DispatchGroups to group API calls in Swift

Need to make multiple network calls for your ViewController? We can use DispatchGroup, and this article explains why.

[](https://medium.com/@stevenpcurtis.sc?source=post_page-----7906b2203854----------------------)

[Steven Curtis](https://medium.com/@stevenpcurtis.sc?source=post_page-----7906b2203854----------------------)

[Oct 18](https://medium.com/@stevenpcurtis.sc/dispatchgroups-to-group-api-calls-in-swift-7906b2203854?source=post_page-----7906b2203854----------------------) · 3 min read





I guess it’s a group?

Basically you can add work to a group, which then can be executed asynchronously than you can run a closure once they are finished.

Now of course it is possible to wait synchronously for a series of API calls to complete, then synchronously run whatever you want. But you’d rather do things the right way, right?

Difficulty: **Easy** | Normal | Challenging

**Prerequisites:**

* Basic understanding of Grand Central Dispatch (async functions)

**Terminology**

Asynchronous: Work that can be run out of order, and usually have a callback when completed

DispatchGroup: A group of tasks to be monitored as a single unit

Synchronous: the thread that initiated that operation will wait for the task to finish before continuing. This means that tasks assigned synchronously run in order. If performed on the main thread, this means that the UI can be blocked which means that users of your software might feel that is has crashed — this is very bad! Of course this actually applies to whichever thread you run the process on, so usually activities like network calls are best made asynchronously

**Creating a DispatchGroup**

DispatchGroups can easily be created with whichever name to mark them that will be useful to refer to later.

For this tutorial, we are going to use the name “waitingGroup”

*let waitingGroup = DispatchGroup()*

**A gotcha**

You need to make sure that each time you enter your DispatchGroup, that you leave. The two operations should be paired so you make sure that for each one you enter, that you also leave.

**Enter the queue**

You mark that each block has entered a group by calling enter, on the DispatchGroup that you have created. As above, ours is called waitingGroup

*waitingGroup.enter()*

This is run before our target async closure

**Perform the task and exit the queue**

Here we use a wait function to simulate an async network or API call

*wait(delay: 1) {*

*waitingGroup.leave()*

*}*

where the wait function would be

*func wait(delay: UInt32, completion: () -> Void) {*

*sleep(delay)*

*completion()*

*}*

**After the task(s) are done**

After (in this case) the same number of *waitingGroup.leave* have been run we now can run a closure marked with .notify:

*waitingGroup.notify(queue: .main) {*

*print(“done”)*

*}*

**Conclusion**

DispatchGroup is a great way to solve a common problem — you need multiple APU calls to display your view controller. Rather than using booleans to make this work, you should really use DispatchGroups. Now, you don’t have an excuse!

**The repo**

I’m not going to let you attempt this without the code available to you; you’re not on your own. Here you go: