
Debugging



Debugging

“Xcode’s debugger (called LLDB) is the fundamental tool that will help you find these bugs and fix them.

Xcode Console

Buggy project

run

See output:

```
2016-08-24 12:52:38.463 Buggy[1961:47078] -[Buggy.ViewController buttonTapped:]:  
unrecognized selector sent to instance 0x7ff6db708870”
```



Debugging

```
@IBAction func buttonTapped(_ sender: UIButton) {  
    print("Called buttonTapped(_:)"  
}  
}
```

Caveman Debugging

caveman debugging

strategically placing `print()` calls in your code to verify that functions and methods are being called (and called in the proper sequence) and to log variable values to the console to keep an eye on important data.

```
“@IBAction func buttonTapped(_ sender: UISwitch) {  
    print("Called buttonTapped(_:)"  
    // Log the control state:  
    print("Is control on? \(sender.isOn)"  
}
```

Breakpoints

ViewController.swift

```
13  
14 ● @IBAction func buttonTapped(_ sender: UIButton) {  
15     print("Method: \(#function) in file: \(#file) line: \(#line) called.")  
16  
17     badMethod()  
18 }  
19
```

```
13 ● @IBAction func buttonTapped(_ sender: UIButton) {  
15     print("Method: \(#function) in file: \(#file) line: \(#line) called.")  
16  
17     badMethod()  
18 }  
19
```

```
13 ● @IBAction func buttonTapped(_ sender: UIButton) {  
14     print("Method: \(#function) in file: \(#file) line: \(#line) called.")  
15  
16     badMethod()  
17 }  
18  
19
```

- Edit Breakpoint...
- Disable Breakpoint
- Delete Breakpoint
- Reveal in Breakpoint Navigator

Breakpoints

The image shows the Xcode IDE interface with a Swift file named `ViewController.swift` open. A breakpoint is set at line 15, which contains a `print` statement. The call stack on the left shows the current execution point at `ViewController.buttonTapped(L:)`. The console at the bottom displays the output of the breakpoint, including the sender and self objects. Labels with arrows point to the following components:

- Debug navigator**: Points to the left sidebar showing the project structure and call stack.
- Variables view**: Points to the area below the call stack showing the current state of variables.
- Debug area**: Points to the main editor area where the source code is displayed.
- Console**: Points to the bottom right area showing the output of the breakpoint.
- Toggle variables view**: Points to the icon in the bottom right corner used to toggle the variables view.
- Toggle console**: Points to the icon in the bottom right corner used to toggle the console.

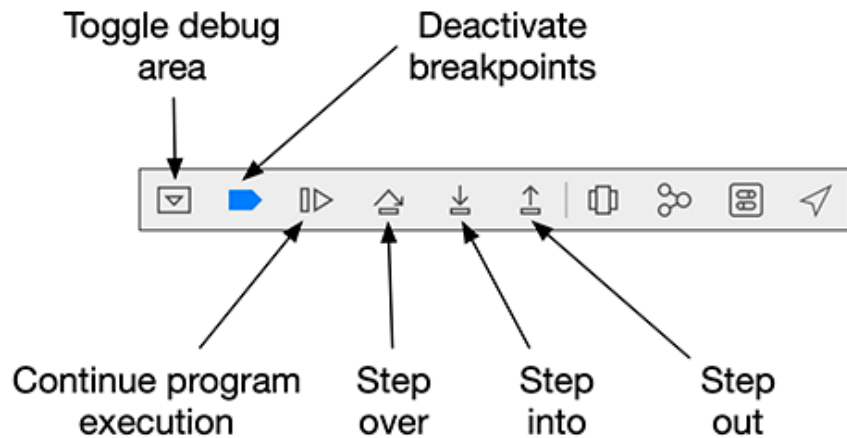
Breakpoints

The screenshot shows the Xcode IDE with a breakpoint set on the `buttonTapped:` method of a `UIButton`. The breakpoint is located at line 15 of the `ViewController.m` file. The call stack on the left shows the sequence of events leading to the breakpoint, including the `UIApplication` sending an action to the `UIButton`. The main editor shows the code for the `buttonTapped:` method, which prints the array of objects. The console at the bottom shows the output of the `print` statement, displaying the array `(11db)`.

```
7  cla
8
9
10
11
12
13
14 Tap me!
15 1 @objc ViewController.buttonTapped:(UIButton *)sender {
16 2 -[UIApplication sendAction:to:from:withEvent:]
17 3 -[UIButton sendAction:to:from:withEvent:]
18 4 -[UIButton _sendActionsForControlEvents:]
19 5 -[UIButton touchesEnded:withEvent:]
20 6 -[UIWindow _sendTouchesForEvent:]
21 7 -[UIWindow sendEvent:]
22 8 -[UIApplication sendEvent:]
23 9 -[UIApplicationAccessibility sendAccessibilityEvent:]
24 10 __dispatchPreprocessedEventQueueProcessEventQueueInternal
25 11 __handleEventQueueInternal
26 12 __CFRunLoop_IS_CALLING_IN_TO_THIS_LOOP
27 13 __CFRunLoopDoSource0
28 14 __CFRunLoopDoSources0
29 15 __CFRunLoopRun
30 16 CFRunLoopRunSpecific
31 17 GSEventRunModal
32 18 UIApplicationMain
33 19 main
34 20 start
35 21 start
36 Thread 2 Queue: com.apple.UIKit.EventFetcher
37 Thread 3
38 Thread 4
39 Thread 5
40 com.apple.UIKit.EventFetcher-thread...
```

Breakpoints

ViewController.swift



Breakpoints

ViewController.swift

```
22
23     for i in 0..<10 {
24         array.insert(i, at: i)
    }
```

ing the array

)

ViewController.swift:24

Condition

Ignore 0 times before stopping

Action Sound 🔔 Tink +-

Log Message +-

Pass number %H

Log message to console @exp@ = expression

Speak message %B = breakpoint name

%H = breakpoint hit count

Options Automatically continue after evaluating actions

Breakpoints

ViewController.swift

```
26
27     // Go one step too far emptying the array
28     for _ in 0...10 {
29         array.removeObject(at: 0)
    }
```

Condition

Ignore 0 times before stopping

Action Sound Tink

Log Message

Pass number %H

Log message to console @exp@ = expression
 Speak message %B = breakpoint name
%H = breakpoint hit count

Options Automatically continue after evaluating actions

Breakpoints

```
“func badMethod() {  
    let array = NSMutableArray()  
  
    for i in 0..<10 {  
        array.insert(i, at: i)  
    }  
}
```

```
// Go one step too far emptying the array (notice the range change):  
for _ in 0...10 {  
    for _ in 0..<10 {  
        array.removeObject(at: 0)  
    }  
}
```