

Section I: Getting Started with Concurrency

SECTION 0: 3 CHAPTERS

Concurrency
SECTION 1: 2 CHAPTERS

Section II: Grand Central
Dispatch
SECTION 2: 3 CHAPTERS

Section III: Operations
SECTION 3: 5 CHAPTERS

Section IV: Real-Life Concurrency

Concurrency
SECTION 4: 3 CHAPTERS

11. Core Data

11.1 NSManagedObjectContext is not thread safe
11.2 Importing data
11.3 NSAsynchronousFetchRequest
11.4 Sharing an NSManagedObject

11.4 Sharing an NSManagedObje
11.5 Where to go from here?
12. Thread Sanitizer
12.1 Why the sanitizer?
12.2 Getting started

12.3 Enabling sanitization12.4 It's not code analysis12.5 Xcode keeps getting smarter

12.6 Where to go from here?

13. Conclusion

Home > iOS & Swift Books > Concurrency by Tutorials

12 Thread Sanitizer Written by Scott Grosch

By the time you reach this chapter, you should be a whiz at concurrent programming.

However, just because your app *seems* to run correctly doesn't mean you took care of various concurrency and threading related edge cases. In this chapter you'll learn how to utilize Xcode's built-in **Thread Sanitizer** to discover races and before you deploy to the App Store.

Why the sanitizer? As discussed in Chapter 5, "Concurrency Problems," you know how important it is to

keep all accesses to a variable on the same thread to avoid data races. Depending on how your app is structured, or the architecture of a third-party library that you're using, it can be hard to tell if you're crossing a thread boundary.

The Thread Sanitizer, commonly referred to as TSan, is a tool Apple provides as part

of the LLVM compiler. TSan allows you to identify when multiple threads attempt to access the same memory without providing proper access synchronization.

Note: TSan is only supported when running on the simulator.

the **Main Thread Checker** — the only one you'll need to actively use at this point is the **Thread Sanitizer**. The other sanitizers are either meant for other languages, or are on by default.

While there are a few other sanitizers available — such as the **Address Sanitizer** or

Getting startedOpen up the Xcode project provided in this chapter's **starter** folder. It's a pretty

simple-looking app that writes to a variable in two separate locations. Build and run the app; you won't see any issues. If you look at the code of

MainViewController.swift, though, you can see you're clearly utilizing the counter

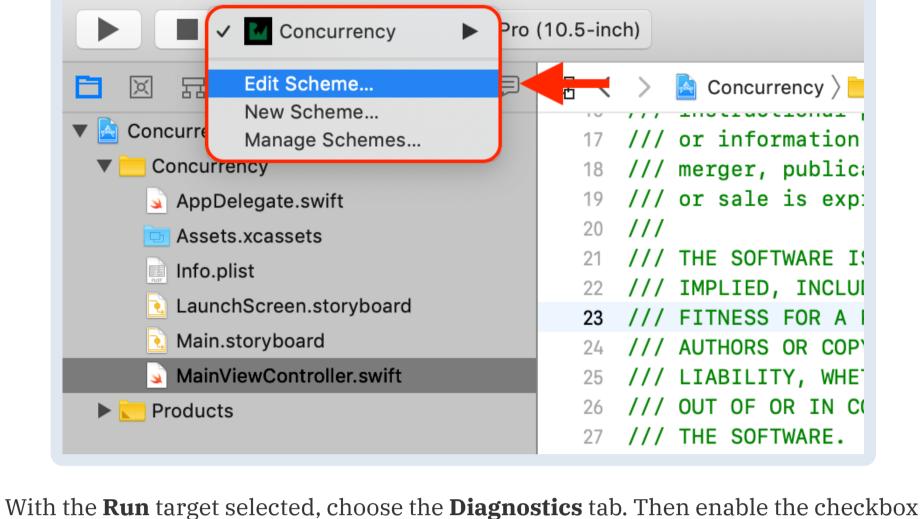
property in different dispatch queues.

Is there any issue with your code? Probably! But you can easily catch and confirm

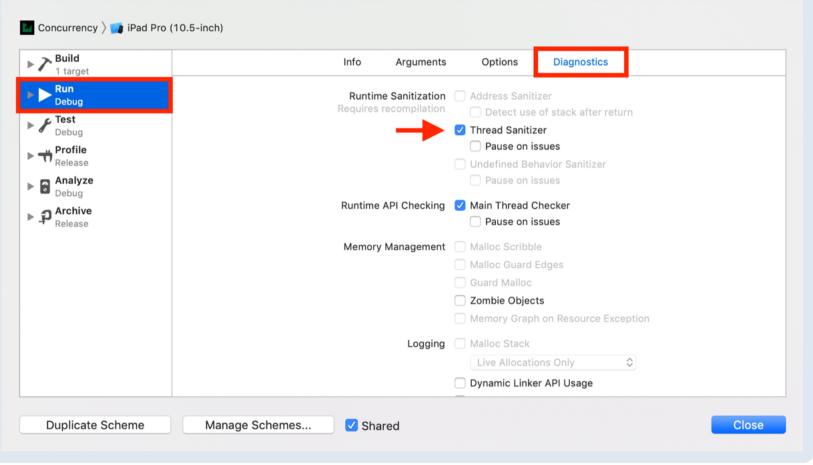
this and other threading issues with the Thread Sanitizer.

Enabling sanitizationThe first step to checking your code with the Thread Sanitizer is enabling it. To do

so, first click on the **Concurrency** scheme in the top-left of Xcode, and select the **Edit Scheme...** option.



for **Thread Sanitizer**.



and run your app.

Almost immediately, you'll notice that Xcode emits a threading issue, and you're

Once you've enabled the thread sanitizer, close the dialog window and then build

Buildtime Runtime (1)

33
34 override func viewDidLoad() {

super.viewDidLoad()

shown that you have an access race in your closure.

▼ Concurrency - 58202 1 issue

```
▼ ■ Threading Issues
                                                   36
             ▼ ■ Swift access race in closure #1 () ->
                                                   37
                                                            let queue = DispatchQueue(label: "q")
                                                            queue.async {
                 Concurrency.MainViewController.vi...
                                                             for _ in 1 ... 10000 {
                 Location is a 856-byte heap
                                                                Thread.sleep(forTimeInterval: 0.1)
                   object at 0x7b5c00001880
                                                                self.counter += 1  Swift access race in closure #1
               ▼ (III) Mutating access by thread 2
                                                   42
                                                           }
                                                   43
                   0 closure #1 in MainViewCont...
                   4 _dispatch_client_callout
                                                           DispatchQueue.main.async {
               ▼ (III) Mutating access by thread 1
                                                             for _ in 1 ... 10000 {
                   0 closure #2 in MainViewCont...
                                                   47
                                                                self.counter += 1
                   5 start
                                                   48
                                                   49
                                                           }
               ▶ I Heap block allocated by thread 1
Switch to the Issue navigator (#-5) and select the Runtime tab. If you expand the
```

the lines to be taken to that point in your source code.

It's not code analysis

stack trace show you'll see exactly where Xcode found the issues. You can click on

It's important to keep in mind that the Thread Sanitizer is **runtime** analysis. What that means is that, if an issue doesn't occur during execution, it won't be flagged. If

you look at the code in MainViewController.swift, you'll see this, around line 40:

Thread.sleep(forTimeInterval: 0.1)

COPY

Comment out that line of code and run the app again. This time, the app runs

without any issues. Why is that? Even though you accessed the counter variable

across multiple threads, those accesses didn't end up happening at the same time. By including the <code>sleep(forTimeinterval:)</code> line you forced iOS to switch threads, triggering the issue.

The important takeaway is that you should try to run using the Thread Sanitizer frequently — and not just once. Enabling the sanitizer has a pretty steep

performance impact of anywhere between 2x to 20x CPU slowdown, and yet, you

should run it often or otherwise integrate it into your workflow, so you don't miss

Apple continues to improve runtime analysis of your apps even without the Thread Sanitizer. You know how important it is, by now, to always execute UI tasks on the

Xcode keeps getting smarter

out on all of these sneaky threading issue!

main thread. If you forget, your app will abort at runtime in debugging. In years past, you'd have had to use the Thread Sanitizer to identify such a mistake.

Xcode does such a good job of flagging errors at runtime that it was actually quite difficult to come up with an example for this chapter!

Where to go from here?

You've seen how important the Thread Sanitizer is to ensuring stable apps for your customers. I suggest you enable the sanitizer at the start of app development and leave it on until you find that it is impacting the performance of testing. Once it has

reached an impactful point, then disable it but continue to run at regular intervals.

Apple's official documentation is always available to you at https://developer.apple.com/documentation/code_diagnostics/thread_sanitizer. You might also look into the various continuous integration techniques available. Many

of them will allow the Thread Sanitizer to be utilized during automated testing.

Mark Complete



13. Conclusion →

11. Core Data

Have a technical question? Want to report a bug? You can ask questions and report bugs to the book authors in our official book forum here.

Have feedback to share about the online reading experience? If you have feedback about

design team with the form below:

the UI, UX, highlighting, or other features of our online readers, you can send them to the

Feedback about the UI, UX, or other features of the online reader? Leave them here!

Send Feedback

© 2021 Razeware LLC

 \leftarrow