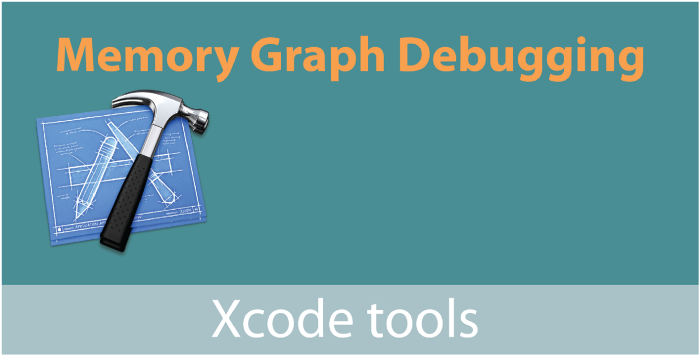
Memory leaks using the Xcode memory graph debugger

[](https://medium.com/@stevenpcurtis.sc?source=post_page-----60de23bab3aa----------------------)

[Steven Curtis](https://medium.com/@stevenpcurtis.sc?source=post_page-----60de23bab3aa----------------------)

[May 22](https://medium.com/@stevenpcurtis.sc/memory-leaks-using-the-xcode-memory-graph-debugger-60de23bab3aa?source=post_page-----60de23bab3aa----------------------) · 5 min read





Memory leaks can be thought of as memory that is allocated, but never released (and then no longer used by the App). When there are no references to a block or section of memory, there is no way to release it an it can’t be used again.

Prerequisites:

* Reference and value types
* MVVM architecture

**Terminology**

Instruments: Instruments are a set of application performance analyzes and visualizers integrated into Xcode.

Memory leak: A memory leak is a portion of memory that will never be used, yet is held onto forever. It is both a waste of space and can cause problems.

Memory footprint: the amount of memory that a program uses or references. The more resources that are used, the larger the footprint. If objects are not released the occupied memory will grow, leading to memory warning and crashes.

Automatic Reference Counting (ARC): ARC is a memory management feature (of the Clang compiler) that provides the reference counting for both Objective-C and Swift. The references may be weak or strong, and when a strong reference is declared the counter increases by one. Upon release, the counter decreases by 1 and when the reference count is 0 ARC can deallocate it from the memory heap (as it will be satisfied that the object is not in use). Since weak references do not increase the reference counter objects that are only held with a weak reference(s) are automatically released since the counter will be set to 0.

Retain cycles: This is the state when two objects hold weak references to one another. Since the first object’s reference count cannot be 0 until the second object is released, and the second object’s reference count cannot be 0 until the first objet is released neither object can be released! It is self-evident that it is not possible to create retain cycles with value types, as they are passed by value.

malloc: allocates a memory block of a given size and returns a pointer to the beginning of the block.

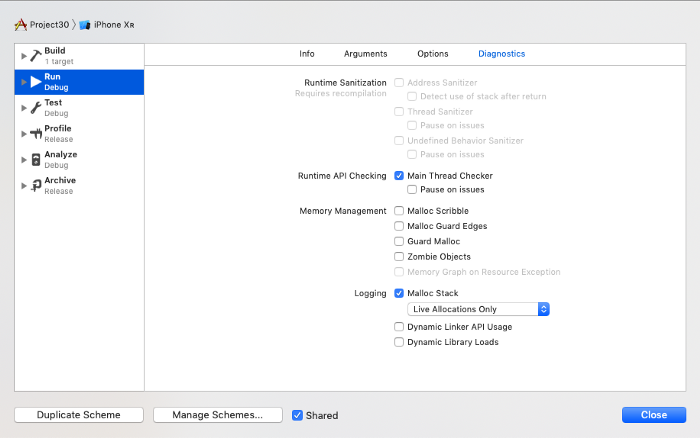
Malloc Stack: Gives the stack trace for memory leaks

Malloc Scribble: Gives Xcode more accurate results for memory leaks, by filling freed memory with a predefined value so it is obvious which memory blocks are not being released.

**Using the memory graph debugging mode within any App**

Edit scheme> Diagnostics > Malloc Stack > Live Allocations Only





There is no need to select All Allocations for most needs, as here we are identifying retain cycles and leaks.

We will only identify a memory leak at a single snapshot in time, so this applies at a time where we set a breakpoint in our code or alternatively by clicking on the icon when your project is running (which will pause it.

It is now possible to enter memory graph debugging mode through the debug bar button that looks like the following:





On the left of the display the App’s heap contents are displayed.

Leaks are displayed with a purple exclamation mark beside the items in the debug navigator.

— Filtering can help identify individual items

— the runtime navigator can also display the number of leaks identified

**Using the memory graph debugging mode using protocol-delegate**

It is common to use the MVVM architecture in iOS Apps, and gives us a good reason to couple a view controller with a delegate (although commonly bindings are not performed this way, think of this as a minimum viable example).

This example App simply has an introduction screen, and allows you to view a detail view controller with an image. It is this detail view controller that has a view model (and also displays the issue with this code)

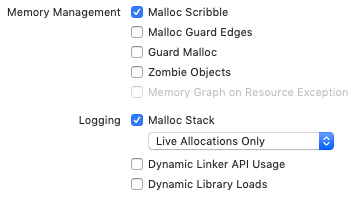
|  |  |
| --- | --- |
|  | class DetailLeakViewController: UIViewController, ViewModelDelegate { |
|  |  |
|  | var viewmodel : ViewModel? |
|  |  |
|  | func retrieveData(\_ data: [String]){ |
|  | print (data) |
|  | } |
|  |  |
|  | override func viewDidLoad() { |
|  | super.viewDidLoad() |
|  | viewmodel = ViewModel() |
|  | viewmodel?.delegate = self |
|  | } |
|  | } |
|  |  |
|  | protocol ViewModelDelegate { |
|  | func retrieveData(\_ data: [String]) |
|  | } |
|  |  |
|  | class ViewModel { |
|  | let data = ["1","2","3","4","5","6"] |
|  | var delegate : ViewModelDelegate? |
|  |  |
|  | func fetchData() { |
|  | delegate?.retrieveData(data) |
|  | } |
|  | } |

[**view raw**](https://gist.github.com/stevencurtis/7e5cc6b3a754558d18f371ec794caa66/raw/7e1cf039f0888aadcb277ad9fa20eb2785a94555/DetailLeakViewController)

[**DetailLeakViewController**](https://gist.github.com/stevencurtis/7e5cc6b3a754558d18f371ec794caa66#file-detailleakviewcontroller) hosted with ❤ by [**GitHub**](https://github.com/)

To view the issue with the memory debugger, as before we set the scheme diagnostics to allow Malloc Scribble and Malloc Stack:





We can then run the App and click on the Memory Leak view controller and the No memory leak buttons (segue to their view controllers of course) several times.

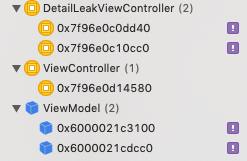
Then enter the memory leak debugger as before:





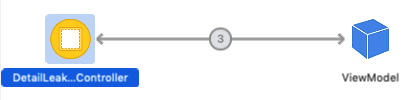
we can then see there is a memory leak!





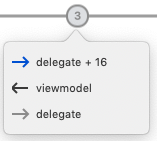
Clicking on the first purple exclamation mark reveals the problem





In fact you can click on the number 3 (yours may display a different number, depending on the number of times you traversed to the view controller).





We can see the issue. Clicking on either the viewmodel or delegate enables the jump to definition button just below it.





which sits just below the diagram:





Which gives the opportunity to explore both references. Of course this example has created a reference to the view model that is strong and the viewmodel has a reference to the delegate which is also strong.

Traditionally delegates are weak to prevent memory cycles in this way, and this is how the code in structured in DetailNoLeakViewController.swift within the GitHub repo for this mini-project:

**[stevencurtis/ProtocolDelegateMemoryLeak](https://github.com/stevencurtis/ProtocolDelegateMemoryLeak?source=post_page-----60de23bab3aa----------------------)**

[Contribute to stevencurtis/ProtocolDelegateMemoryLeak development by creating an account on GitHub.](https://github.com/stevencurtis/ProtocolDelegateMemoryLeak?source=post_page-----60de23bab3aa----------------------)

[github.com](https://github.com/stevencurtis/ProtocolDelegateMemoryLeak?source=post_page-----60de23bab3aa----------------------)

**Side effects of memory leaks**

Using KVO, objects can listen for notifications and perform actions upon receiving the notification. Due to the potential for increasing duplicate objects through memory leaks, a (even wanted) action can happen multiple times. There may be conflicts between these actions (for example actions taking place on a database), corrupting the state of the App.

**Further information about memory leaks**

You can look into the following WWDC video that rounds up debugging including memory debugging in an accessible way:

**[Visual Debugging with Xcode - WWDC 2016 - Videos - Apple Developer](https://developer.apple.com/videos/play/wwdc2016/410/?source=post_page-----60de23bab3aa----------------------)**

[Discover Xcode's enhancements for debugging autolayout issues at runtime. Learn how issues inside complex…](https://developer.apple.com/videos/play/wwdc2016/410/?source=post_page-----60de23bab3aa----------------------)

[developer.apple.com](https://developer.apple.com/videos/play/wwdc2016/410/?source=post_page-----60de23bab3aa----------------------)

**Want to get in contact? Use Twitter:**

**[Steven Curtis (@stevenpcurtis) | Twitter](https://twitter.com/stevenpcurtis?source=post_page-----60de23bab3aa----------------------)**

[The latest Tweets from Steven Curtis (@stevenpcurtis). Studying for a Masters in Computing while developing iOS Apps…](https://twitter.com/stevenpcurtis?source=post_page-----60de23bab3aa----------------------)

[twitter.com](https://twitter.com/stevenpcurtis?source=post_page-----60de23bab3aa----------------------)