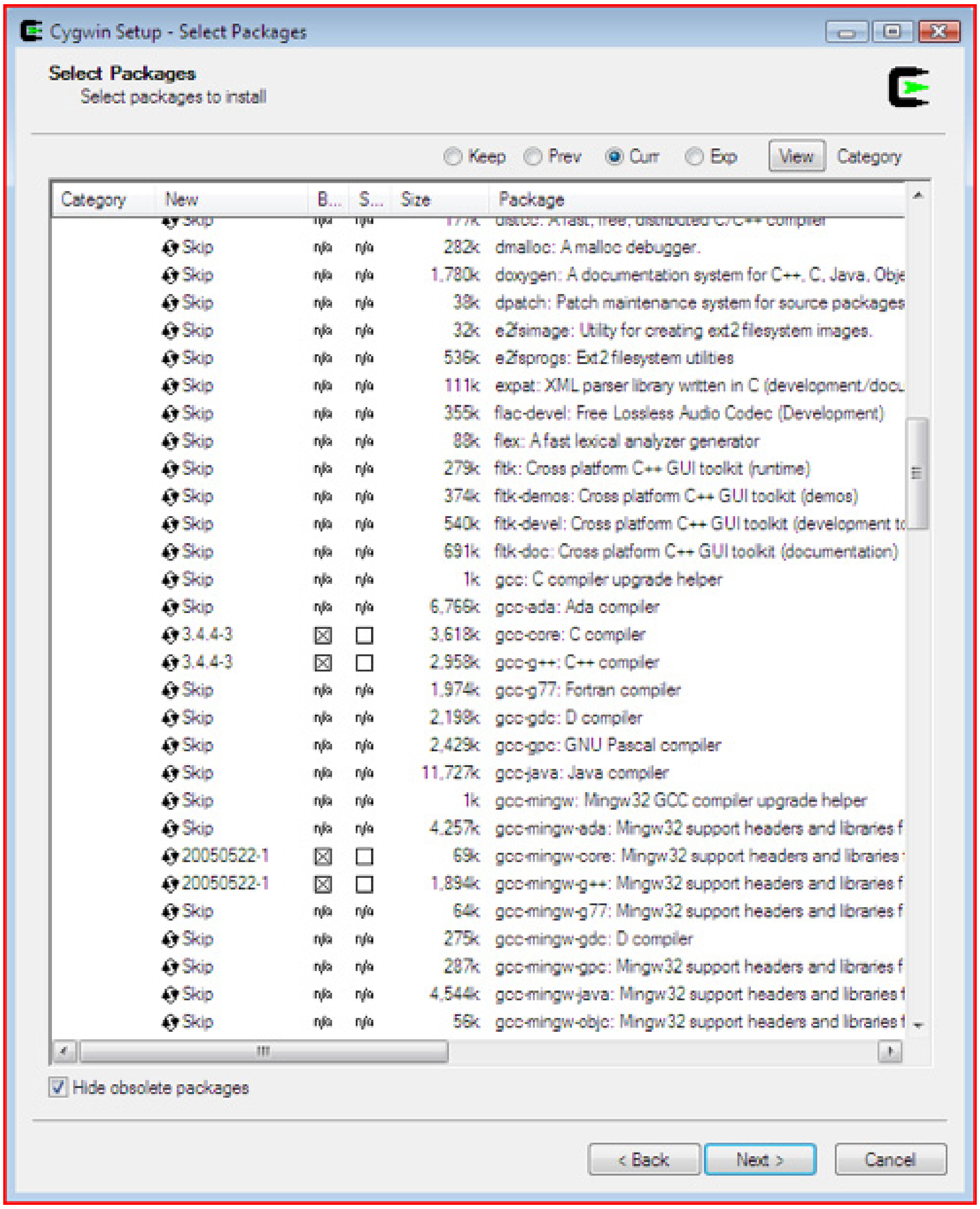
Keep Prev [✓ Curr] Exp View Category

Category	New	B...	S...	Size	Package
⊟ All Default					
⊞ Accessibility Default					
⊞ Admin Default					
⊞ Archive Default					
⊞ Audio Default					
⊞ Base Default					
⊞ Database Default					
⊟ Devel Default					
	Skip	n/a	n/a	157k	ELFIO: ELF file reader and producer implemented as a C
	Skip	n/a	n/a	269k	ORBit-devel: High-performance CORBA ORB for GNOM
	Skip	n/a	n/a	2,132k	SWI-Prolog: Prolog Interpreter
	Skip	n/a	n/a	104k	aalib-devel: An ascii art library (development)
	Skip	n/a	n/a	159k	apr1: The Apache Portable Runtime (development/docu
	Skip	n/a	n/a	58k	aprutil1: Additional utility library for use with the Apache F
	Skip	n/a	n/a	140k	asciidoc: Text based document generation
	Skip	n/a	n/a	185k	astyle: Artistic Style is a reindenter and reformatter of C, C
	Skip	n/a	n/a	3k	autoconf: Wrapper scripts for autoconf commands
	Skip	n/a	n/a	203k	autoconf2.1: Stable version of the automatic configure s
	Skip	n/a	n/a	791k	autoconf2.5: Development version of the automatic conf
	Skip	n/a	n/a	3k	automake: Wrapper scripts for automake and aclocal
	Skip	n/a	n/a	676k	automake1.10: (1.10) a tool for generating GNU-complia
	Skip	n/a	n/a	245k	automake1.4: (1.4) a tool for generating GNU-compliant
	Skip	n/a	n/a	330k	automake1.5: (1.5) a tool for generating GNU-compliant
	Skip	n/a	n/a	365k	automake1.6: (1.6) a tool for generating GNU-compliant
	Skip	n/a	n/a	426k	automake1.7: (1.7) a tool for generating GNU-compliant
	Skip	n/a	n/a	500k	automake1.8: (1.8) a tool for generating GNU-compliant
	Skip	n/a	n/a	558k	automake1.9: (1.9) a tool for generating GNU-compliant
	Skip	n/a	n/a	142k	bashdb: Debugger for bash scripts
	Skip	n/a	n/a	4,815k	binutils: The GNU assembler, linker and binary utilities
	Skip	n/a	n/a	294k	bison: A parser generator that is compatible with YACC

[✓] Hide obsolete packages

< Back Next > Cancel

7. For the autoconf, binutils, bison, flex, gcc-c++, and make packages, click **Skip** to the left of the package you want to include in the installation, as shown below. Clicking **Skip** once selects the **Binary** column check box and the version number of the package to show where the "skip" was. Note that the actual version numbers may differ from what is shown because packages are updated continuously.
Figure 6. Select the packages to include in the installation



Cygwin Setup - Select Packages

Select Packages
Select packages to install

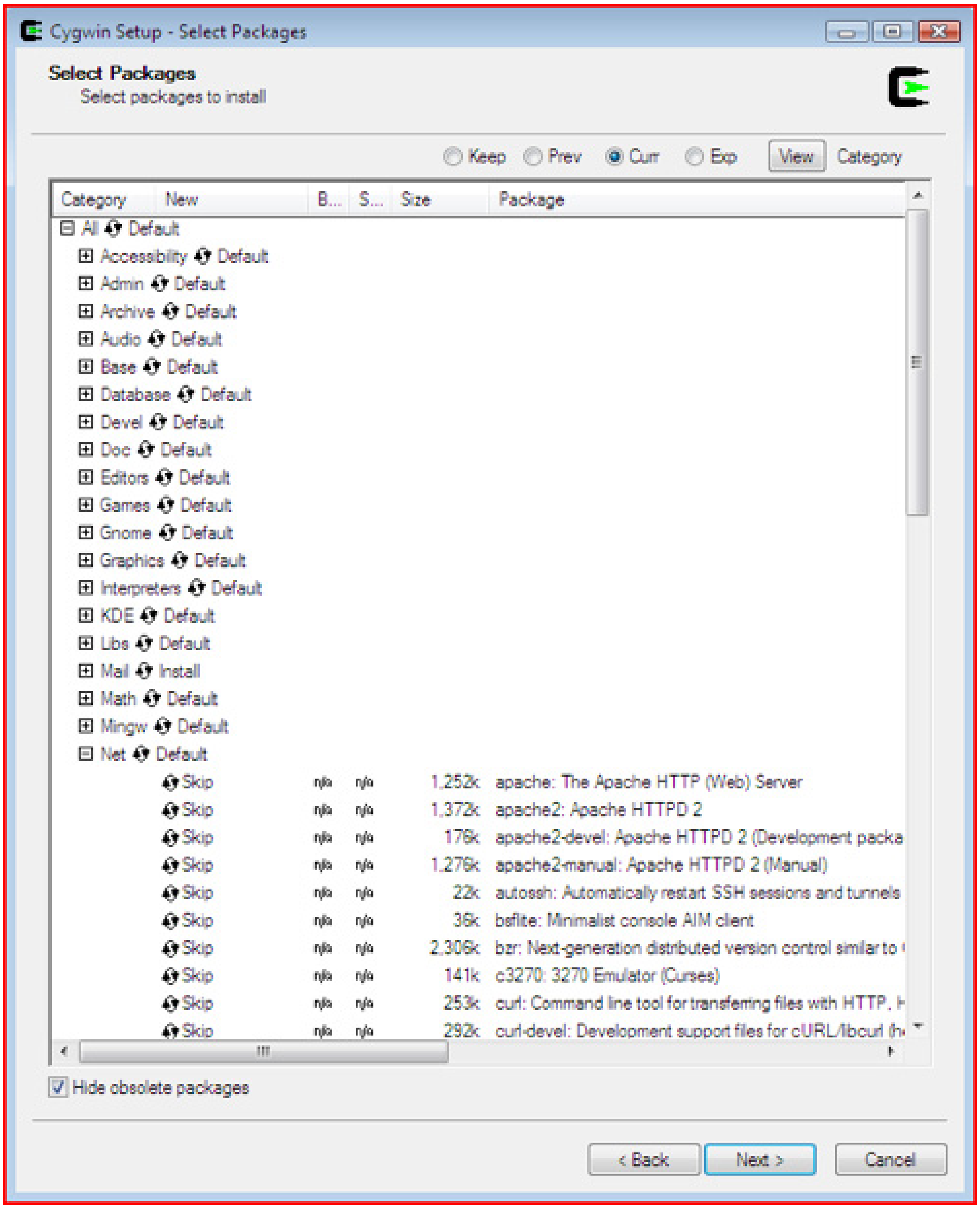
○ Keep ○ Prev [✓ ● Curr] ○ Exp View Category

Category	New	B...	S...	Size	Package
	Skip	n/a	n/a	177k	distcc: A fast, free, distributed C/C++ compiler
	Skip	n/a	n/a	282k	dmalloc: A malloc debugger.
	Skip	n/a	n/a	1,780k	doxygen: A documentation system for C++, C, Java, Obje
	Skip	n/a	n/a	38k	dpatch: Patch maintenance system for source packages
	Skip	n/a	n/a	32k	e2fsimage: Utility for creating ext2 filesystem images.
	Skip	n/a	n/a	536k	e2fsprogs: Ext2 filesystem utilities
	Skip	n/a	n/a	111k	expat: XML parser library written in C (development/docu
	Skip	n/a	n/a	355k	flac-devel: Free Lossless Audio Codec (Development)
	Skip	n/a	n/a	88k	flex: A fast lexical analyzer generator
	Skip	n/a	n/a	279k	fltk: Cross platform C++ GUI toolkit (runtime)
	Skip	n/a	n/a	374k	fltk-demos: Cross platform C++ GUI toolkit (demos)
	Skip	n/a	n/a	540k	fltk-devel: Cross platform C++ GUI toolkit (development to
	Skip	n/a	n/a	691k	fltk-doc: Cross platform C++ GUI toolkit (documentation)
	Skip	n/a	n/a	1k	gcc: C compiler upgrade helper
	Skip	n/a	n/a	6,766k	gcc-ada: Ada compiler
	3.4.4-3	☒	☐	3,618k	gcc-core: C compiler
	3.4.4-3	☒	☐	2,958k	gcc-g++: C++ compiler
	Skip	n/a	n/a	1,974k	gcc-g77: Fortran compiler
	Skip	n/a	n/a	2,198k	gcc-gdc: D compiler
	Skip	n/a	n/a	2,429k	gcc-gpc: GNU Pascal compiler
	Skip	n/a	n/a	11,727k	gcc-java: Java compiler
	Skip	n/a	n/a	1k	gcc-mingw: Mingw32 GCC compiler upgrade helper
	Skip	n/a	n/a	4,257k	gcc-mingw-ada: Mingw32 support headers and libraries f
	20050522-1	☒	☐	69k	gcc-mingw-core: Mingw32 support headers and libraries
	20050522-1	☒	☐	1,894k	gcc-mingw-g++: Mingw32 support headers and libraries f
	Skip	n/a	n/a	64k	gcc-mingw-g77: Mingw32 support headers and libraries f
	Skip	n/a	n/a	275k	gcc-mingw-gdc: D compiler
	Skip	n/a	n/a	287k	gcc-mingw-gpc: Mingw32 support headers and libraries f
	Skip	n/a	n/a	4,544k	gcc-mingw-java: Mingw32 support headers and libraries f
	Skip	n/a	n/a	56k	gcc-mingw-objc: Mingw32 support headers and libraries f

☑ Hide obsolete packages

< Back Next > Cancel

8. Scroll down, and expand Net, as shown below.
Figure 7. Expand the Net category to show its packages



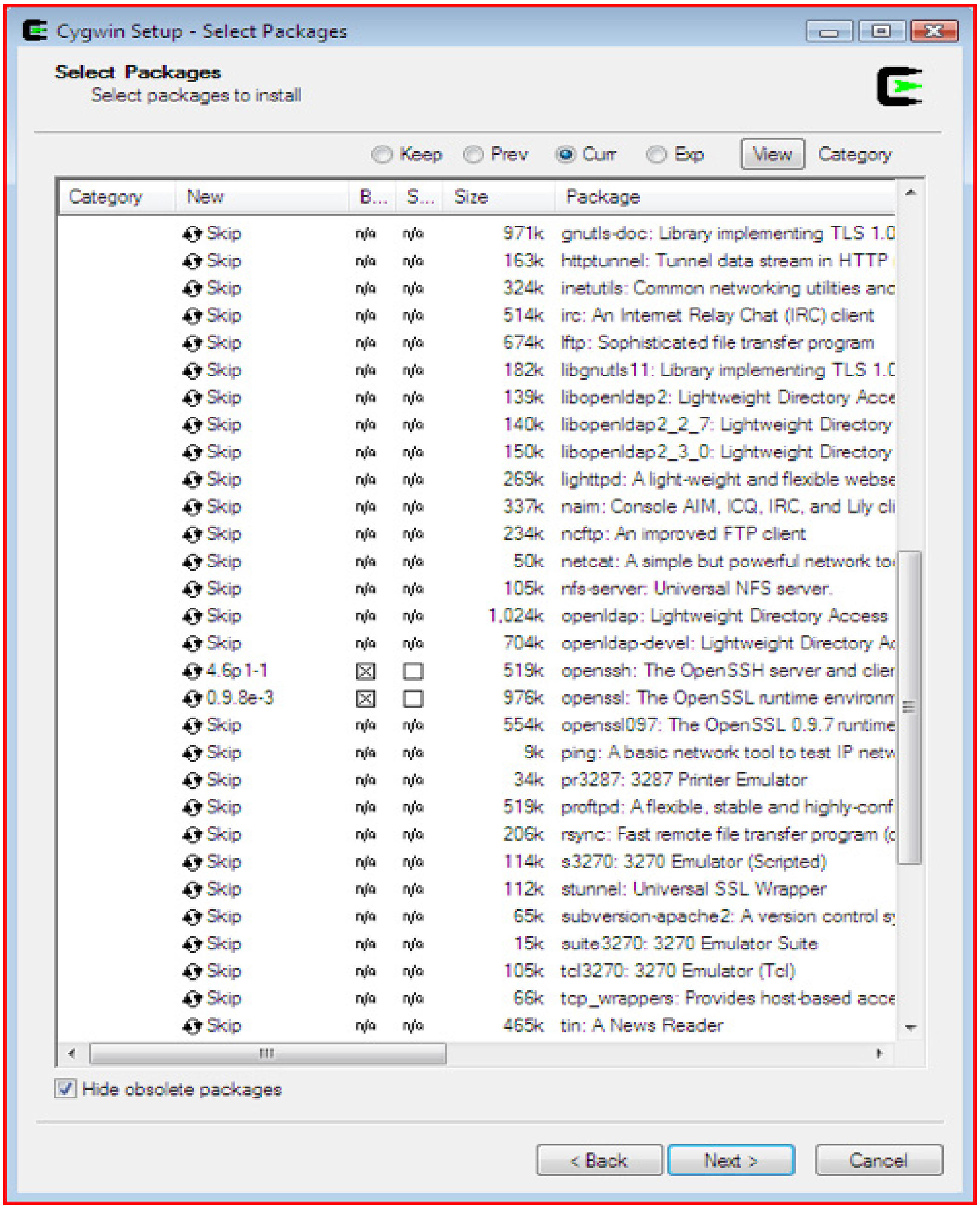
Select Packages
Select packages to install

( ) Keep ( ) Prev [✓ Curr] ( ) Exp View Category

Category	New	B...	S...	Size	Package
⊟ All ♻ Default					
⊞ Accessibility ♻ Default					
⊞ Admin ♻ Default					
⊞ Archive ♻ Default					
⊞ Audio ♻ Default					
⊞ Base ♻ Default					
⊞ Database ♻ Default					
⊞ Devel ♻ Default					
⊞ Doc ♻ Default					
⊞ Editors ♻ Default					
⊞ Games ♻ Default					
⊞ Gnome ♻ Default					
⊞ Graphics ♻ Default					
⊞ Interpreters ♻ Default					
⊞ KDE ♻ Default					
⊞ Libs ♻ Default					
⊞ Mail ♻ Install					
⊞ Math ♻ Default					
⊞ Mingw ♻ Default					
⊟ Net ♻ Default					
	♻ Skip	n/a	n/a	1,252k	apache: The Apache HTTP (Web) Server
	♻ Skip	n/a	n/a	1,372k	apache2: Apache HTTPD 2
	♻ Skip	n/a	n/a	176k	apache2-devel: Apache HTTPD 2 (Development packa
	♻ Skip	n/a	n/a	1,276k	apache2-manual: Apache HTTPD 2 (Manual)
	♻ Skip	n/a	n/a	22k	autossh: Automatically restart SSH sessions and tunnels
	♻ Skip	n/a	n/a	36k	bsflite: Minimalist console AIM client
	♻ Skip	n/a	n/a	2,306k	bzr: Next-generation distributed version control similar to
	♻ Skip	n/a	n/a	141k	c3270: 3270 Emulator (Curses)
	♻ Skip	n/a	n/a	253k	curl: Command line tool for transferring files with HTTP, H
	♻ Skip	n/a	n/a	292k	curl-devel: Development support files for cURL/libcurl (h

[✓] Hide obsolete packages

9. Select the openssh and openssl packages, shown in Figure 8.
Figure 8. Select the openssh and openssl packages


Cygwin Setup - Select Packages

Select Packages
Select packages to install

○ Keep ○ Prev [✓ Curr] ○ Exp　View　Category

Category	New	B...	S...	Size	Package
	Skip	n/a	n/a	971k	gnutls-doc: Library implementing TLS 1.0
	Skip	n/a	n/a	163k	httptunnel: Tunnel data stream in HTTP
	Skip	n/a	n/a	324k	inetutils: Common networking utilities and
	Skip	n/a	n/a	514k	irc: An Internet Relay Chat (IRC) client
	Skip	n/a	n/a	674k	lftp: Sophisticated file transfer program
	Skip	n/a	n/a	182k	libgnutls11: Library implementing TLS 1.0
	Skip	n/a	n/a	139k	libopenldap2: Lightweight Directory Acce
	Skip	n/a	n/a	140k	libopenldap2_2_7: Lightweight Directory
	Skip	n/a	n/a	150k	libopenldap2_3_0: Lightweight Directory
	Skip	n/a	n/a	269k	lighttpd: A light-weight and flexible webse
	Skip	n/a	n/a	337k	naim: Console AIM, ICQ, IRC, and Lily cli
	Skip	n/a	n/a	234k	ncftp: An improved FTP client
	Skip	n/a	n/a	50k	netcat: A simple but powerful network to
	Skip	n/a	n/a	105k	nfs-server: Universal NFS server.
	Skip	n/a	n/a	1,024k	openldap: Lightweight Directory Access
	Skip	n/a	n/a	704k	openldap-devel: Lightweight Directory Ac
	4.6p1-1	☒	☐	519k	openssh: The OpenSSH server and clier
	0.9.8e-3	☒	☐	976k	openssl: The OpenSSL runtime environm
	Skip	n/a	n/a	554k	openssl097: The OpenSSL 0.9.7 runtime
	Skip	n/a	n/a	9k	ping: A basic network tool to test IP netw
	Skip	n/a	n/a	34k	pr3287: 3287 Printer Emulator
	Skip	n/a	n/a	519k	proftpd: A flexible, stable and highly-conf
	Skip	n/a	n/a	206k	rsync: Fast remote file transfer program (c
	Skip	n/a	n/a	114k	s3270: 3270 Emulator (Scripted)
	Skip	n/a	n/a	112k	stunnel: Universal SSL Wrapper
	Skip	n/a	n/a	65k	subversion-apache2: A version control sy
	Skip	n/a	n/a	15k	suite3270: 3270 Emulator Suite
	Skip	n/a	n/a	105k	tcl3270: 3270 Emulator (Tcl)
	Skip	n/a	n/a	66k	tcp_wrappers: Provides host-based acce
	Skip	n/a	n/a	465k	tin: A News Reader

☑ Hide obsolete packages

< Back　Next >　Cancel

10. Click **Next**. Cygwin setup downloads the packages from the mirror site and installs itself at the directory specified earlier. This process can take upwards of an hour, even on 2-megabit broadband and faster, because those mirrors are in high demand, not only for Cygwin packages but for other open source software they mirror.

When Cygwin finishes downloading and installing packages, you can choose whether you want Cygwin to install shortcut icons on the desktop or in the Start menu, and you're done. You have just installed the GCC toolchain and a complete

UNIX-like environment that runs natively on Windows. You can now go directly to "installing the iPhone toolchain from source."

Install prerequisite tools in Linux

Good news for Gentoo Linux users: You more than likely have everything you need already installed and can move on to "installing the iPhone toolchain from source." If you're using Debian, Ubuntu, or any distributions based on those two, you can install the software needed to compile the iPhone toolchain by installing the bison, flex, build-essentials, and autoconf packages using the `apt-get` command:

```
sudo apt-get install build-essential bison flex autoconf
```

If you're using SuSE or Red Hat Linux or any derivative of these, such as CentOS, you need to install autoconf, binutils, bison, gcc, g++, flex, and make with `yum`.

```
sudo yum install bison flex autoconf binutils gcc gcc-c++ make
```

If you're using any other Linux distribution, consult your documentation for instructions on installing packages. You want to install the packages listed for `yum`-based systems. Be aware that gcc-c++ may be called something else on your system or may be included with GCC. When the requisite packages are installed, you can move on to "installing the iPhone toolchain from source."

Section 5. Installing the iPhone toolchain from source

Download the toolchain source code

Why build the toolchain from source rather than downloading a binary? In a word: licensing. The source code contains few distribution limitations, being open source. The iPhone Dev Team based the toolchain on source code found in public repositories and modified the code so it would compile on Linux and Cygwin.

I packaged the source code needed to compile the toolchain and provide a link in the Resources section. Download that file and copy it to the home folder of your user account. If you're using Cygwin, copy it to C:/cygwin/home/*yourusername* if you installed Cygwin to the recommended directory. Extract the package with your favorite gzipped tar-compatible tool, and you'll find a folder named iphone-2.0-toolchain in your home directory.

Download the iPhone firmware from Apple

Next, obtain the iPhone firmware and add it to the toolchain source code. The Apple code you need is the iPhone V2.0 firmware. It matters which version you obtain: You want the iPhone OS V2.0 firmware for the first-generation iPhone, with the file name iPhone1,1_2.0_5A347_Restore.ipsw. Refer to Resources for any of the Pwnage or WinPwn how-to articles for the link from which to download it.

The firmware restore file is nothing more than a ZIP file. After you have the firmware, change the extension to .zip and open it with any extraction utility for your platform. Inside the ZIP file, you'll find a file called *018-3785-2.dmg.* Copy this DMG file to the home folder of your user account, right next to the iphone-2.0-toolchain folder from earlier. (Windows users: We refer to the Cygwin user home folder, not your Windows user home folder.)

Mount the operating system image

This DMG file is an actual encrypted image of the operating system partition on the device. You must decrypt this file before you can mount the image and extract the files you need. Luckily, the iPhone Dev Team wrote a tool to decrypt the partition image and have obtained and published the encryption key. The next step is to compile the image decryptor and use it to obtain a decrypted image you can mount. Open a terminal session (in Windows, use the Cygwin shortcut, not the Windows command line), and type the following :

```
gcc ~/iphone-2.0-toolchain/src/vfdecrypt.c -o ~/vfdecrypt -lcrypto
```

This command compiles the DMG decryptor code to an executable file named vfdecrypt on your home folder. Now you can decrypt the DMG file.

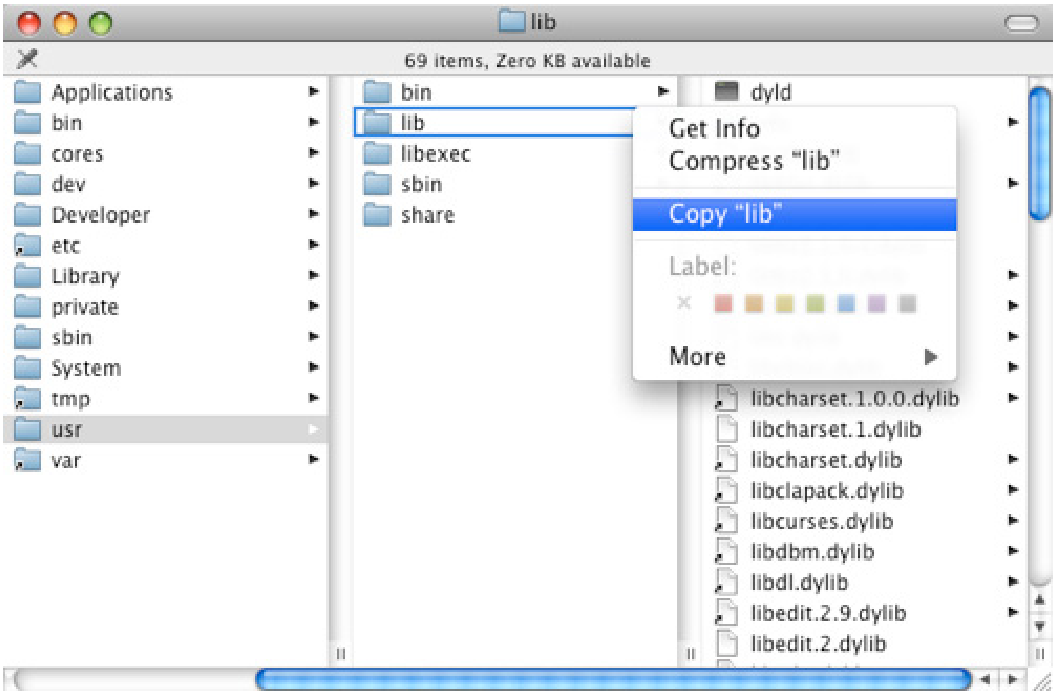
```
~/vfdecrypt -i ~/018-3785-2.dmg -o ~/decrypted.dmg -k \
2cfca55aabb22fde7746e6a034f738b7795458be9902726002a8341995558990f41e3755
```

Notice that this is two separate lines, and don't miss that backslash (\) at the end of the first line. It is really a single command, but we split it into two lines with the backslash to avoid ambiguity caused by the sheer length of the command. You may want to paste the two lines in rather than manually enter the monstrous decryption key.

Now that you've decrypted the DMG, mount it, and extract the contents. Linux users can install HFS drivers and mount the DMG. It's OK if your distro only has read-only HFS drivers, as you just want to copy some files. Windows users can use either PowerISO or Transmac (see Resources) to open the DMG file and copy the files you want.

Copy the required folders

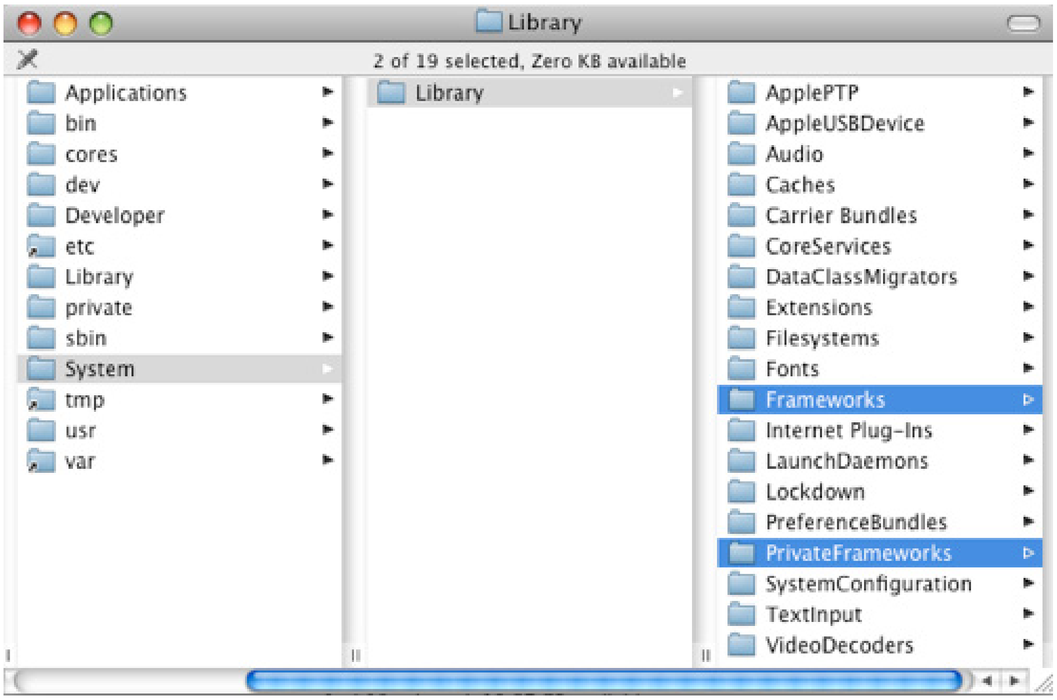
Now that you've mounted or opened the decrypted DMG, you must copy three folders from there. Assume that *decrypted.dmg* refers to the mounted volume, regardless of whether you've mounted it on Linux or opened the file with Transmac or PowerISO. The folders are *decrypted.dmg/usr/lib, decrypted.dmg/System/Library/Frameworks*, and *decrypted.dmg/System/Library/PrivateFrameworks*. Inside the iphone-2.0-toolchain folder are two folders: iphone-fs/System/Library/ and iphone-fs/usr/. Copy the lib folder to iphone-fs/usr/. Copy Frameworks and PrivateFrameworks to iphone-fs/System/Library/. The steps below help clarify which folders you need to copy and where to paste them:

1. Copy the lib folder to your desktop temporarily, as shown in Figure 9. (You'll move it where it belongs shortly.)
Figure 9. Copy the lib folder to your desktop


2. Copy the Frameworks and PrivateFrameworks folders to your desktop temporarily, as shown below.

3. Copy the lib folder to your desktop temporarily, as shown in Figure 9. (You'll move it where it belongs shortly.)

   Figure 10. Copy the Frameworks and PrivateFrameworks folders

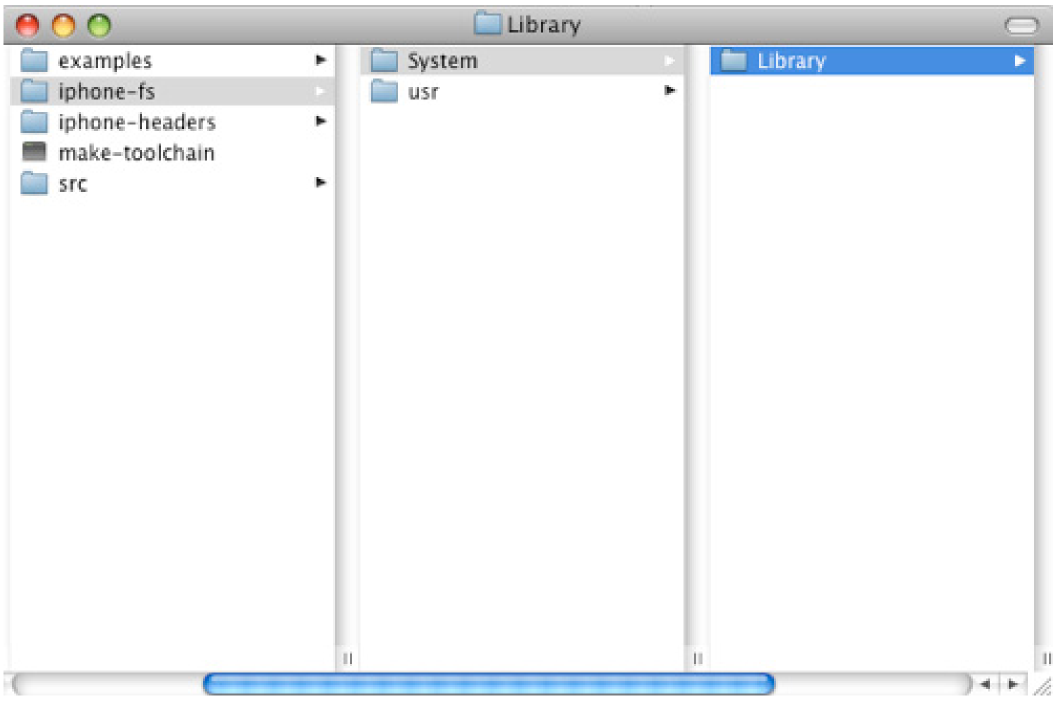


4. Move the lib folder from your desktop into the iphone-fs/usr folder.

5. Move Frameworks and PrivateFrameworks together into the iphone-fs//System/Library folder, as shown below.

6. Copy the lib folder to your desktop temporarily, as shown in Figure 9. (You'll move it where it belongs shortly.)

   Figure 11. lib folder goes into lib folder and the Frameworks and PrivateFrameworks folders go into /System/Library



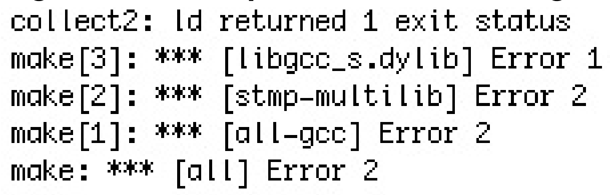
Compile the toolchain

After you have copied the folders to their destinations, you can unmount the DMG or close Transmac or PowerISO. Now, open a terminal and type the following commands:

```
cd ~/iphone-2.0-toolchain
./make-toolchain
```

These commands finally start the process of compiling the iPhone toolchain and installing it to /usr/local on your system. The process takes about two hours on a 2.0-GHz Core 2 Duo MacBook running Microsoft Windows XP inside VMware with 640 MB of RAM allocated to the virtual machine (VM). After a wait of a few minutes, you'll see a lot of gibberish scroll by very quickly. Don't worry: That means it's working! The only time you should be concerned is if it stops and returns to the shell prompt after displaying several error lines, such as those below.

Figure 12. Example error lines during the make process



```
collect2: ld returned 1 exit status
make[3]: *** [libgcc_s.dylib] Error 1
make[2]: *** [stmp-multilib] Error 2
make[1]: *** [all-gcc] Error 2
make: *** [all] Error 2
```

If the make process stops but you don't see any error lines like those in Figure 12, you have successfully completed the compilation process. Now that you have built the toolchain, you're ready to get started creating your own software for the iPhone. Next, you must install Eclipse CDT and you can explore how it lets you program C and C++ within Eclipse.

Section 6. C and C++ development with Eclipse CDT

Install Eclipse CDT and other plug-ins

To work with Objective-C in Eclipse, you need to install two things: Eclipse CDT (to manage makefiles and your project files) and an Eclipse editor plug-in (to give you Objective-C syntax highlighting). The Color Editor plug-in offers syntax highlighting for more than 140 languages.

If you don't already have Eclipse or if you don't feel like integrating all these plug-ins into your installation of Eclipse, check out the EasyEclipse project (see Resources) and download the EasyEclipse for the C and C++ distribution. EasyEclipse includes the Eclipse, Eclipse CDT, Subclipse, and CVS plug-ins for managing your source revisions, as well as several plug-ins for syntax highlighting and editor default handing of different languages, including Objective-C. Find all of this in one simple, single download for Windows, Linux, and Mac OS X.

If you already have Eclipse installed and have no problems adding plug-ins manually, you can add both to your copy of Eclipse easily. To install the Color Editor, download the plug-in and drop the JAR into your Eclipse plugins directory. To install the Eclipse CDT, go to the Eclipse CDT download page and find the correct URL for your version of Eclipse. In Eclipse, go to **Help > Software Update > Find and Install... > Search for new features to install > New Remote Site...** and enter the correct CDT URL for your version of Eclipse. Proceed with the installation and restart Eclipse (see Resources for the CDT and Color Editor).

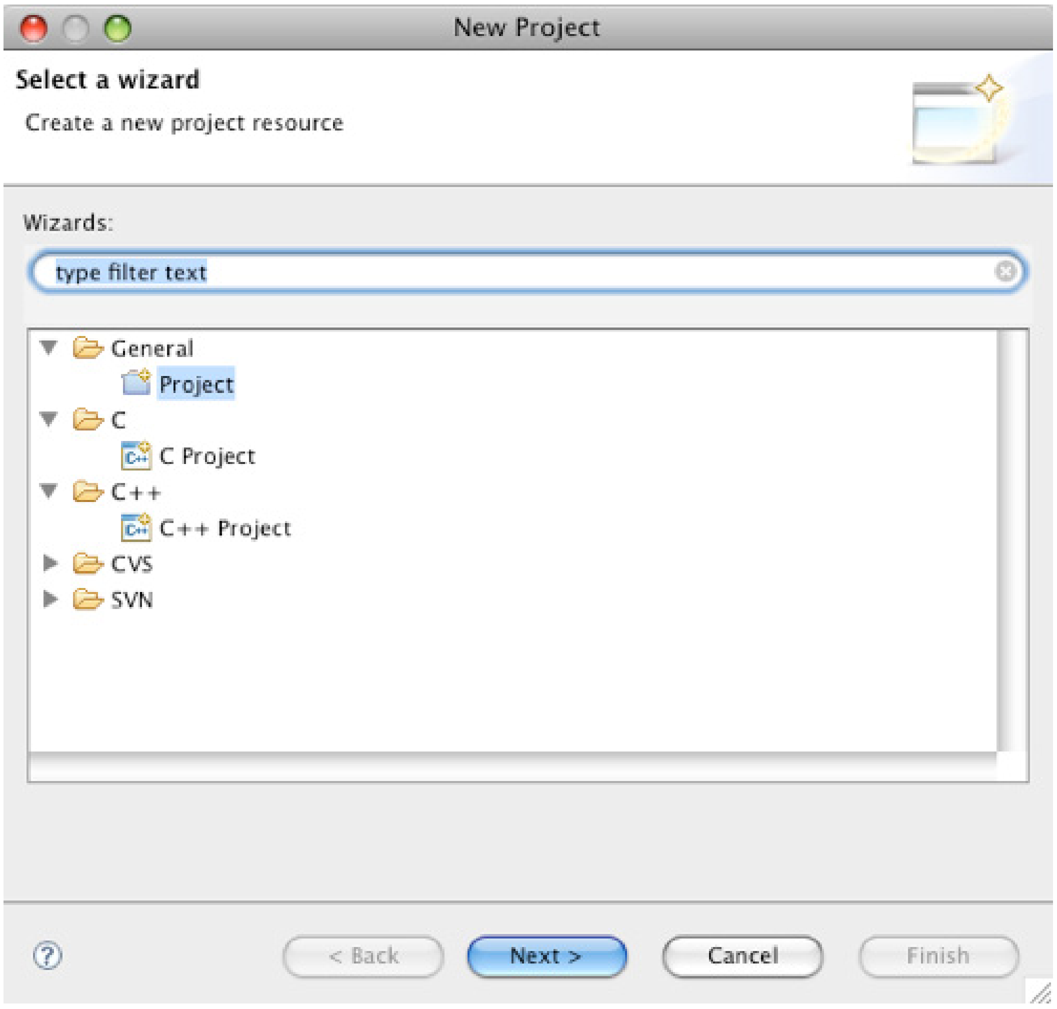
Create an Eclipse CDT project

After installing Eclipse CDT and the Color Editor, either by installing EasyEclipse or manually adding the plug-ins, you can begin to program in C or C++.

Begin by creating a simple project for Eclipse CDT:

1. Run Eclipse. From the workbench, choose **File > New > Project** to bring up the **New Project** window.

2. Choose **General > Project**, and then click **Next**, as shown below.
Figure 13. The New Project window



3. Name the project `HelloWorld`.

4. If you're using Windows, change the default location to C:/cygwin/home/*yourusername*/HelloWorld, then click **Finish**.
The new project appears in your Project Explorer on the workbench. Click **Yes** if Eclipse asks you to change to the C/C++ Perspective.

5. Create two files inside the project: Makefile and HelloWorld.c.

6. Copy the code in Listing 1 into Makefile.

Listing 1. Contents of Makefile

```
CC=gcc
CXX=g++
LD=$(CC)

all:    HelloWorld

HelloWorld:    HelloWorld.o
    $(LD) $(LDFLAGS) -o $@ $^

%.o:    %.c
    $(CC) -c $(CFLAGS) $(CPPFLAGS) $< -o $@

clean:
    rm *.o HelloWorld
```

7. Copy the code in Listing 2 into HelloWorld.c.
   Listing 2. Contents of HelloWorld.c

```
#include <stdio.h>

int main(void) {
  printf("Hello, world!\n");
  return 0;
}
```

Run the project

Now you can run this project. Open a terminal, and change directories to where your project was created. From there, type the `make` command and check for make errors. If you created the Makefile and HelloWorld.c files correctly, there should be no errors, and the output produced in the console should just be two lines.

```
gcc -c   HelloWorld.c -o HelloWorld.o
gcc  -o HelloWorld HelloWorld.o
```

Typing `make` at the command line causes a program called *make* to interpret the makefile, which effectively tells `make` that it should compile HelloWorld.c into an object file called *HelloWorld.o*, link HelloWorld.o implicitly with the standard C libraries, and produce an executable file called HelloWorld. That is the meaning of the two commands produced as console output by `make`.

If you have programmed in C on UNIX before, this is probably not exciting news. However, it's wise to verify that the simple C code and makefile work correctly by executing the compiled and linked program with the following command:

```
./HelloWorld
```

This should produce the output "Hello, world!" in the console. Now, modify the makefile and produce an executable file that runs on the iPhone using your new toolchain.

Create an executable file for the iPhone

To create such an executable file, you need to overwrite the first two lines of the makefile with the following code:

```
CC=/usr/local/bin/arm-apple-darwin9-gcc
CXX=/usr/local/bin/arm-apple-darwin9-g++
```

This command tells make to use your new iPhone toolchain, rather than your system's toolchain you used earlier. Type `make clean` at the command line, and the make program runs the `clean` rule at the bottom, which deletes the executable file and object file produced earlier by the previous version of the makefile.

Now you can type `make`, and the new toolchain is invoked, rather than the system toolchain. This produces a HelloWorld.o object file and HelloWorld executable file for the iPhone. Copy it to your device to see whether your toolchain really works. Connect your device to your WiFi network, find out its IP address in the Settings application and send the HelloWorld executable file to it with a command like the following (substitute *IPHONE_IP* with the IP address for your device).

```
scp HelloWorld root@IPHONE_IP:~/
```

Execute the program

Now, log in through `ssh` to execute the program. Substitute *IPHONE_IP* again with your device's IP address.

```
ssh root@IPHONE_IP
```

When you've logged in, you're at the root user prompt on the iPhone.

```
./HelloWorld
```

And I bet you're disappointed by the "Killed" output in the console. What happened is that the iPhone executable loader expects all executables to be signed. The iPhone Dev Team did manage to remove nearly all restrictions from the iPhone OS V2.0, but they didn't want to mess too much with the executable loader. All is not lost: You can use the `ldid` command from the Linker Identity Editor utility you installed with Cydia earlier. The `ldid` command produces a bogus executable signature to fool the executable loader.

```
ldid -S HelloWorld
./HelloWorld
```

This time, it works, and the crowd roars its approval. You've successfully run code on your device that you wrote and compiled yourself just minutes ago. You reach the only logical conclusion a budding iPhone programmer can reach at this point: Let's put a GUI on it!

Section 7. Compiling and building a Cocoa Touch iPhone native application bundle

Assuming that your earlier experiments have worked, this is now where the rubber meets the road. You will create a simple GUI application that can be launched by touching its icon from the iPhone's touchscreen.

Modify your project makefile

You start creating the GUI application by modifying the makefile for your project and writing some Objective-C code. To keep things simple, there's sample program code in the iphone-2.0-toolchain folder in examples/GUI/HelloWorldiPhone. Grab the makefile and all the other files in that folder and overwrite your Eclipse project files with it. Delete HelloWorld.c, which you replaced with HelloWorld.m, in Objective-C.

Open the new makefile in Eclipse, shown in Listing 3, and examine it.

Listing 3. The new make file

```
CC=/usr/local/bin/arm-apple-darwin9-gcc
CXX=/usr/local/bin/arm-apple-darwin9-g++
LD=$(CC)

CFLAGS=-I/usr/local/lib/gcc/arm-apple-darwin9/4.2.1/include \
    -isysroot /usr/local/iphone-sysroot
```

```
LDFLAGS=-framework CoreFoundation -framework Foundation -framework UIKit \
    -lobjc -bind_at_load -isysroot /usr/local/iphone-sysroot

all:    HelloWorld.app

HelloWorld.app:    HelloWorld Info.plist
    mkdir -p HelloWorld.app
    cp Info.plist HelloWorld Default.png icon.png HelloWorld.app/

HelloWorld:    HelloWorld.o HelloWorldApp.o
    $(LD) $(LDFLAGS) -o $@ $^

%.o:    %.m
    $(CC) -c $(CFLAGS) $(CPPFLAGS) $< -o $@

clean:
    rm -rf *.o HelloWorld HelloWorld.app
```

This makefile version adds `CFLAGS` and `LDFLAGS`, which tell the compiler and linker where to find its include and library files. Notice the `-framework` linker parameters. These tell the linker to use the frameworks you grabbed from the iPhone firmware when preparing your iPhone toolchain.

The CC compilation rule has also changed. You're no longer compiling .c files to .o files; rather, you now concern yourself with .m files. As you probably inferred when I said that HelloWorld.m substitutes HelloWorld.c, *.m* (which stands for module) is the source file for Objective-C programs.

There is an extra rule for something called HelloWorld.app. From looking at the commands below the rule, it seems to be a folder in which you pile the executable and some other files, like Info.plist. Listing 4 shows that file.

Listing 4. Info.plist

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
 "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>CFBundleDevelopmentRegion</key>
    <string>en</string>
    <key>CFBundleExecutable</key>
    <string>HelloWorld</string>
    <key>CFBundleIdentifier</key>
    <string>com.pjtrix.helloworld</string>
    <key>CFBundleInfoDictionaryVersion</key>
    <string>0.1</string>
    <key>CFBundleName</key>
    <string>HelloWorld</string>
    <key>CFBundlePackageType</key>
    <string>APPL</string>
    <key>CFBundleShortVersionString</key>
    <string>1.0.0</string>
    <key>CFBundleSignature</key>
    <string>????</string>
    <key>CFBundleVersion</key>
    <string>1.0</string>
</dict>
</plist>
```

The Info.plist file is used by the iPhone dashboard UI to figure out what the name of the executable is and what to call the icon on the touchscreen. The Info.plist is what turns your HelloWorld.app from just a folder with some stuff in it into an actual iPhone application bundle. Note that Info.plist has an uppercase *I* in its name: The iPhone dashboard is picky and expects the files in the bundle to be named precisely.

The bundle also contains two graphics files: Default.png and icon.png. The iPhone dashboard uses icon.png as the icon for your program on the dashboard. Default.png is used to display a dummy graphic on the screen while the user waits for your executable to load and draw its UI.

Note: The icon.png file name must be all lowercase or it won't be displayed. Likewise, Default.png must have an uppercase *D* or it won't be displayed at application start-up.

Now, move on to the real meat of the application: the Objective-C code that makes it all happen.

The Objective-C code

Begin with HelloWorld.m, which Listing 5 shows.

Listing 5. HelloWorld.m

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

#import <Foundation/Foundation.h>
#import <UIKit/UIKit.h>
#import "HelloWorldApp.h"

int main(int argc, char *argv[]) {
    // This autorelease pool is managed by the UI's event dispatcher. It autoreleases
    // any allocated objects that fall out of scope. The iPhone's implementation of
    // Objective-C 2.0 does not have garbage collection yet, but autorelease pools and
    // proper release of allocated objects is still a good practice.
    NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];

    // This is where UI execution gets started. This instantiates the UIApplication
    // subclass and UIApplicationDelegate subclass specified as string parameters.
    // The name of an UIApplication subclass can be passed as both UIApplication and
    // UIApplicationDelegate.
    UIApplicationMain(argc, argv, @"HelloWorldApp", @"HelloWorldApp");

    // Force release of the autorelease pool and all objects still allocated.
    [pool release];
    return 0;
}
```

It actually doesn't look terribly different from C. Foundation and UIKit are the two Objective-C frameworks you will use the most in the Cocoa Touch platform. Foundation is where strings, lists, and base object-oriented hierarchies are defined in Cocoa and Cocoa Touch. UIKit defines quite a lot of the UI elements in an iPhone application.

Objective-C being a superset of C, application start-up is defined pretty much the same way as in C: It all begins at the `main` function. The first line of actual code inside the `main` function is the first bit of Objective-C in this part of the code. There, you're allocating and initializing an instance of an Objective-C class, `NSAutoreleasePool`, used in managing memory. In Mac OS X and Objective-C V2, it's rather like garbage collection, but it is not fully implemented in the iPhone Objective-C environment in this version of the iPhone OS.

The second line of actual code is where the UI framework begins executing.`UIApplicationMain` instantiates the classes named in the last two string parameters and calls specific methods in these two class instances that get the application rolling. The last bit of code gets called when the UI quits. It releases the auto-release pool and returns to the calling environment.

Now, look at HelloWorldApp.m. This is where `UIApplicationMain` has moved the action. Load the following file in Eclipse.

Listing 6. HelloWorldApp.m

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

#import "HelloWorldApp.h"

@implementation HelloWorldApp

- (void)applicationDidFinishLaunching:(UIApplication *)application {
    // Create a "full-screen" window to hold the UI.
    window = [[UIWindow alloc] initWithContentRect:
        [UIHardware fullScreenApplicationContentRect] ];

    // Create a view to hold the window contents.
    contentView = [[UIView alloc] initWithFrame: CGRectMake(0.0f, 0.0f, 320.0f, 480.0f)];

    // Create a navigation bar for the top of the view, with two buttons.
    // The buttons do not do anything in this example.
    nav = [[UINavigationBar alloc] initWithFrame: CGRectMake(0.0f, 0.0f, 320.0f, 48.0f)];
    [nav pushNavigationItem:[[UINavigationItem alloc] initWithTitle:@"Hello World"]];
    [nav showButtonsWithLeftTitle: @"Left" rightTitle: @"Right"];
    [nav setBarStyle: 0];
    [contentView addSubview: nav];
    // Create a text view, this is a basic text editor, with incorporated keyboard.
    text = [[UITextView alloc] initWithFrame: CGRectMake(0.0f, 48.0f, 320.0f, 480.0f)];
    [text setText: [[NSString alloc]
        initWithString: @"Hello World\nCocoa Touch Application"]];
    [contentView addSubview: text];

    // UIWindow con only hold one view, the contentView. The contentView can hold many
    // navigation controllers, which control the different views in your application.
    window.contentView = contentView;

    // These three instructions effectively show the window.
    [window orderFront: self];
    [window makeKey: self];
    [window _setHidden: NO];
}

- (void)dealloc {
    // Release the UI elements as they were allocated
    [text release];
    [nav release];
    [contentView release];
    [window release];

    // And don't forget to call the parent class dealloc
    [super dealloc];
}

@end
```

When `UIApplicationMain` instantiates the `HelloWorldApp` class, application execution resumes with the method called `applicationDidFinishLaunching`. This is where you create the initial UI elements that get the application started from the user's point of view. First, the function allocates a window instance and initializes it with the screen size by calling the `UIHardware` class method `fullScreenApplicationContentRect`.

Next, it allocates and initializes a `UIView` instance, which holds all the other screen elements. There is a navigation bar with two buttons called **Left** and **Right** that in this example are just for show. There is also a big `UITextView` instance with the text "Hello, World" and "Cocoa Touch Application" on two separate lines. Finally, set this view as the main view for your window, and set the window to receive input and be visible.

If you run `make` and everything goes on without a hitch, `make` will have created a HelloWorld.app folder and copied into it your Info.plist file, the icon.png, and Default.png images, and your HelloWorld executable file. Copy that folder to your device. As before, replace your iPhone's IP address for *IPHONE_IP* in the commands below.

```
scp -r HelloWorld.app root@IPHONE_IP:/Applications/
ssh root@IPHONE_IP
```

When you log in to the device, sign the executable to make it runnable.

```
ldid -S /Applications/HelloWorld.app/HelloWorld
```

Make the application accessible

To make your application accessible from the iPhone dashboard, copy the whole application bundle to a folder called Applications on the root of the iPhone file system. But for it to show on the dashboard without having to reboot the device, next run the Respring application you installed at the beginning of the tutorial, and "respring" the dashboard.

After you've resprung the dashboard, your HelloWorld application shows among the other icons. Touch it to launch it and behold: a navigation bar at the top with two buttons named **Left** and **Right**, and a large text view with the phrases, "Hello World" and "Cocoa Touch Application," on two separate lines. And the big surprise: Touch the text area and a keyboard pops up from the bottom of the screen to let you type. The keyboard comes "for free" when you create a text area. You didn't have to write code to enable it.

Section 8. Summary

Using Eclipse to program applications for the iPhone is not without issues. Objective-C is not supported by the refactoring tools, it is not easy to set up remote debugging at the moment (even though GDB supports Objective-C debugging), and third-party plug-ins like Color Editor have to be installed to obtain syntax highlighting of Objective-C code. I could have spent more time setting up Eclipse CDT to allow compilation within Eclipse rather than using `make` from the command line. But then, this would have turned into an Eclipse CDT tutorial instead of a Cocoa Touch tutorial.

With time, I hope to gather a group of developers to help me extend Objective-C support for Eclipse CDT. If you want to learn more about Cocoa Touch development without the iPhone SDK, using Eclipse on Windows or Linux, you can reach me through my e-mail, and I can direct you to more information. Likewise, contact me if you get stuck on any point of this tutorial, such as problems getting the toolchain to build or getting the examples to run.

I hope this tutorial gets you started. Programming the iPhone can be lots of fun.

Resources

Learn

- Stop by the iPhone DevCenter to learn about the Apple iPhone SDK.
- The official iPhone Dev Team information portal, containing "accurate, useful and concise information portal about Dev Team progress."
- For more information about jailbreaking your iPhone using WinPwn, read "How to Unlock/Jailbreak Your 2.0 EDGE iPhone with WinPwn on Windows" from iClarified.
- For information about jailbreaking your iPhone 3G using WinPwn, read "How to Jailbreak Your 2.0 3G iPhone with WinPwn on Windows" from iClarified.
- For information about jailbreaking your iPhone using Pwnage, read "How to Unlock/Jailbreak Your 2.0 EDGE iPhone with Pwnage Tool on Mac OS X" from iClarified.
- Read the developerworks article "Develop iPhone Web applications with Eclipse" to learn how to create iPhone Web sites using Eclipse, Aptana's iPhone Development plug-in, and the iUi framework.
- Read the developerworks tutorial "Debug iPhone Web applications with Eclipse" to learn how to debug Asynchronous JavaScript + XML (Ajax) Web applications using Eclipse, Aptana's iPhone Development plug-in, Aptana's Firefox JavaScript debugger, and Firebug.
- For information on jailbreaking your iPhone 3G using Pwnage, read "How to Jailbreak Your 2.0 3G iPhone with Pwnage Tool on Mac OS X" from iClarified.
- Check out CNet's video explanation of jailbreaking the iPhone.
- For information about jailbreaking your iPhone using QuickPwn, read "How to Jailbreak Your 2.0 iPhone with QuickPwn on Windows" from iClarified.
- See the official information page for Xpwn.
- Check out the "Recommended Eclipse reading list."
- Browse all the Eclipse content on developerWorks.
- New to Eclipse? Read the developerWorks article "Get started with Eclipse Platform" to learn its origin and architecture, and how to extend Eclipse with plug-ins.
- Expand your Eclipse skills by checking out IBM developerWorks' Eclipse project resources.
- To listen to interesting interviews and discussions for software developers, check out developerWorks podcasts.

- Stay current with developerWorks' Technical events and webcasts.
- Watch and learn about IBM and open source technologies and product functions with the no-cost developerWorks On demand demos.
- Check out upcoming conferences, trade shows, webcasts, and other Events around the world that are of interest to IBM open source developers.
- Visit the developerWorks Open source zone for extensive how-to information, tools, and project updates to help you develop with open source technologies and use them with IBM's products.

Get products and technologies

- Download the author's source code to iPhone development tools from his Web site.
- Eclipse Ganymede is at the Eclipse Foundation.
- EasyEclipse is a bare-bones Eclipse distribution for experienced C and C++ developers who are new to Eclipse. EasyEclipse for C and C++ includes Eclipse, the C Development Toolkit, Subclipse, and editor plug-ins for syntax highlighting of several languages, including Objective-C.
- Find out more about Eclipse CDT, and the CDT URLs for each version of Eclipse.
- Learn more about Color Editor and download the syntax coloring tool for Eclipse for more than 140 languages.
- Find out more about Cygwin and download the precompiled UNIX tools for Windows.
- Check out Transmac, commercial software for opening DMG images in Windows.
- Check out the latest Eclipse technology downloads at IBM alphaWorks.
- Download Eclipse Platform and other projects from the Eclipse Foundation.
- Download IBM product evaluation versions, and get your hands on application development tools and middleware products from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.
- Innovate your next open source development project with IBM trial software, available for download or on DVD.

Discuss

- The Eclipse Platform newsgroups should be your first stop to discuss questions regarding Eclipse. (Selecting this will launch your default Usenet news reader application and open eclipse.platform.)

- The Eclipse newsgroups has many resources for people interested in using and extending Eclipse.
- Participate in developerWorks blogs and get involved in the developerWorks community.

About the author

PJ Cabrera
PJ Cabrera is a freelance software developer specializing in Ruby on Rails e-commerce and content-management systems. His interests include Ruby on Rails and open source scripting languages, and frameworks, Agile development practices, mesh networks, compute clouds, XML parsing, and processing technologies, microformats for more semantic Web content, and research into innovative uses of Bayesian filtering and symbolic processing for improved information retrieval, question answering, text categorization, and extraction. Check out his blog.