Appendix A

Installing Clang Static Analyzer on Mac OS X Leopard

If you're using the iPhone SDK on Mac OS X Leopard, you can install the Clang Static Analyzer as a command-line utility and use it to check for coding errors in your Xcode projects. Of course, if you are developing on Mac OS X 10.6 ("Snow Leopard"), the Clang Static Analyzer has been integrated into Xcode and you can analyze your code from within the IDE.

A.1 Downloading and Installing

Begin by visiting http://clang.llvm.org/StaticAnalysis.html, the Static Analyzer's page. Find the download link on this page—as of this writing, it's checker-0.214.tar.bz2—and download the compressed archive to your Mac. The tarred and bzip2ed file can be easily uncompressed by simply double-clicking it in the Finder, which will create a folder named checker-XXX, where XXX is the build number. The contents of this file are the executables and the files that they need, so you could execute the application right away by typing in a full path to the executable. More likely, you probably want to move the folder some place permanent, and to get it in your command-line PATH, so you can execute it without typing in a full path.

What I usually do is to copy the analyzer to the /usr/local directory, and then make a symbolic link in /usr/local/bin to the scan-build executable. With this approach, I can keep a couple builds of Clang hanging around in /usr/local, so if a new build doesn't work, I just re-point the sym-

link. To do this, go to the Terminal and change your current directory to Downloads with a simple cd ~/Downloads. Assuming you've already decompressed the checker-XXX folder, then you copy it to /usr/local by means of the sudo command, which performs your copy command as an Administrator (which you need in order to copy into that system directory):

```
Yuna:~ cadamson$ sudo cp -R ~/Downloads/checker-0.214 /usr/local
```

Next, you create the symbolic link in the /usr/local/bin directory that points to the scan-build executable in the directory you just copied. This is a convenience so you can kick off the analyzer by just typing scan-build rather than having to type the entire path /usr/local/checker-0.214/scan-build. Of course, you might have to create /usr/local/bin if it doesn't already exist (check with ls /usr/local). Here are the necessary commands:[1]

```
Yuna:Downloads cadamson$ sudo mkdir /usr/local/bin  # only if needed
Yuna:Downloads cadamson$ sudo ln -s /usr/local/checker-0.214/scan-build\
/usr/local/bin/scan-build
```

By the way, if you install a newer version of scan-build and repeat these steps, you need to manually remove the old symlink with sudo rm /usr/local/bin/scan-build before creating the new symlink.

There's a second executable, scan-view, that offers an optimized view of the build results, by running as a tiny web server on port 8181. Let's get that in your path too:

```
Yuna:Downloads cadamson$ sudo ln -s /usr/local/checker-0.214/scan-view\
/usr/local/bin/scan-view
```

A.2 Preparing your project for the Analyzer

So, now you've installed the static analyzer and should have it in your PATH (you can test this with which scan-build). Test it out by printing its usage message:

```
Yuna:Downloads cadamson$ scan-build --help
```

This will produce a long message that shows the build date and number, along with a summary of the command-line options.

1. Note that we have used the line-break character (\) to present the ln command on two lines to fit the book's formatting. You can omit the backslash and the line-break and just type this command on a single line.

Assuming you've been working through the book's "Performance Tuning" chapter, let's run the Clang Static Analyzer against that chapter's PathologicalPrimeCounter code and see what turns up. Use the cd command to change your current directory to the folder containing your Xcode project.[2]

You need to take care of one more piece of business before running the analyzer. Be sure that your project builds without errors—on the command-line, CSA's scan-build actually builds your code, either via make or xcodebuild, so you have to be sure that your project builds correctly with its default options. If it doesn't, you may see a bewildering error message saying that scan-build can't find a needed version of gcc.

In practice, we've seen this when a project defaults to building for the device instead of the simulator, and the current environment isn't provisioned for code signing. This may be the case if you download someone else's code, the project is set to build for the device, and you haven't joined the paid iPhone developer program and gotten an app signing certificate yet. You can get around this by changing the project's properties to build for the simulator by default, rather than the device.

In your Xcode project, select the project icon from the very top of the "Groups & Files" tree, then choose "Get Info" from the toolbar or the "File" menu. As shown in the (as yet) unwritten *fig.clang-set-base-sdk*, set the "Base SDK" to "Simulator", rather than "Device". If you want you can double-check your work on the command-line by using xcodebuild to build the project with its default options; don't try running the Analyzer until this gives you a BUILD SUCCEEDED result.

A.3 Running the Analyzer

Now we're ready to actually run scan-build on the project. To do this, you use the command scan-build, followed by whatever command builds your project: either xcodebuild or make. In our case, it's the former:

```
Yuna:PathologicalPrimeCounter cadamson$ scan-build xcodebuild
```

Once you do this, you'll see a bunch of logging messages from the build, concluding with the output from scan-build:

2. Tip: Rather than typing this path into Terminal, you can type cd, a space, and then drag the folder icon from a Finder window's title bar into the Terminal window to have the path to that folder inserted onto the command line.

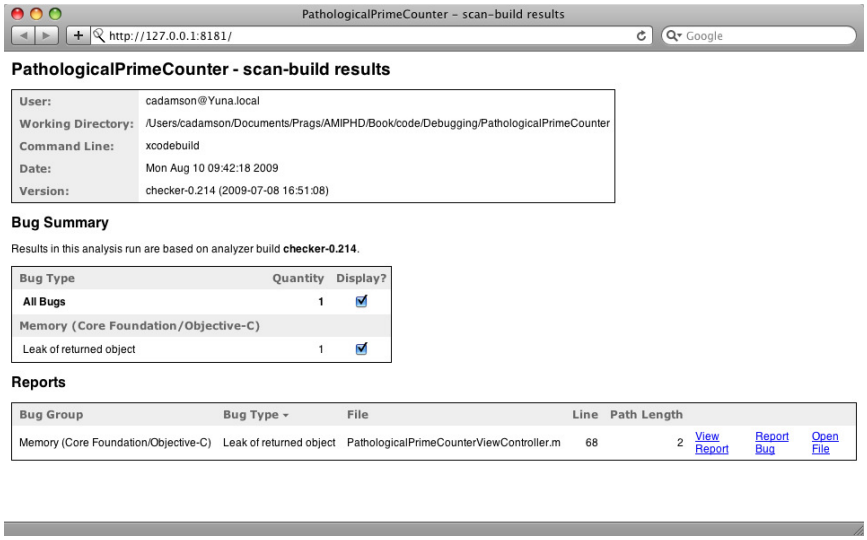

Figure A.1: SCAN-VIEW OVERVIEW OF STATIC ANALYSIS REPORT

```
** BUILD SUCCEEDED **
scan-build: 1 bugs found.
scan-build: Run 'scan-view /var/folders/ba/baOL2wJxE8aPV3tF1AeZsU+++TI/
        -Tmp-/scan-build-2009-08-10-1' to examine bug reports.
```

Ah, a bug. Let's take a look. If you're using Terminal, copy and paste the scan-view command to your command-line and execute it to bring up the bug report in your preferred browser, as seen in Figure A.1.[3] This launches your default browser, and shows an overview of the static analyzer's report, including how the analyzer was run, and a count of the various types of bugs.

Down in the "Reports" section, you'll see a list of each discovered bug, described by group, type, and the source file and line number where you'll find the bug. After this, you'll find three links:

- *View Report* to investigate code paths that lead to the bug.
- *Report Bug* to file a bug with the Clang project, if you believe the bug report to be erroneous.

3. Instead of manually typing scan-view, you can go directly to the results at the end of the scan by using scan-build's -V argument.

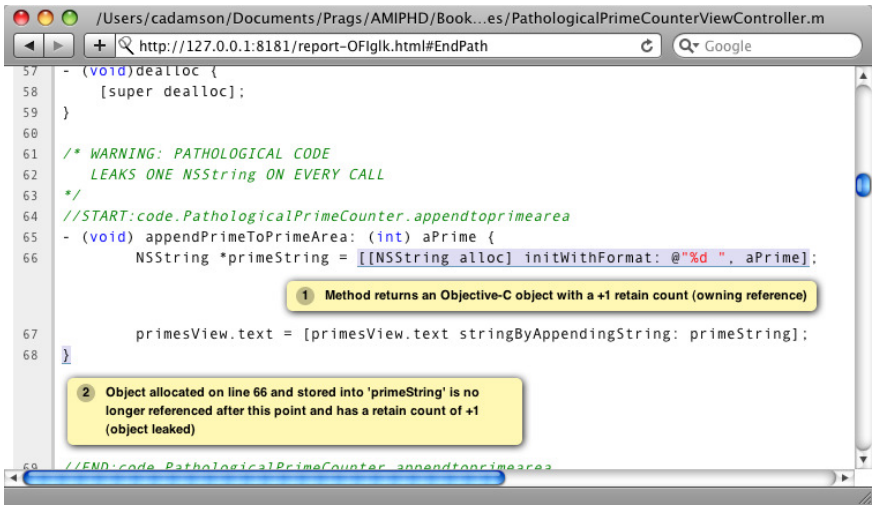

Figure A.2: VIEWING THE CLANG STATIC ANALYZER REPORT OF AN OBJECTIVE-C MEMORY LEAK

- *Open File* to open the source file in Xcode.

Click "View Report" to see an HTML-formatted presentation of the discovered bug. Figure A.2

The report shows each step of a path through the various branches of your code that leads to the bug. In this case, there are just two annotations: point 1 shows where we have **alloc**'ed an NSString, and point 2 indicates where it is no longer referenced (indeed, it has gone out of scope) and has not been **release**d. With a retain count of 1, the object leaks. The fix here is to either manually **release** the string when it is no longer needed, or to auto-release it (by appending the **autorelease** method to the line that creates the string, or by using the autoreleasing class method stringWithFormat:).

A.4 Final Thoughts

Now that you have the Clang Static Analyzer installed, you can make it a regular part of your development process. Whether you have a runaway memory leak, or you're just being careful to pick off the little leaks that could add up over time, scan-build will save you a lot of time and

precious device memory. We used it to check most of the book's example code,[4] and it helped root out a number of mistakes that we're glad not to have shipped with the final version.

Despite the fact that it is not even at a full 1.0 release, Apple has incorporated the Clang Static Analyzer into Snow Leopard's Xcode's tools, and that's a strong statement about the promise and value of this project. With the ability to find important and subtle bugs with little effort on your part—arguably a lot less than tracking down a leak with Instruments—it will likely be a major part of your iPhone development toolkit going forward, whether used as part of Xcode or run from the command-line as a stand-alone utility, as we've presented in this appendix.

4. We had problems with a few device-only projects and went over those manually.