Debugging Tips and Tricks
Xcode 8 edition
Session 417

Kate Stone
Enrico Granata
Sean Callanan
Jim Ingham
LLDB is Apple’s Debugger

... and it’s everywhere you need it

LLDB in the Xcode debug console

- Xcode hosts app console + LLDB prompt
LLDB is Apple’s Debugger
... and it’s everywhere you need it

LLDB in the Xcode debug console
• Xcode hosts app console + LLDB prompt

```
self = (SKTZoomingScrollView *) 0x100351e50
_cmd = (SEL) "validateFactorPopUpButton"
  factorPopUpButtonCell = (NSPopUpButtonCell *) NULL
  index (NotNull)
```
LLDB is Apple’s Debugger
... and it’s everywhere you need it

LLDB in the Xcode debug console

- Xcode hosts app console + LLDB prompt

```bash
self = (SKZoomingScrollView *) 0x100351e50
_cmd = (SEL) "validateFactorPopUpButton"
factorPopUpButtonCell = (NSNotification *) NULL
index (NSInteger)
```
LLDB is Apple’s Debugger
... and it’s everywhere you need it

Scheme option to use standalone terminal
LLDB is Apple's Debugger

Scheme option to use standalone terminal

... and it's everywhere you need it
LLDB is Apple's Debugger

Scheme option to use standalone terminal

... and it's everywhere you need it
LLDB is Apple’s Debugger
... and it’s everywhere you need it

Scheme option to use standalone terminal
• Remainder of talk focused on LLDB commands

<table>
<thead>
<tr>
<th>Visual Debugging with Xcode</th>
<th>Presidio</th>
<th>Wednesday 4:00PM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread Sanitizer and Static Analysis</td>
<td>Mission</td>
<td>Thursday 10:00AM</td>
</tr>
</tbody>
</table>
LLDB is Apple’s Debugger
... and it’s everywhere you need it

Scheme option to use standalone terminal
• Remainder of talk focused on LLDB commands
The Swift REPL is LLDB

Visual Debugging with Xcode
Presidio
Wednesday 4:00PM

Thread Sanitizer and Static Analysis
Mission
Thursday 10:00AM
LLDB is Apple’s Debugger
... and it’s everywhere you need it

Scheme option to use standalone terminal
• Remainder of talk focused on LLDB commands

The Swift REPL is LLDB
• :<command> enables any LLDB command

Visual Debugging with Xcode          Presidio          Wednesday 4:00PM

Thread Sanitizer and Static Analysis Mission Thursday 10:00AM
$ swift
Welcome to Apple Swift version 3.0. Type :help for assistance.
   1>
$ swift
Welcome to Apple Swift version 3.0. Type :help for assistance.

1> :type lookup Comparable

protocol Comparable : Equatable {
    @warn_unused_result func <(lhs: Self, rhs: Self) -> Swift.Bool
    @warn_unused_result func <=(lhs: Self, rhs: Self) -> Swift.Bool
    @warn_unused_result func >=(lhs: Self, rhs: Self) -> Swift.Bool
    @warn_unused_result func >(lhs: Self, rhs: Self) -> Swift.Bool
}

1>
protocol Comparable : Equatable {
    @warn_unused_result func <(lhs: Self, rhs: Self) -> Swift.Bool
    @warn_unused_result func <=(lhs: Self, rhs: Self) -> Swift.Bool
    @warn_unused_result func >=(lhs: Self, rhs: Self) -> Swift.Bool
    @warn_unused_result func >(lhs: Self, rhs: Self) -> Swift.Bool
}

func abs<T : SignedNumber>(_ x: T) -> T
$ swift
Welcome to Apple Swift version 3.0. Type :help for assistance.

1> :type lookup Comparable

protocol Comparable : Equatable {
  @warn_unused_result func <(lhs: Self, rhs: Self) -> Swift.Bool
  @warn_unused_result func <=(lhs: Self, rhs: Self) -> Swift.Bool
  @warn_unused_result func >=(lhs: Self, rhs: Self) -> Swift.Bool
  @warn_unused_result func >(lhs: Self, rhs: Self) -> Swift.Bool
}

1> :type lookup abs

func abs<T : SignedNumber>(_ x: T) -> T

1> :type lookup Swift

import SwiftShims

struct UnsafePointer<Pointee> : Strideable, Hashable, _Pointer {
  typealias Distance = Swift.Int
  let _rawValue: Builtin.RawPointer
  init(_ _rawValue: Builtin.RawPointer)
  init(_ from: Swift.OpaquePointer)
  ...
$ swift
Welcome to Apple Swift version 3.0. Type :help for assistance.
  1>
$ swift
Welcome to Apple Swift version 3.0. Type :help for assistance.

    func greet() {
        print("Welcome to WWDC16!")
    }
$ swift
Welcome to Apple Swift version 3.0. Type :help for assistance.

1> func greet() {
2.     print("Welcome to WWDC16!"
3. }
4> :b 2
Breakpoint 1: where = $__lldb_expr2`__lldb_expr_1.greet () -> () + 4 at repl.swift:2,
address = 0x0000000100558074
...
$ swift
Welcome to Apple Swift version 3.0. Type :help for assistance.

    1> func greet() {
    2.       print("Welcome to WWDC16!")
    3. }
    4> :b 2
Breakpoint 1: where = $__lldb_expr2`__lldb_expr_1.greet () -> () + 4 at repl.swift:2,
address = 0x0000000100558074
...
    4> greet()
Execution stopped at breakpoint. Enter LLDB commands to investigate (type help for assistance.)
...
(lldb)
$ swift
Welcome to Apple Swift version 3.0. Type :help for assistance.

1> func greet() {
2.    print("Welcome to WWDC16!")
3. }
4> :b 2
Breakpoint 1: where = __lldb_expr2`__lldb_expr_1.greet () -> () + 4 at repl.swift:2, address = 0x0000000100558074
...
4> greet()
Execution stopped at breakpoint. Enter LLDB commands to investigate (type help for assistance.)
...
(lldb) bt
* thread #1: tid = 0xd698f, 0x000000010068f064 __lldb_expr2`greet() -> () + 4 at repl.swift:2, queue = 'com.apple.main-thread', stop reason = breakpoint 1.1
  * frame #0: 0x000000010068f064 __lldb_expr2`greet() -> () + 4 at repl.swift:2
  frame #1: 0x000000010068f84e __lldb_expr4`main + 94 at repl.swift:4
  frame #2: 0x0000000100000e00 repl_swift`_mh_execute_header + 3584
  frame #3: 0x00007fffd408d285 libdyld.dylib`start + 1

func greet() {
    print("Welcome to WWDC16!")
}
Welcome to Apple Swift version 3.0. Type :help for assistance.

1>
$ swift
Welcome to Apple Swift version 3.0. Type :help for assistance.
  1> :

(lldb)
$ swift
Welcome to Apple Swift version 3.0. Type :help for assistance.

   1> :

(lltdb) repl

   1>
LLDB as a Command-Line Tool
LLDB as a Command-Line Tool

Ideal for automating debugging tasks
LLDB as a Command-Line Tool

Ideal for automating debugging tasks
• Provide a file containing LLDB commands

```bash
lldb --source <filename>
```
Ideal for automating debugging tasks

• Provide a file containing LLDB commands
  `{lldb --source <filename>}`

• Provide LLDB commands without requiring a file
  `{lldb --one-line <command>}`
LLDB as a Command-Line Tool

Ideal for automating debugging tasks

- Provide a file containing LLDB commands
  ```
  lldb --source <filename>
  ```

- Provide LLDB commands without requiring a file
  ```
  lldb --one-line <command> -o <command2>
  ```
LLDB as a Command-Line Tool

Ideal for automating debugging tasks

- Provide a file containing LLDB commands
  
  `lldb --source <filename>`

- Provide LLDB commands without requiring a file
  
  `lldb --one-line <command> -o <command2>`

- Run a series of commands and exit – unless target crashes
  
  `lldb --batch --source <filename>`
LLDB as a Command-Line Tool

Ideal for automating debugging tasks

• Provide a file containing LLDB commands
  `lldb --source <filename>`

• Provide LLDB commands without requiring a file
  `lldb --one-line <command> -o <command2>`

• Run a series of commands and exit – unless target crashes
  `while true; lldb --batch --source <filename>; done`
LLDB as a Command-Line Tool

Ideal for automating debugging tasks

• Provide a file containing LLDB commands
  ```
  lldb --source <filename>
  ```

• Provide LLDB commands without requiring a file
  ```
  lldb --one-line <command> -o <command2>
  ```

• Run a series of commands and exit – unless target crashes
  ```
  while true; lldb --batch --source <filename>; done
  ```

Review  ```
  lldb --help
  ```  for details
Xcode 8 and LLDB: Distinct Processes

... and it's completely transparent
Xcode 8 and LLDB: Distinct Processes

... and it’s completely transparent

Multiple debugger versions supported
• Debugger selected automatically
Xcode 8 and LLDB: Distinct Processes

... and it’s completely transparent

Multiple debugger versions supported

- Debugger selected automatically
- Swift 3 uses latest debugger
  - As does pure Objective-C and C++
Xcode 8 and LLDB: Distinct Processes

... and it’s completely transparent

Multiple debugger versions supported

• Debugger selected automatically
• Swift 3 uses latest debugger
  - As does pure Objective-C and C++
• Swift 2.3 uses Xcode 7.3.1-era debugger
Xcode 8 and LLDB: Distinct Processes

... and it’s completely transparent

Multiple debugger versions supported

- Debugger selected automatically
- Swift 3 uses latest debugger
  - As does pure Objective-C and C++
- Swift 2.3 uses Xcode 7.3.1-era debugger
- Open source Swift uses matching debugger
Xcode 8 and LLDB: Distinct Processes

... and it’s completely transparent

Multiple debugger versions supported

• Debugger selected automatically
• Swift 3 uses latest debugger
  - As does pure Objective-C and C++
• Swift 2.3 uses Xcode 7.3.1-era debugger
• Open source Swift uses matching debugger

Xcode gracefully recovers when LLDB cannot
Customization and Introspection

Enrico Granata
Debugger Customization
Debugger Customization

Customize your debugger for greater awesomeness
Debugger Customization

Customize your debugger for greater awesomeness

Command aliases
Custom commands
Data formatters
Debugger Customization

Customize your debugger for greater awesomeness
Command aliases
Custom commands
Data formatters
Stepping actions
Command Aliases
Command Aliases

Create shorter syntax for frequent actions
Command Aliases

Create shorter syntax for frequent actions
Customize help text
(lldb) command alias
command alias -h "Run a command in the UNIX shell." --
(lldb) command alias -h "Run a command in the UNIX shell." -- shell
(lldb) command alias -h "Run a command in the UNIX shell." -- shell
(lldb) command alias -h "Run a command in the UNIX shell." -- shell platform shell
(lldb)
(lldb) command alias -h "Run a command in the UNIX shell." -- shell platform shell
(lldb) help shell
(lldb) command alias -h "Run a command in the UNIX shell." -- shell platform shell

(lldb) help shell

Run a command in the UNIX shell. This command takes 'raw' input (no need to quote stuff).

...
Run a command in the UNIX shell. This command takes 'raw' input (no need to quote stuff).

...
Run a command in the UNIX shell. This command takes 'raw' input (no need to quote stuff).

...
Scripting LLDB in Python
Scripting LLDB in Python

LLDB comes with a Python API
Scripting LLDB in Python

LLDB comes with a Python API

Get started
Scripting LLDB in Python

LLDB comes with a Python API

Get started

• Previous WWDC sessions

<table>
<thead>
<tr>
<th>Debugging with LLDB</th>
<th>WWDC 2012</th>
</tr>
</thead>
<tbody>
<tr>
<td>Advanced Debugging with LLDB</td>
<td>WWDC 2013</td>
</tr>
</tbody>
</table>
Scripting LLDB in Python

LLDB comes with a Python API

Get started

• Previous WWDC sessions
• http://lldb.llvm.org

Debugging with LLDB

Advanced Debugging with LLDB

| WWDC 2012 |
| WWDC 2013 |
Scripting LLDB in Python

LLDB comes with a Python API

Get started

• Previous WWDC sessions
• http://lldb.llvm.org

Community doing amazing things

<table>
<thead>
<tr>
<th>Debugging with LLDB</th>
<th>WWDC 2012</th>
</tr>
</thead>
<tbody>
<tr>
<td>Advanced Debugging with LLDB</td>
<td>WWDC 2013</td>
</tr>
</tbody>
</table>
Example
Example

A command to retrieve the return value of the last function call
Example

A command to retrieve the return value of the last function call

Only works if you **finish** your way out of a function
Example

A command to retrieve the return value of the last function call

Only works if you **finish** your way out of a function

- And don’t step!
(lldb) command script import ~/getreturn.py
(lldb)
(lldb) command script import ~/getreturn.py
(lldb) finish
...
Return value: (unsigned int) $0 = 2416832525
...
(lldb)
Return value: (unsigned int) $0 = 2416832525
frame #15: 0x00007fff81ad32c9 AE`dispatchEventAndSendReply(AEDesc const*, AEDesc*) + 39
frame #16: 0x00007fff81ad31d5 AE`aeProcessAppleEvent + 312
frame #17: 0x00007fff80285ae7 HIToolbox`AEProcessAppleEvent + 55
frame #18: 0x00007fff7e9e5583 AppKit`_DPSNextEvent + 1811
frame #19: 0x00007fff7f0eab6c AppKit`-[NSApplication(NSEvent)_nextEventMatchingEventMask:untilDate:inMode:dequeue:] + 670
frame #20: 0x00007fff7e9d9bf2 AppKit`-[NSApplication run] + 929
frame #21: 0x00007fff7e9a551f AppKit`NSApplicationMain + 1237
frame #22: 0x0000000100001462 WWDCrash`main(argc=1, argv=0x00007fff5fbff638) + 34 at main.m:12
frame #23: 0x00007fff94dc0285 libdyld.dylib`start + 1
frame #24: 0x00007fff94dc0285 libdyld.dylib`start + 1
(lldb)
... frame #15: 0x00007fff81ad32c9 AE\`dispatchEventAndSendReply(AEDesc const*, AEDesc*) + 39
frame #16: 0x00007fff81ad31d5 AE\`aeProcessAppleEvent + 312
frame #17: 0x00007fff80285ae7 HIToolbox\`AEProcessAppleEvent + 55
frame #18: 0x00007fff7e9e5583 AppKit\`_DPSNextEvent + 1811
frame #19: 0x00007fff7f0eab6c AppKit\`-[NSApplication(NSEvent)_nextEventMatchingEventMask:untilDate:inMode:dequeue:] + 670
frame #20: 0x00007fff7e9d9bf2 AppKit\`-[NSApplication run] + 929
frame #21: 0x00007fff7e9a551f AppKit\`NSApplicationMain + 1237
frame #22: 0x0000000100001462 WWDCrash\`main(argc=1, argv=0x00007fff5fbff638) + 34 at main.m:12
frame #23: 0x00007fff94dc0285 libdyld.dylib\`start + 1
frame #24: 0x00007fff94dc0285 libdyld.dylib\`start + 1
(lldb) getreturn
frame #15: 0x00007fff81ad32c9 AE`dispatchEventAndSendReply(AEDesc const*, AEDesc*) + 39
frame #16: 0x00007fff81ad31d5 AE`aeProcessAppleEvent + 312
frame #17: 0x00007fff8028ae7 HIToolbox`AEProcessAppleEvent + 55
frame #18: 0x00007fff7e9e5583 AppKit`_DPSNextEvent + 1811
frame #19: 0x00007fff7f0eab6c AppKit`-[NSApplication(NSEvent)_nextEventMatchingEventMask:untilDate:inMode:dequeue:] + 670
frame #20: 0x00007fff7e9d9bf2 AppKit`-[NSApplication run] + 929
frame #21: 0x00007fff7e9a551f AppKit`NSApplicationMain + 1237
frame #22: 0x0000000100001462 WWDCrash`main(argc=1, argv=0x00007fff5fbff638) + 34 at main.m:12
frame #23: 0x00007fff94dc0285 libdyld.dylib`start + 1
frame #24: 0x00007fff94dc0285 libdyld.dylib`start + 1

(lldb) getreturn
(unsigned int) $0 = 2416832525
(lldb)
class GetLatestReturnCommand:
    def __init__(self, debugger, session_dict):
        pass

    def __call__(self, debugger, command, exe_ctx, result):
        retval = exe_ctx.thread.GetStopReturnValue()
        T = retval.GetType().GetName()
        N = retval.GetName()
        V = retval.GetValue()
        S = retval.GetSummary()
        print >>result,"(%s) %s = %s" % (T, N, S if S else V if V else "")

    def get_short_help(self):
        return "Retrieve the last value returned by a function call on this thread."

    def __lldb_init_module(debugger, *args):
        debugger.HandleCommand('com scr add --class command.GetLatestReturnCommand getreturn')
Persistent Customizations
Persistent Customizations

Save yourself from repetitive typing
Persistent Customizations

Save yourself from repetitive typing

Initialization file:
Persistent Customizations

Save yourself from repetitive typing

Initialization file:

~/.lldbinit
Persistent Customizations

Save yourself from repetitive typing

Initialization file:

~/.lldbinit

Xcode specific: ~/.lldbinit-Xcode
Persistent Customizations

Save yourself from repetitive typing

Initialization file:

~/.lldbindit

Xcode specific: ~/.lldbindit-Xcode

Python at startup: use `command script import`
Data Served Three Ways
Data Served Three Ways

\[ p <expression> \]
\[ po <expression> \]
Data Served Three Ways

- $p <expression>$
- $po <expression>$
Data Served Three Ways

- $p <\text{expression}>$
- $po <\text{expression}>$
- $+\text{Full expressions}$
Data Served Three Ways

- $p \ <\text{expression}>$
- $po \ <\text{expression}>$
- +Full expressions
  - Executed in target
Data Served Three Ways

- p <expression>
- po <expression>

+ Full expressions
  - Executed in target
  - Not always possible
Data Served Three Ways

- \( p \) \(<\text{expression}>\)
- \( po \) \(<\text{expression}>\)

- +Full expressions
  - Executed in target
  - Not always possible
Data Served Three Ways

p <expression>

+ Full expressions
  - Executed in target
  - Not always possible

± Customized by type author
Data Served Three Ways

- p <expression>
- po <expression>

- +Full expressions
  - Executed in target
  - Not always possible

- ±Customized by type author
  - Executed in target
Data Served Three Ways

- **e** \( p \ < \text{expression}> \)
- **o e** \( po \ < \text{expression}> \)
- Frame variable \( < \text{local-name}> \)

- **e** +Full expressions
  - Executed in target
  - Not always possible

- **o** ±Customized by type author
  - Executed in target
Data Served Three Ways

- **e** p <expression>
- **o e** po <expression>
- **f** frame variable <local-name>

**e**
- +Full expressions
  - Executed in target
  - Not always possible

**o**
- ±Customized by type author
  - Executed in target
Data Served Three Ways

- **e** p <expression>
- **o e** po <expression>
- **f** frame variable <local-name>

**e** +Full expressions
- Executed in target
- Not always possible

**o** ±Customized by type author
- Executed in target

**f** +Extremely predictable
Data Served Three Ways

- p <expression>
- po <expression>
- frame variable <local-name>

+ Full expressions
  - Executed in target
  - Not always possible

± Customized by type author
  - Executed in target

+ Extremely predictable
  - Limited syntax
Data Served Three Ways

Many

\( p \ <\text{expression}> \)

\( po \ <\text{expression}> \)

\( \text{frame variable } <\text{local-name}> \)

\( +\text{Full expressions} \)
  - Executed in target
  - Not always possible

\( \pm\text{Customized by type author} \)
  - Executed in target

\( +\text{Extremely predictable} \)
  - Limited syntax
Data Served Three Ways

Many

- **e** p <expression>
- **o e** po <expression>
- **f** frame variable <local-name>
- **e** parray <count> <expression>
- **o e** poarray <count> <expression>
- **e** +Full expressions
  - Executed in target
  - Not always possible
- **o** ±Customized by type author
  - Executed in target
- **f** +Extremely predictable
  - Limited syntax
Example

```c
int main() {
    int count = 0;
    int* dataset = readData(count);
    for (int i = 0; i < count; i++)
        processElement(dataset[i]);
    return 0;
}
```
Example

```c
int main() {
    int count = 0;
    int* dataset = readData(count);
    for (int i = 0; i < count; i++)
        processElement(dataset[i]);
    return 0;
}
```

C pointers have no notion of element count
int main() {
    int count = 0;
    int* dataset = readData(count);
    for (int i = 0; i < count; i++)
        processElement(dataset[i]);
    return 0;
}
```c
int main() {
    int count = 0;
    int* dataset = readData(count);
    for (int i = 0; i < count; i++)
        processElement(dataset[i]);
    return 0;
}
```
(lldb) p dataset
(int *) $0 = 0x0000000000000260
(lldb)

```c
int main() {
    int count = 0;
    int* dataset = readData(count);
    for (int i = 0; i < count; i++)
        processElement(dataset[i]);
    return 0;
}
```
(lldb) p dataset
(int *) $0 = 0x0000000100200260
(lldb) p dataset[0]
(int *) $1 = 0
(lldb)

```c
int main() {
    int count = 0;
    int* dataset = readData(count);
    for (int i = 0; i < count; i++)
        processElement(dataset[i]);
    return 0;
}
```
(lldb) p dataset
(int *) $0 = 0x0000000100200260
(lldb) p dataset[0]
(int *) $1 = 0
(lldb) p dataset[1]
(int) $2 = 16842769
...

int main() {
    int count = 0;
    int* dataset = readData(count);
    for (int i = 0; i < count; i++)
        processElement(dataset[i]);
    return 0;
}
```c
int main() {
    int count = 0;
    int* dataset = readData(count);
    for (int i = 0; i < count; i++)
        processElement(dataset[i]);
    return 0;
}
```
int main() {
    int count = 0;
    int* dataset = readData(count);
    for (int i = 0; i < count; i++)
        processElement(dataset[i]);
    return 0;
}
int main() {
    int count = 0;
    int* dataset = readData(count);
    for (int i = 0; i < count; i++)
        processElement(dataset[i]);
    return 0;
}
```c
int main() {
    int count = 0;
    int* dataset = readData(count);
    for (int i = 0; i < count; i++)
        processElement(dataset[i]);
    return 0;
}
```
```c
int main() {
    int count = 0;
    int* dataset = readData(count);
    for (int i = 0; i < count; i++)
        processElement(dataset[i]);
    return 0;
}
```
int main() {
    int count = 0;
    int* dataset = readData();
    for (int i = 0; i < count; i++)
        processElement(dataset[i]);
    return 0;
}
int main() {
    int count = 0;
    int* dataset = readData();
    for (int i = 0; i < count; i++)
        processElement(dataset[i]);
    return 0;
}
int main() {
    int count = 0;
    int* dataset = readData();
    for (int i = 0; i < count; i++)
        processElement(dataset[i]);
    return 0;
}
```c
int main() {
    int count = 0;
    int* dataset = readDataset();
    for (int i = 0; i < count; i++)
        processElement(dataset[i]);
    return 0;
}
```
(lldb)
(lldb) po array `numCustomers` customers
(lldb) poarray `numCustomers` customers
{
    Kate, 1 Infinite Loop, Cupertino, CA
    Enrico, 700 Swift Street, Mountain View, CA
    Sean, SF MoMA, San Francisco, CA
    Jim, He Won’t Tell Me Blvd., Somewhere, CA
}
(lldb)
Exploring Memory Addresses
Exploring Memory Addresses

(lldb) po 0x1003183e0
Enrico, 700 Swift Street, Mountain View, CA
Exploring Memory Addresses

(lldb) po 0x1003183e0
Enrico, 700 Swift Street, Mountain View, CA

(lldb) po 0x1003183e0
4298212320
Exploring Memory Addresses
Exploring Memory Addresses

(lldb) expr -O --language objc -- 0x1003183e0
Enrico, 700 Swift Street, Mountain View, CA
Low-Level Debugging
Low-Level Debugging

First rule: don’t!
Low-Level Debugging

First rule: *don’t*

Optimized code
Low-Level Debugging

First rule: don’t!

Optimized code

Third-party code with no debug info
Low-Level Debugging

First rule: *don’t!*

Optimized code

Third-party code with no debug info

Proceed at your own risk
Reading Registers

Read processor register values
- All registers or only a few
- Apply custom formats
(lldb) register read
General Purpose Registers:

rax = 0x0000000000000000
rbx = 0x0000000000000000
rcx = 0x0000000000000000
rdx = 0x0000000000000020
rdi = 0x00000000000004d2
rsi = 0x00007fff914ab7df  "stringByAppendingString:"
rbp = 0x00007fff5fbfcf60
rsp = 0x00007fff5fbfcf28
r8  = 0x0000000000000010
r9  = 0x0000000100100080
r10 = 0x0000000000000ffffff00
r11 = 0x000000000000000000292
r12 = 0x0000000000000000000000
r13 = 0x00007fff5fbfcf70
r14 = 0x000000000100096000

...
Reading Registers
Reading Registers

Arguments often passed in registers
Reading Registers

Arguments often passed in registers
Mapping given by Application Binary Interface (ABI)
Reading Registers

Arguments often passed in registers
Mapping given by Application Binary Interface (ABI)
Debugger exposes $arg1, $arg2 \ldots$ pseudo-registers
Reading Registers

Arguments often passed in registers
Mapping given by Application Binary Interface (ABI)
Debugger exposes $arg1, $arg2 ... pseudo-registers
Map one-to-one for scalar/pointer arguments
Reading Registers

Arguments often passed in registers
Mapping given by Application Binary Interface (ABI)
Debugger exposes $\texttt{arg1}, \texttt{arg2} \ldots$ pseudo-registers
Map one-to-one for scalar/pointer arguments
Available in C-family expressions
int function(int some, void* thing, char more) {

    // great code

}
int function(int some, void* thing, char more) {

    // great code

}

function(12, myBuffer, 'Q');
int function(int some, void* thing, char more) {
    // great code
}

function(12, myBuffer, 'Q');
int function(int some, void* thing, char more) {

    // great code

}

function(12, myBuffer, 'Q');
int function(int some, void* thing, char more) {
    // great code
}

function(12, myBuffer, 'Q');
int function(int some, void* thing, char more) {
    // great code
}

function(12, myBuffer, 'Q');
(lldb) frame select 0
frame #0: 0x00007fff8eae04d7 libobjc.A.dylib`objc_msgSend + 23
libobjc.A.dylib`objc_msgSend:
   0x7fff8eae04d7 <+23>: andq   (%rdi), %r10
   0x7fff8eae04da <+26>: movq   %rsi, %r11
   0x7fff8eae04dd <+29>: andl   0x18(%r11), %r11d
   0x7fff8eae04e1 <+33>: shlq   $0x4, %r11
(lldb)
(lldb) frame select 0
frame #0: 0x00007fff8eae04d7 libobjc.A.dylib\objc_msgSend + 23
libobjc.A.dylib\objc_msgSend:
->  0x7fff8eae04d7 <+23>: andq (%rdi), %r10
  0x7fff8eae04da <+26>: movq %rsi, %r11
  0x7fff8eae04dd <+29>: andl 0x18(%r11), %r11d
  0x7fff8eae04e1 <+33>: shlq $0x4, %r11

(lldb) register read $arg1 $arg2
  rdi = 0x00000000000004d2
  rsi = 0x00007fff914ab7df  "stringByAppendingString:"
(lldb) frame select 0
frame #0: 0x000007fff8eae04d7 libobjc.A.dylib\objc_msgSend + 23
libobjc.A.dylib\objc_msgSend:
    0x7fff8eae04d7 <+23>: andq (%rdi), %r10
    0x7fff8eae04da <+26>: movq %rsi, %r11
    0x7fff8eae04dd <+29>: andl 0x18(%r11), %r11d
    0x7fff8eae04e1 <+33>: shlq $0x4, %r11
    0x7fff8eae04e5 <+37>: addq %rax, %rsp

(lldb) register read $arg1 $arg2
    rdi = 0x00000000000004d2
    rsi = 0x00007fff914ab7df "stringByAppendingString:"

(lldb) memory read $arg1
error: memory read failed for 0x400
(lldb)
id objc_msgSend(id object, SEL selector, ...)
id objc_msgSend(id object, SEL selector, ...)

objc_msgSend(                     ,                                  )

0x000000000000000000000000000004d2

stringByAppendingString:

$arg1

$arg2
id objc_msgSend(id object, SEL selector, ...)

objc_msgSend(0x00000000000004d2,stringByAppendingString:)

$arg1 $arg2

0x00000000000004d2

stringByAppendingString:

$arg1 $arg2
id objc_msgSend(id object, SEL selector, ...)  

objc_msgSend(0x00000000000004d2, stringByAppendingString:)

Calling selector on a bad object

0x00000000000004d2

stringByAppendingString:

$arg1

$arg2
Stack Frames
Stack Frames

frame 0

objc_msgSend
Stack Frames

frame N

main
NSApplicationMain
myFunction1
myFunction2
-didFinishLaunching:
objc_msgSend

frame 0
Stack Frames

- main
- NSApplicationMain
- myFunction1
- myFunction2
- -didFinishLaunching:
- objc_msgSend
Stack Frames

- main
- NSApplicationMain
- myFunction1
- myFunction2
- -didFinishLaunching:
- objc_msgSend

Current Frame
Stack Frames

Current Frame

- didFinishLaunching: 
- objc_msgSend
- myFunction2
- myFunction1
- NSApplicationMain
- main

(lldb) up
(lldb)
Stack Frames

(lldb) up
(lldb)
Stack Frames

Current Frame

- main
- NSAapplicationMain
- myFunction1
- myFunction2
- -didFinishLaunching:
- objc_msgSend
Stack Frames

(lldb)

Current Frame

main
NSApplicationMain
myFunction1
myFunction2
-didFinishLaunching:
objc_msgSend
Stack Frames

Current Frame

main
NSApplicationMain
myFunction1
myFunction2
-didFinishLaunching:
objc_msgSend

(llldb) down
(llldb)
Disassembly

The disassemble command shows disassembled machine code

- For the current frame, an address, or a function
Disassembly

The disassemble command shows disassembled machine code

• For the current frame, an address, or a function

Customize disassembly format:

• Intel vs. AT&T
• Entirely custom format (disassembly-format setting)
Disassembly

The disassemble command shows disassembled machine code

• For the current frame, an address, or a function

Customize disassembly format:

• Intel vs. AT&T

• Entirely custom format (disassembly-format setting)

Show disassembly:

• When no source or no debug info

• Always

• Never
(lldb) bt
* thread #1: tid = 0x158321, 0x00007fff8eae04d7 libobjc.A.dylib`objc_msgSend + 23, queue = 'com.apple.main-thread', stop reason = EXC_BAD_ACCESS (code=1, address=0x4d2)
  * frame #0: 0x00007fff8eae04d7 libobjc.A.dylib`objc_msgSend + 23
    frame #1: 0x000000010000149f WWDCrash`-[AppDelegate applicationDidFinishLaunching:]
      (self=0x000000010000149f, _cmd="applicationDidFinishLaunching:", aNotification="@"NSApplicationDidFinishLaunchingNotification"`) + 47 at AppDelegate.m:24
...
(lldb)
(lldb) up
...
(lldb)
(lldb) up
...
(lldb) disassemble --frame
...

0x100001484 <+20>: callq 0x100001460 ; getGlobalToken
0x100001489 <+25>: leaq 0xb90(%rip), %rdx ; @"magicText"
0x100001490 <+32>: movq 0x14a9(%rip), %rsi ; "stringByAppendingString:"
0x100001497 <+39>: movq %rax, %rdi
0x10000149a <+42>: callq 0x1000014c2 ; symbol stub for: objc_msgSend
...
(lldb)
0x100001484 <+20>: callq 0x100001460 ; getGlobalToken
0x100001489 <+25>: leaq 0xb90(%rip), %rdx ; @"magicText"
0x100001490 <+32>: movq 0x14a9(%rip), %rsi ; "stringByAppendingString:"
0x100001497 <+39>: movq %rax, %rdi
0x10000149a <+42>: callq 0x1000014c2 ; symbol stub for: objc_msgSend

(llldb)
(lldb) ni
0x100001484 <+20>: callq 0x100001460 ; getGlobalToken
0x100001489 <+25>: leaq 0xb90(%rip), %rdx ; @"'magicText'"
0x100001490 <+32>: movq 0x14a9(%rip), %rsi ; "stringByAppendingString:"
0x100001497 <+39>: movq %rax, %rdi
0x10000149a <+42>: callq 0x1000014c2 ; symbol stub for: objc_msgSend

(llldb)
(lldb) reg read rax

0x100001484 <+20>: callq 0x100001460 ; getGlobalToken
0x100001489 <+25>: leaq 0xb90(%rip), %rdx ; @"magicText"
0x100001490 <+32>: movq 0x14a9(%rip), %rsi ; "stringByAppendingString:"
0x100001497 <+39>: movq %rax, %rdi
0x10000149a <+42>: callq 0x1000014c2 ; symbol stub for: objc_msgSend
rax = 0x00000000000004d2

(lldb) reg read rax
    rax = 0x00000000000004d2

(lldb)
rax = 0x00000000000004d2

reg read rax

rax = 0x00000000000004d2

(si)
0x100001484 <+20>: callq 0x100001460 ; getGlobalToken
0x100001489 <+25>: leaq 0xb90(%rip), %rdx ; @"magicText"
0x100001490 <+32>: movq 0x14a9(%rip), %rsi ; "stringByAppendingString:
0x100001497 <+39>: movq %rax, %rdi
0x10000149a <+42>: callq 0x1000014c2 ; symbol stub for: objc_msgSend
0x100001484 <+20>: callq 0x100001460 ; getGlobalToken
0x100001489 <+25>: leaq 0xb90(%rip), %rdx ; @"magicText"
0x100001490 <+32>: movq 0x14a9(%rip), %rsi ; "stringByAppendingString:"
0x100001497 <+39>: movq %rax, %rdi
0x10000149a <+42>: callq 0x1000014c2 ; symbol stub for: objc_msgSend

(lldb)
(lldb) si

0x100001484 <+20>: callq 0x100001460 ; getGlobalToken
0x100001489 <+25>: leaq 0xb90(%rip), %rdx ; @"magicText"
0x100001490 <+32>: movq 0x14a9(%rip), %rsi ; "stringByAppendingString:"
0x100001497 <+39>: movq %rax, %rdi
0x10000149a <+42>: callq 0x1000014c2 ; symbol stub for: objc_msgSend
0x100001484 <+20>:  callq  0x100001460 ; getGlobalToken
0x100001489 <+25>:  leaq  0xb90(%rip), %rdx  ; @"magicText"
0x100001490 <+32>:  movq  0x14a9(%rip), %rsi  ; "stringByAppendingString:"
0x100001497 <+39>:  movq  %rax, %rdi
0x10000149a <+42>:  callq  0x1000014c2 ; symbol stub for: objc_msgSend
0x100001484 <+20>: callq 0x100001460 ; getGlobalToken
0x100001489 <+25>: leaq 0xb90(%rip), %rdx ; @"magicText"
0x100001490 <+32>: movq 0x14a9(%rip), %rsi ; "stringByAppendingString:"
0x100001497 <+39>: movq %rax, %rdi
0x10000149a <+42>: callq 0x1000014c2 ; symbol stub for: objc_msgSend

(lldb)
0x100001484 <+20>: callq 0x100001460 ; getGlobalToken
0x100001489 <+25>: leaq 0xb90(%rip), %rdx ; "magicText"
0x100001490 <+32>: movq 0x14a9(%rip), %rsi ; "stringByAppendingString:"
0x100001497 <+39>: movq %rax, %rdi
0x10000149a <+42>: callq 0x1000014c2 ; symbol stub for: objc_msgSend

(lldb) si
0x100001484 <+20>: callq 0x100001460 ; getGlobalToken
0x100001489 <+25>: leaq 0xb90(%rip), %rdx ; @"magicText"
0x100001490 <+32>: movq 0x14a9(%rip), %rsi ; "stringByAppendingString:"
0x100001497 <+39>: movq %rax, %rdi
0x10000149a <+42>: callq 0x1000014c2 ; symbol stub for: objc_msgSend

(lldb)
(lldb) reg read $arg1

0x100001484 <+20>: callq 0x100001460 ; getGlobalToken
0x100001489 <+25>: leaq 0xb90(%rip), %rdx ; @"magicText"
0x100001490 <+32>: movq 0x14a9(%rip), %rsi ; "stringByAppendingString:"
0x100001497 <+39>: movq %rax, %rdi
0x10000149a <+42>: callq 0x1000014c2 ; symbol stub for: objc_msgSend
reg read $arg1

rdi = 0x00000000000004d2
Low-Level Debugging
Low-Level Debugging

getGlobalToken() returns an invalid object
getGlobalToken() returns an invalid object
Subsequent usage causes a crash
Expression Parsing
Better and more convenient

Sean Callanan
@import AppKit;
void doSomeWork() {
    NSLog(@"Doing work");
}

Thread 1: breakpoint 1.1
@import AppKit;
void doSomeWork() {
    NSLog(@"Doing work");
}

(llldb)
SDK Modules in Objective-C

```objective-c
@import AppKit;
void doSomeWork() {
    NSLog(@"Doing work");
}
```

```
(llldb) p [NSApplication sharedApplication].undoManager
```
SDK Modules in Objective-C

@import AppKit;
void doSomeWork() {
    NSLog(@"Doing work");
}

(lldb) p [NSApplication sharedApplication].undoManager
error: property ‘undoManager’ not found on object of type ‘id’ 😠
SDK Modules in Objective-C

```objective-c
@import AppKit;
void doSomeWork() {
    NSLog(@"Doing work");
}
```
SDK Modules in Objective-C

```objective-c
@import AppKit;
void doSomeWork() {
    NSLog(@"Doing work");
}
```
SDK Modules in Objective-C

```objective-c
@import AppKit;
void doSomeWork() {
    NSLog(@"Doing work");
}
```

(llldb) p @import AppKit 🙄
(llldb)
SDK Modules in Objective-C

```objective-c
@import AppKit;
void doSomeWork() {
    NSLog(@"Doing work");
}
```

```
(lldb) p @import AppKit 😐
(lldb) p [NSApplication sharedApplication].undoManager
(NSUndoManager * __nullable) $1 = 0x00007fb399629cd0
```
SDK Modules in Objective-C

@import AppKit;
void doSomeWork() {
    NSLog(@"Doing work");
}

(lldb) p [NSApplication sharedApplication].undoManager
(NSUndoManager * __nullable) $1 = 0x00007fb399629cd0
SDK Modules in Objective-C

```objective-c
@import AppKit;
void doSomeWork() {
    NSLog(@"Doing work");
}
```

```
(lldb) p [NSApplication sharedApplication].undoManager
(NSUndoManager * __nullable) $1 = 0x00007fb399629cd0
```

Controlled by

```
(lldb) settings show target.auto-import-clang-modules false
```
Reusable Code

Swift
Reusable Code

Swift

(lldb)
Reusable Code

Swift

(lldb) expr let a = 3; print(a)
3

(lldb)
Reusable Code

Swift

(lldb) expr let a = 3; print(a)
3

(lldb) expr a

error: <EXPR>:3:1: error: use of unresolved identifier 'a'
Swift

```swift
func functionYouAreStoppedIn() {
    doSomeWork()

    if (true) {
        let a = 3; print(a)
    }
}
```
Reusable Code

Swift

(lldb) expr let $a = 3; print($a)
3

(lldb) expr $a
(Int) $R1 = 3
Defining Reusable Functions

Swift

(lldb)
Defining Reusable Functions

Swift

(lldb) expr

Enter expressions, then terminate with an empty line to evaluate:
Defining Reusable Functions

Swift

(lldb) expr

Enter expressions, then terminate with an empty line to evaluate:

1: func $addTwoNumbers(a: Int, b: Int) -> Int {
2:     return a + b
3: }
4:

(llldb)
Defining Reusable Functions

Swift

(lldb) expr

Enter expressions, then terminate with an empty line to evaluate:

1: func $addTwoNumbers(a: Int, b: Int) -> Int {
2:     return a + b
3: }
4:

(lldb) expr $addTwoNumbers(a: 2, b: 3)
(Int) $R0 = 5
Defining Reusable Functions

C, C++, and Objective-C

(lldb)
Defining Reusable Functions

C, C++, and Objective-C

(lldb) expr

Enter expressions, then terminate with an empty line to evaluate:
Defining Reusable Functions

C, C++, and Objective-C

(int $addTwoNumbers(int a, int b) {
    return a + b
})
Defining Reusable Functions

C, C++, and Objective-C

(lldb) expr

Enter expressions, then terminate with an empty line to evaluate:

1: int $addTwoNumbers(int a, int b) {
2:     return a + b
3: }
4:

error: function declaration is not allowed here
error: 1 error parsing expression
Why Not?

Swift
Why Not?

Swift

```swift
func functionYouAreStoppedIn() {
    func $addTwoNumbers(…) -> Int {
    }
}
```
func functionYouAreStoppedIn() {
    func $addTwoNumbers(…) -> Int {
    }
}

Why Not?

Swift

```swift
func functionYouAreStoppedIn() {
    func $addTwoNumbers(...) -> Int {
    }
}
```

C, C++, and Objective-C

```c
void functionYouAreStoppedIn() {
    int $addTwoNumbers(...) {
    }
}
```
Why Not?

Swift

```swift
func functionYouAreStoppedIn() {
    func $addTwoNumbers(…) -> Int {
    }
}
```

C, C++, and Objective-C

```c
void functionYouAreStoppedIn() {
    int $addTwoNumbers(...) {
    }
}
```
Defining Reusable Functions

C, C++, and Objective-C
Defining Reusable Functions

C, C++, and Objective-C

(lldb)
Defining Reusable Functions

C, C++, and Objective-C

(lldb) expr --top-level --

Enter expressions, then terminate with an empty line to evaluate:
Defining Reusable Functions

C, C++, and Objective-C

(lldb) expr --top-level --

Enter expressions, then terminate with an empty line to evaluate:

1: int $addTwoNumbers(int a, int b) {
2:   return a + b;
3: }
4: 

(lldb)
Defining Reusable Functions

C, C++, and Objective-C

(lldb) expr --top-level --

Enter expressions, then terminate with an empty line to evaluate:

1: int $addTwoNumbers(int a, int b) {
2:   return a + b;
3: }
4:   

(lldb) expr $addTwoNumbers(2,3)

(Int) $R0 = 5
Defining Reusable Closures

Swift
Defining Reusable Closures

Swift

(lldb)
Defining Reusable Closures

Swift

(lldb) p let $add = { (a:Int, b:Int) in a+b }

(lldb)
Defining Reusable Closures

Swift

```
(lldb) p let $add = { (a:Int, b:Int) in a+b }
(lldb) p $add(s.startIndex, s.count)
(Int) $R0 = 6
```
Defining Reusable Closures

Blocks

Swift

```swift
(lldb) p let $add = { (a: Int, b: Int) in a + b }
(lldb) p $add(s.startIndex, s.count)
(Int) $R0 = 6
```

C, C++, and Objective-C

```c
(lldb) p int (^$add)(int, int) =
    ^(int a, int b) { return a + b; }
(lldb) p $add(r.location, r.length)
(int) $0 = 4
```
Defining Reusable Closures

Blocks and lambdas

```cpp
(lldb) p auto $add = [](int a, int b) {
    return a+b;
}
(lldb) p $add(a.offset,a.elements().size())
(int) $0 = 4
```
Passing Blocks to Functions

Objective-C
Passing Blocks to Functions

Objective-C

(lldb)
Passing Blocks to Functions

Objective-C

```objective-c
(lldb) p dispatch_sync(dispatch_get_global_queue(0,0),
^(){ printf("Hello world\n"); });
Hello world
```
Passing Blocks to Functions

Objective-C

(lldb) p dispatch_sync(dispatch_get_global_queue(0,0),
    ^(){ printf("Hello world\n") });
Passing Blocks to Functions

Objective-C

```
(lldb) p dispatch_sync(dispatch_get_global_queue(0,0),
    ^(){ printf("Hello world\n") });
```

error: expected ';' after expression
error: 1 error parsing expression
Passing Blocks to Functions

Objective-C

(lldb) p dispatch_sync(dispatch_get_global_queue(0,0),
    ^(){ printf("Hello world\n") });

Hello world

Fixit applied, fixed expression was:

dispatch_sync(dispatch_get_global_queue(0,0),
    ^(){ printf("Hello world\n") });
Fix-Its Work in Swift, Too!

Swift
Fix-Its Work in Swift, Too!

Swift

(lldb)
Fix-Its Work in Swift, Too!

Swift

(lldb) p let $myInt : Int? = 3
(lldb)
Fix-Its Work in Swift, Too!

Swift

(lldb) p let $myInt : Int? = 3
(lldb) p $myInt + 2
Swift

```swift
(lldb) p let $myInt : Int? = 3
(lldb) p $myInt + 2
(Int) $R0 = 5
    Fixit applied, fixed expression was:
    $myInt! + 2
```
Swift

(let) p let $myInt : Int? = 3
(let) p $myInt + 2

(Int) $R0 = 5
   Fixit applied, fixed expression was:
   $myInt! + 2

Controlled by

(let) settings set target.auto-apply-fixits false
(let) settings set target.notify-about-fixits false
Defining Reusable Types

Swift
Defining Reusable Types

Swift

(lldb)
Defining Reusable Types

Swift

(lldb) expr

Enter expressions, then terminate with an empty line to evaluate:
Defining Reusable Types

Swift

(lldb) expr

Enter expressions, then terminate with an empty line to evaluate:

1: class $MyClass {
2:   let m_a: Int
3:   init(a: Int) { m_a = a }
4: }

(lldb)
Swift

(lldb) expr

Enter expressions, then terminate with an empty line to evaluate:

1: class $MyClass {
2:   let m_a: Int
3:   init(a: Int) { m_a = a }
4: }

(lldb) expr $MyClass(a:1)

($MyClass) $R0 = 0x00000001010023e0 (m_a = 1)
Defining Reusable Types

C++

(lldb)
Defining Reusable Types

C++

(lldb) expr

Enter expressions, then terminate with an empty line to evaluate:
Defining Reusable Types

C++

(lldb) expr

Enter expressions, then terminate with an empty line to evaluate:

1: class $MyClass {
2:   int m_a;
3:   $MyClass(int a) : m_a(a) { }
4: }

(lldb)
Defining Reusable Types

C++

(lldb) expr

Enter expressions, then terminate with an empty line to evaluate:

1: class $MyClass {
2:   int m_a;
3:   $MyClass(int a) : m_a(a) { }
4: }

(lldb) expr $MyClass(1)

($MyClass) $0 = (m_a = 1)
Example

User-defined predicates

Objective-C

(lldb)
Example
User-defined predicates

Objective-C

(lldb) expr

Enter expressions, then terminate with an empty line to evaluate:
Example

User-defined predicates

Objective-C

(lldb) expr

Enter expressions, then terminate with an empty line to evaluate:

1: NSPredicate **p =
2:   [NSPredicate predicateWithBlock:
3:     ^(NSString *str, NSDictionary *bind) {
4:         return [str containsString:@"error"]; }


Example
User-defined predicates

Objective-C

(lldb) expr

Enter expressions, then terminate with an empty line to evaluate:

1: NSPredicate *$p =
2:   [NSPredicate predicateWithBlock:
3:     ^(NSString *str, NSDictionary *bind) {
4:       return [str containsString:@"error"]; }

Example
User-defined predicates

Objective-C

(lldb) **expr**

Enter expressions, then terminate with an empty line to evaluate:

1: NSPredicate **$p** =
2:   [NSPredicate predicateWithBlock:
3:     ^(NSString *str, NSDictionary *bind) {
4:       return [str containsString:@"error"]; }]

(lldb)
Example

User-defined predicates

Objective-C

(lldb) expr

Enter expressions, then terminate with an empty line to evaluate:

1: NSPredicate *$p =
2:     [NSPredicate predicateWithBlock:
3:         ^(NSString *str, NSDictionary *bind) {
4:             return [str containsString:@"error"]; }

(lldb) po [messages filteredArrayUsingPredicate:$p]
Example
User-defined predicates

Objective-C

(lldb) expr

Enter expressions, then terminate with an empty line to evaluate:

1: NSPredicate *$p =
2:   [NSPredicate predicateWithBlock:
3:     ^(NSString *str, NSDictionary *bind) {
4:       return [str containsString:@"error"];
    ]

(lldb) po [messages filteredArrayUsingPredicate:$p]

<__NSSingleObjectArrayI 0x100307f90>(
error parsing JSON
)
Breakpoints and Troubleshooting

Jim Ingham
Breakpoints
Breakpoints

Simple notion:
Breakpoints

Simple notion:

• Breakpoints stop your program
Simple notion:

- Breakpoints stop your program

LLDB's view:
Breakpoints

Simple notion:

• Breakpoints stop your program

LLDB's view:

• Breakpoints are searches for places to stop
Breakpoints

Simple notion:
• Breakpoints stop your program

LLDB's view:
• Breakpoints are searches for places to stop
• Breakpoints specify search criteria
Breakpoints

Simple notion:
- Breakpoints stop your program

LLDB's view:
- Breakpoints are searches for places to stop
- Breakpoints specify search criteria
- Search hits are *actual* places to stop: "Breakpoint Locations"
Xcode's Breakpoints
Xcode's Breakpoints

Xcode breakpoints are LLDB breakpoints
Xcode's Breakpoints

Xcode breakpoints are LLDB breakpoints
Created from editor gutter; LLDB does:

(lldb)
Xcode's Breakpoints

Xcode breakpoints are LLDB breakpoints
Created from editor gutter; LLDB does:

(lldb) break set --line 36 --file GreatCode.swift
Xcode's Breakpoints

Xcode breakpoints are LLDB breakpoints
Created from editor gutter; LLDB does:

```
(lldb) break set --line 36 --file GreatCode.swift
```

Symbolic breakpoint; LLDB does:

```
(lldb)
```
Xcode's Breakpoints

Xcode breakpoints are LLDB breakpoints
Created from editor gutter; LLDB does:

```
(lldb) break set --line 36 --file GreatCode.swift
```

Symbolic breakpoint; LLDB does:

```
(lldb) break set --name Foo
```
Multiple Locations

When and why?
Multiple Locations

When and why?

All breakpoints are searches
Multiple Locations

When and why?

All breakpoints are searches
Multiple results are always possible
Multiple Locations
When and why?

All breakpoints are searches
Multiple results are always possible
Let's see some examples
Multiple Locations

When and why?

All breakpoints are searches
Multiple results are always possible
Let's see some examples

• First for symbolic breakpoints:
(lldb) break set --name main
Breakpoint 1: 19 locations.
(lldb)
(lldb) break set --name main
Breakpoint 1: 19 locations.

(lldb) break list 1
1: name = 'main', locations = 19
   1.1: where = Sketch`main + 55 at SKTMain.m:17
   1.2: where = Foundation`-[NSThread main]
   1.3: where = Foundation`-[NSBlockOperation main]

...
(lldb) break set --name main

Breakpoint 1: 19 locations.

(lldb) break list 1

1: name = 'main', locations = 19
   1.1: where = Sketch`main + 55 at SKTMain.m:17
   1.2: where = Foundation`-[NSThread main]
   1.3: where = Foundation`-[NSBlockOperation main]

...
(lldb) break set --name main
Breakpoint 1: 19 locations.
(lldb) break list 1
1: name = 'main', locations = 19
   1.1: where = Sketch`main + 55 at SKTMain.m:17
   1.2: where = Foundation`-[NSThread main]
   1.3: where = Foundation`-[NSBlockOperation main]
...
(lldb)
(lldb) break set --name main
Breakpoint 1: 19 locations.
(lldb) break list 1
1: name = 'main', locations = 19
   1.1: where = Sketch`main + 55 at SKTMain.m:17
   1.2: where = Foundation`-[NSThread main]
   1.3: where = Foundation`-[NSBlockOperation main]
...

(lldb) break set --fullname main

Try a full-name breakpoint
(lldb) break set --name main
Breakpoint 1: 19 locations.
(lldb) break list 1
1: name = 'main', locations = 19
   1.1: where = Sketch`main + 55 at SKTMain.m:17
   1.2: where = Foundation`-[NSThread main]
   1.3: where = Foundation`-[NSBlockOperation main]
...
(lldb) break set --fullname main
Breakpoint 2: 2 locations.
(lldb)
Breakpoint 1: 19 locations.

1: name = 'main', locations = 19
   1.1: where = Sketch`main + 55 at SKTMain.m:17
   1.2: where = Foundation`-[NSThread main]
   1.3: where = Foundation`-[NSBlockOperation main]

...
(lldb) break set --name main
Breakpoint 1: 19 locations.
(lldb) break list 1
1: name = 'main', locations = 19
   1.1: where = Sketch`main + 55 at SKTMain.m:17
   1.2: where = Foundation`-[NSThread main]
   1.3: where = Foundation`-[NSBlockOperation main]
...
(lldb) break set --fullname main
Breakpoint 2: 2 locations.
(lldb) break list 2
2: name = 'main', locations = 2
   2.1: where = Sketch`main + 55 at SKTMain.m:17
   2.2: where = libpcap.A.dylib`main

Two shared libraries with the same symbol
(lldb) break set --name main
Breakpoint 1: 19 locations.
(lldb) break list 1
1: name = 'main', locations = 19
  1.1: where = Sketch`main + 55 at SKTMain.m:17
  1.2: where = Foundation`-[NSThread main]
  1.3: where = Foundation`-[NSBlockOperation main]
...
(lldb) break set --fullname main
Breakpoint 2: 2 locations.
(lldb) break list 2
2: name = 'main', locations = 2
  2.1: where = Sketch`main + 55 at SKTMain.m:17
  2.2: where = libpcap.A.dylib`main
(lldb)
(lldb) break set --name main
Breakpoint 1: 19 locations.

(lldb) break list 1
1: name = 'main', locations = 19
   1.1: where = Sketch`main + 55 at SKTMain.m:17
   1.2: where = Foundation`-[NSThread main]
   1.3: where = Foundation`-[NSBlockOperation main]

...

(lldb) break set --fullname main
Breakpoint 2: 2 locations.

(lldb) break list 2
2: name = 'main', locations = 2
   2.1: where = Sketch`main + 55 at SKTMain.m:17
   2.2: where = libpcap.A.dylib`main

(lldb) break set --fullname main --shlib Sketch
Breakpoint 3: where = Sketch`main + 55 at SKTMain.m:17, address = 0x000000100018fe7
Multiple Locations

When and why?
Multiple Locations

When and why?

Example with file and line breakpoint locations:
func callIt () {
    my_object.useClosure() {() -> Void in
        print ("Main's closure did something.")
    }
}
func callIt ()
{
    my_object.useClosure() {() -> Void in
        print ("Main's closure did something.")
    }
}
```swift
    func callIt () {
        my_object.useClosure() {() -> Void in
            print ("Main's closure did something.")
        }
    }
```
func callIt ()
{
    my_object.useClosure() {() -> Void in
        print ("Main's closure did something.")
    }
}
```swift
func callIt () {
    my_object.useClosure() {() -> Void in
        print ("Main's closure did something.")
    }
}
```
func callIt () {
    my_object.useClosure() {() -> Void in
        print ("Main's closure did something.")
    }

break set --line 12 --file Example.swift

Breakpoint 1: 2 locations.

break list 1

1: file = '/tmp/Example.swift', line = 12, exact_match = 0, locations = 2
   1.1: where = Example`Example.callIt () -> () + 25 at Example.swift:12, ...
   1.2: where = Example`Example.(callIt () -> ()).(closure #1) + 15 at Example.swift:13, ...
func callIt () {
    my_object.useClosure() {() -> Void in
        print("Main's closure did something.")
    }
}
func callIt () {
    my_object.useClosure() {() -> Void in
        print ("Main's closure did something.")
    }
}

break set --line 12 --file Example.swift
Breakpoint 1: 2 locations.

break list 1
1: file = '/tmp/Example.swift', line = 12, exact_match = 0, locations = 2
    1.1: where = Example`Example.callIt () -> () + 25 at Example.swift:12, ...
    1.2: where = Example`Example.(callIt () -> ()).(closure #1) + 15 at Example.swift:13, ...
Breakpoint Set Command
Breakpoint Set Command

**breakpoint set** command form:

```lldb```
Breakpoint Set Command

**breakpoint set** command form:

```
(lldb) break set --<Type> <Value> --<OtherOptions>
```
Breakpoint Set Command

**breakpoint set** command form:

```
(lldb) break set --<Type> <Value> --<OtherOptions>
```

Type option:
Breakpoint Set Command

**breakpoint set** command form:

```plaintext
(lldb) breakpoint set --<Type> <Value> --<OtherOptions>
```

Type option:

- Sets the kind of search you are doing (file and line, symbol name, etc.)
**Breakpoint Set Command**

**breakpoint set** command form:

```lldb
(lldb) break set --<Type> <Value> --<OtherOptions>
```

**Type option:**

- Sets the kind of search you are doing (file and line, symbol name, etc.)
- Value is the data for the search
Breakpoint Set Command

**breakpoint set** command form:

```
(lldb) break set --<Type> <Value> --<OtherOptions>
```

Type option:
- Sets the kind of search you are doing (file and line, symbol name, etc.)
- Value is the data for the search

Other options:
Breakpoint Set Command

`breakpoint set` command form:

```
(lldb) breakpoint set --<Type> <Value> --<OtherOptions>
```

Type option:
- Sets the kind of search you are doing (file and line, symbol name, etc.)
- Value is the data for the search

Other options:
- Ignore count, condition, and so on
Breakpoint Set Command

**breakpoint set** command form:

```
(lldb) break set --<Type> <Value> --<OtherOptions>
```

Type option:

- Sets the kind of search you are doing (file and line, symbol name, etc.)
- Value is the data for the search

Other options:

- Ignore count, condition, and so on
- Specify whether to break, not where...
Breakpoint Set Command

**breakpoint set** command form:

```
(lldb) breakpoint set --<Type> <Value> --<OtherOptions>
```

**Type option:**
- Sets the kind of search you are doing (file and line, symbol name, etc.)
- Value is the data for the search

**Other options:**
- Ignore count, condition, and so on
- Specify whether to break, not where…
- Can be modified after the fact
Breakpoint Locations

Where to stop
Breakpoint Locations

Where to stop

Each breakpoint location is a single search result
Breakpoint Locations
Where to stop

Each breakpoint location is a single search result
• Unique address where program execution may halt
Breakpoint Locations
Where to stop

Each breakpoint location is a single search result
- Unique address where program execution may halt
- Specified by breakpoint and location numbers:
Breakpoint Locations

Where to stop

Each breakpoint location is a single search result

- Unique address where program execution may halt
- Specified by breakpoint and location numbers:
  - Written separated by a dot
Breakpoint Locations
Where to stop

Each breakpoint location is a single search result
• Unique address where program execution may halt
• Specified by breakpoint and location numbers:
  - Written separated by a dot
  - 1.1, 2.2...
Options for Breakpoints and Locations
Options for Breakpoints and Locations

Breakpoints and locations take the same generic options
Options for Breakpoints and Locations

Breakpoints and locations take the same generic options

• Conditions, commands, and so on
Options for Breakpoints and Locations

Breakpoints and locations take the same generic options

- Conditions, commands, and so on

Location options override breakpoint options
Options for Breakpoints and Locations

Breakpoints and locations take the same generic options

- Conditions, commands, and so on

Location options override breakpoint options

Disabling the breakpoint deactivates all locations
Options for Breakpoints and Locations

Breakpoints and locations take the same generic options
• Conditions, commands, and so on

Location options override breakpoint options
Disabling the breakpoint deactivates all locations
• Locations can be disabled individually
Options for Breakpoints and Locations

Breakpoints and locations take the same generic options

• Conditions, commands, and so on

Location options override breakpoint options

Disabling the breakpoint deactivates all locations

• Locations can be disabled individually

• Disabled locations stay disabled when disabling/enabling breakpoint
More Powerful Breakpoint Types
More Powerful Breakpoint Types

How to search for places to stop?
More Powerful Breakpoint Types

How to search for places to stop?
LLDB offers two spaces to search:
More Powerful Breakpoint Types

How to search for places to stop?
LLDB offers two spaces to search:

• Both use *regular expressions* to express search patterns
More Powerful Breakpoint Types

How to search for places to stop?
LLDB offers two spaces to search:

• Both use *regular expressions* to express search patterns
• Function name searches:
More Powerful Breakpoint Types

How to search for places to stop?
LLDB offers two spaces to search:

• Both use regular expressions to express search patterns
• Function name searches:
  ```bash
  --func-regex (or -r)
  ```
More Powerful Breakpoint Types

How to search for places to stop?
LLDB offers two spaces to search:

- Both use *regular expressions* to express search patterns
- Function name searches:
  - `--func-regex` (or `-r`)
- Source text searches:
More Powerful Breakpoint Types

How to search for places to stop?
LLDB offers two spaces to search:

• Both use *regular expressions* to express search patterns
• Function name searches:
  `--func-regex` (or `-r`)
• Source text searches:
  `--source-pattern-regexp` (or `-p`)
Pattern Matching for Function Names
Pattern Matching for Function Names

Example problems:
Pattern Matching for Function Names

Example problems:

- Stop on all methods implemented by a class
Pattern Matching for Function Names

Example problems:

• Stop on all methods implemented by a class
• But not parent or subclasses
Pattern Matching for Function Names

Example problems:

- Stop on all methods implemented by a class
- But not parent or subclasses
  - Swift:

(lldb)
Pattern Matching for Function Names

Example problems:
- Stop on all methods implemented by a class
- But not parent or subclasses
  - Swift:

```sh
(lldb) break set -r "\.ClassName\..*
```
Pattern Matching for Function Names

Example problems:

• Stop on all methods implemented by a class
• But not parent or subclasses
  - Swift:
    ```lldb
    (lldb) break set -r "\.ClassName\..*"
    ```
  - Objective-C:
    ```lldb
    ```
Pattern Matching for Function Names

Example problems:

- Stop on all methods implemented by a class
- But not parent or subclasses
  - Swift:
    
    ```bash
    (lldb) break set -r "\.ClassName\..*"
    ```
  - Objective-C:
    
    ```bash
    (lldb) break set -r "\[ClassName \.*\]"
    ```
Pattern Matching for Function Names
Pattern Matching for Function Names

Example problems:
Pattern Matching for Function Names

Example problems:

- Stop on all functions in a given module:
Example problems:

- Stop on all functions in a given module:

```bash
(lldb) break set -r ".*" --shlib MyModule
```
Pattern Matching for Function Names

Example problems:

- Stop on all functions in a given module:

  (lldb) break set -r ".*" --shlib MyModule

- Use with breakpoint commands to trace execution
Pattern Matching for Function Names

Example problems:

• Stop on all functions in a given module:

(lldb) break set -r ".*" --shlib MyModule

- Use with breakpoint commands to trace execution
- This will slow down execution
Pattern Matching for Function Names

Example problems:
• Stop on all functions in a given module:

```
(lldb) break set -r ".*" --shlib MyModule
```

- Use with breakpoint commands to trace execution
- This will slow down execution
- Disable locations as you hit them
Pattern Matching in Source
Pattern Matching in Source

Some constructs are obvious in source
Pattern Matching in Source

Some constructs are obvious in source
But hard to identify in generated code
Pattern Matching in Source

Some constructs are obvious in source
But hard to identify in generated code
• Use of MACROS
Pattern Matching in Source

Some constructs are obvious in source
But hard to identify in generated code

• Use of MACROS
• Very specific usages:
Some constructs are obvious in source
But hard to identify in generated code

• Use of MACROS

• Very specific usages:
  - Places where you get a particular field from a pointer:
Pattern Matching in Source

Some constructs are obvious in source
But hard to identify in generated code

• Use of MACROS
• Very specific usages:
  - Places where you get a particular field from a pointer:
    
    \rightarrow\text{someField}
Pattern Matching in Source

Some constructs are obvious in source
But hard to identify in generated code

• Use of MACROS
• Very specific usages:
  - Places where you get a particular field from a pointer:
    \[ \text{\texttt{\rightarrow someField}} \]

You can also use it to make your own markers:
Pattern Matching in Source

Some constructs are obvious in source

But hard to identify in generated code

• Use of MACROS

• Very specific usages:
  - Places where you get a particular field from a pointer:

    ->someField

You can also use it to make your own markers:

    // Break here
Pattern Matching in Source

Command format:
Pattern Matching in Source

Command format:

(lldb)
Pattern Matching in Source

Command format:

(lldb) break set --source-regexp "// Break here" -f main.swift
Pattern Matching in Source

Command format:

```sh
(lldb) break set --source-regexp "// Break here" -f main.swift
```

Patterns in code can mark useful spots to stop
Pattern Matching in Source

Command format:

```
(lldb) break set --source-regexp "// Break here" -f main.swift
```

- `-f` specifies files to search for matches
Pattern Matching in Source
Pattern Matching in Source

Example problem:
Pattern Matching in Source

Example problem:

- In a complex function that can return from many places
Pattern Matching in Source

Example problem:

- In a complex function that can return from many places
- Stop whenever it returns null
Example problem:

- In a complex function that can return from many places
- Stop whenever it returns null

`--source-regexp-function` (or `-X`) limits search to a function
Pattern Matching in Source

Example problem:

• In a complex function that can return from many places
• Stop whenever it returns null

--source-regexp-function (or -X) limits search to a function

(llldb) break set -p "return *nullptr" -X Foo::StateMachine -f Foo.cpp
Additional Breakpoint Options
Additional Breakpoint Options

Specify the language for a breakpoint
Additional Breakpoint Options

Specify the language for a breakpoint

• Use the `--language` (or `-L`) option to `break set`
Additional Breakpoint Options

Specify the language for a breakpoint

• Use the `--language` (or `-L`) option to `break set`

• Useful in mixed Swift/Objective-C projects
Additional Breakpoint Options
Additional Breakpoint Options

Restricting a breakpoint to a specific thread:
Additional Breakpoint Options

Restricting a breakpoint to a specific thread:

• By thread id:
Restricting a breakpoint to a specific thread:

- By thread id:
  
  --thread-id (or -t)
Restricting a breakpoint to a specific thread:

• By thread id:

  --thread-id (or -t)

• By name for threads named by `pthread_setname_np()`:
Additional Breakpoint Options

Restricting a breakpoint to a specific thread:

- By thread id:
  
  ```
  --thread-id (or -t)
  ```

- By name for threads named by `pthread_setname_np()`:
  
  ```
  --thread-name (or -T)
  ```
Additional Breakpoint Options

Restricting a breakpoint to a specific thread:

- By thread id:
  
  \[ \text{--thread-id} \text{ (or -t)} \]

- By name for threads named by \texttt{pthread_setname_np()}
  
  \[ \text{--thread-name} \text{ (or -T)} \]

- To threads servicing a particular named queue:
Additional Breakpoint Options

Restricting a breakpoint to a specific thread:

• By thread id:
  
  --thread-id (or -t)

• By name for threads named by `pthread_setname_np()`:
  
  --thread-name (or -T)

• To threads servicing a particular named queue:
  
  --queue-name (or -q)
Applying Options to Existing Breakpoints
Applying Options to Existing Breakpoints

Options can be set/modified on extant breakpoints
Applying Options to Existing Breakpoints

Options can be set/modified on extant breakpoints
Can modify Xcode breakpoints as well
Applying Options to Existing Breakpoints

Options can be set/modified on extant breakpoints
Can modify Xcode breakpoints as well
Command is `break modify`

```
(lldb) break modify -T ImportantThreads 1 2.1 4.1-4.5 7-10
```
Applying Options to Existing Breakpoints

Options can be set/modified on extant breakpoints
Can modify Xcode breakpoints as well
Command is `break modify`
- Specify breakpoints, breakpoint locations, or ranges of either

(llldb) `break modify -T ImportantThreads 1 2.1 4.1–4.5 7–10`
Applying Options to Existing Breakpoints

Options can be set/modified on extant breakpoints
Can modify Xcode breakpoints as well
Command is `break modify`

• Specify breakpoints, breakpoint locations, or ranges of either

```(lldb) break modify -T ImportantThreads 1 2.1 4.1-4.5 7-10```
Applying Options to Existing Breakpoints

Options can be set/modified on extant breakpoints
Can modify Xcode breakpoints as well
Command is `break modify`
• Specify breakpoints, breakpoint locations, or ranges of either

```
(lldb) break modify -T ImportantThreads 1 2.1 4.1-4.5 7-10
```
Applying Options to Existing Breakpoints

Options can be set/modified on extant breakpoints
Can modify Xcode breakpoints as well
Command is **`break modify`**
  • Specify breakpoints, breakpoint locations, or ranges of either

```
(lldb) break modify -T ImportantThreads 1 2.1 4.1-4.5 7-10
```
Applying Options to Existing Breakpoints

Options can be set/modified on extant breakpoints
Can modify Xcode breakpoints as well
Command is **break modify**

- Specify breakpoints, breakpoint locations, or ranges of either
- Defaults to last set breakpoint when none specified

```
(llldb) break modify -T ImportantThreads 1 2.1 4.1-4.5 7-10
```
Storing Complex Breakpoints
Storing Complex Breakpoints

Xcode only persists breakpoints set through the UI
Storing Complex Breakpoints

Xcode only persists breakpoints set through the UI

• For breakpoints in *all* projects, set in ~/.lldbinit
No Breakpoints
Storing Breakpoints in a Project

Make a breakpoint Xcode will store
Storing Breakpoints in a Project

Make a breakpoint Xcode will store

Something hit early on
Storing Breakpoints in a Project

Make a breakpoint Xcode will store
Something hit early on
Add your breakpoints as commands
Storing Breakpoints in a Project

Make a breakpoint Xcode will store
Something hit early on
Add your breakpoints as commands
Storing Breakpoints in a Project

Make a breakpoint Xcode will store
Something hit early on
Add your breakpoints as commands
Storing Breakpoints in a Project

Make a breakpoint Xcode will store
Something hit early on
Add your breakpoints as commands
Storing Breakpoints in a Project

Make a breakpoint Xcode will store

Something hit early on

Add your breakpoints as commands
Storing Breakpoints in a Project

Make a breakpoint Xcode will store
Something hit early on
Add your breakpoints as commands
Storing Breakpoints in a Project

Make a breakpoint Xcode will store
Something hit early on
Add your breakpoints as commands
Storing Breakpoints in a Project

Make a breakpoint Xcode will store

Something hit early on

Add your breakpoints as commands
Storing Breakpoints in a Project

Make a breakpoint Xcode will store something hit early on. Add your breakpoints as commands.
Storing Breakpoints in a Project

Make a breakpoint Xcode will store
Something hit early on
Add your breakpoints as commands
Storing Breakpoints in a Project

Make a breakpoint Xcode will store

Something hit early on

Add your breakpoints as commands
Storing Breakpoints in a Project

Make a breakpoint Xcode will store
Something hit early on
Add your breakpoints as commands
Storing Breakpoints in a Project

Make a breakpoint Xcode will store something hit early on.
Add your breakpoints as commands.
Storing Breakpoints in a Project

Make a breakpoint Xcode will store
Something hit early on
Add your breakpoints as commands
Storing Breakpoints in a Project

Make a breakpoint Xcode will store
Something hit early on
Add your breakpoints as commands
Storing Breakpoints in a Project

Make a breakpoint Xcode will store

Something hit early on

Add your breakpoints as commands
Stepping
Targeted Stepping in Complex Situations
Targeted Stepping in Complex Situations

In modern languages, many simple expressions are actually function calls
Targeted Stepping in Complex Situations

In modern languages, many simple expressions are actually function calls
Often not interesting to step through...
Targeted Stepping in Complex Situations

In modern languages, many simple expressions are actually function calls.

Often not interesting to step through...

This is a common scenario:
Process 4971 stopped
* thread #1: tid = 0x1c8535, function: stepping.main () -> (), stop reason = breakpoint 1.1
frame #0: 0x000000010001165 stepping\`stepping.main () -> () at stepping.swift:38

35   main () -> Void
36   {
37       let my_cp = ComputedProperties()
38       doSomething(my_cp.computed_ivar_1,
39           my_cp.computed_ivar_2,
40           my_cp.computed_ivar_3)
41   }

(lldb)
Process 4971 stopped
* thread #1: tid = 0x1c8535, function: stepping.main () -> (), stop reason = breakpoint 1.1
  frame #0: 0x0000000100001165 stepping`stepping.main () -> () at stepping.swift:38
  35   main () -> Void
  36   {
  37       let my_cp = ComputedProperties()
  -> 38       doSomething(my_cp.computed_ivar_1,
  39       my_cp.computed_ivar_2,
  40       my_cp.computed_ivar_3)
  41   }
(lldb)

I want to stop in doSomething
Process 4971 stopped
* thread #1: tid = 0x1c8535, function: stepping.main () -> (), stop reason = breakpoint 1.1
  frame #0: 0x00000000100001165 stepping`stepping.main () -> () at stepping.swift:38
 35   main () -> Void
 36   {
 37       let my_cp = ComputedProperties()
 38       doSomething(my_cp.computed_ivar_1,
 39           my_cp.computed_ivar_2,
 40           my_cp.computed_ivar_3)
 41   }
(lldb) step
Process 5108 stopped
* thread #1: tid = 0xaa0a3, function: stepping.ComputedProperties.computed_ivar_1.getter : Swift.Int, stop reason = step in
  frame #0: 0x000000001000010fd stepping`stepping.ComputedProperties.computed_ivar_1.getter : Swift.Int at stepping.swift:5
 2     {
 3        var computed_ivar_1 : Int {
 4            get {
 5                return 10
 6            }
 7        }
 8     var computed_ivar_2 : Int {
(lldb)
Process 4971 stopped
* thread #1: tid = 0x1c8535, function: stepping.main () -> (), stop reason = breakpoint 1.1
  frame #0: 0x0000000100001165 stepping`stepping.main () -> () at stepping.swift:38
 35   main () -> Void
 36   {
 37       let my_cp = ComputedProperties()
-> 38       doSomething(my_cp.computed_ivar_1,
 39           my_cp.computed_ivar_2,
 40           my_cp.computed_ivar_3)
 41   }
(lldb)

Process 5108 stopped
* thread #1: tid = 0xaa0a3, function: stepping.ComputedProperties.computed_ivar_1.getter : Swift.Int , stop reason = step in
  frame #0: 0x00000001000010fd stepping`stepping.ComputedProperties.computed_ivar_1.getter : Swift.Int at stepping.swift:5
  2   {
->  3       var computed_ivar_1 : Int {
  4           get {
  5               return 10
  6           }
  7       }
  8       var computed_ivar_2 : Int {
(lldb)

Instead I stopped in an accessor
Process 4971 stopped
* thread #1: tid = 0x1c8535, function: stepping.main () -> (), stop reason = breakpoint 1.1
  frame #0: 0x0000000100001165 stepping`stepping.main () -> () at stepping.swift:38
  35   main () -> Void
  36   {
  37       let my_cp = ComputedProperties()
->  38       doSomething(my_cp.computed_ivar_1,
  39             my_cp.computed_ivar_2,
  40             my_cp.computed_ivar_3)
  41   }
(lldb)
Process 5108 stopped
* thread #1: tid = 0xaa0a3, function: stepping.ComputedProperties.computed_ivar_1.getter : Swift.Int, stop reason = step in
  frame #0: 0x00000001000010fd stepping`stepping.ComputedProperties.computed_ivar_1.getter : Swift.Int at stepping.swift:5
  2   {
->  3       var computed_ivar_1 : Int {
    4           get {
      5               return 10
    6           }
    7       }
  8   var computed_ivar_2 : Int {
(lldb)
Process 4971 stopped
* thread #1: tid = 0x1c8535, function: stepping.main () -> (), stop reason = breakpoint 1.1
  frame #0: 0x0000000100001165 stepping`stepping.main () -> () at stepping.swift:38
  35   main () -> Void
  36   {
  37       let my_cp = ComputedProperties()
  38       doSomething(my_cp.computed_ivar_1,
  39           my_cp.computed_ivar_2,
  40           my_cp.computed_ivar_3)
  41   }
(lldb) step
Process 5108 stopped
* thread #1: tid = 0xaa0a3, function: stepping.ComputedProperties.computed_ivar_1.getter : Swift.Int , stop reason = step in
  frame #0: 0x00000001000010fd stepping`stepping.ComputedProperties.computed_ivar_1.getter : Swift.Int at stepping.swift:5
  2   {
  3       var computed_ivar_1 : Int {
  4           get {
  5               return 10
  6           }
  7       }
  8       var computed_ivar_2 : Int {
(lldb) finish
...
(lldb)
Process 4971 stopped
* thread #1: tid = 0x1c8535, function: stepping.main () -> (), stop reason = breakpoint 1.1
  frame #0: 0x0000000100001165 stepping`stepping.main () -> () at stepping.swift:38
35   main () -> Void
36   {
37       let my_cp = ComputedProperties()
38       doSomething(my_cp.computed_ivar_1,
39                   my_cp.computed_ivar_2,
40                   my_cp.computed_ivar_3)
41   }
(lldb) step
Process 5108 stopped
* thread #1: tid = 0xaa0a3, function: stepping.ComputedProperties.computed_ivar_1.getter : Swift.Int, stop reason = step in
  frame #0: 0x00000001000010fd stepping`stepping.ComputedProperties.computed_ivar_1.getter : Swift.Int at stepping.swift:5
2   {  
3       var computed_ivar_1 : Int {
4           get {
5               return 10
6           }  
7       }  
8       var computed_ivar_2 : Int {
9      
(lldb) finish
...  
(lldb) step  
...
Targeted Stepping in Complex Situations
Targeted Stepping in Complex Situations

Step into `doSomething` without stopping in accessors?
Targeted Stepping in Complex Situations

Step into `doSomething` without stopping in accessors?
Use the step command’s `--step-in-target` option:

```
(lldb)
```
Targeted Stepping in Complex Situations

Step into `doSomething` without stopping in accessors?

Use the step command’s `--step-in-target` option:

```
(lldb) step --step-in-target doSomething
```
Targeted Stepping in Complex Situations

Step into `doSomething` without stopping in accessors?

Use the step command’s `--step-in-target` option:

```bash
(lldb) step --step-in-target doSomething
```

That *almost* works in this case:
Process 4971 stopped
* thread #1: tid = 0x1c8535, function: stepping.main () -> () , stop reason = breakpoint
frame #0: 0x0000000100001165 stepping`stepping.main () -> () at stepping.swift:38
  35   main () -> Void
  36   {
  37       let my_cp = ComputedProperties()
  -> 38       doSomething(my_cp.computed_ivar_1,
  39           my_cp.computed_ivar_2,
  40           my_cp.computed_ivar_3)
  41   }
(lldb)
Process 4971 stopped
* thread #1: tid = 0x1c8535, function: stepping.main () -> (), stop reason = breakpoint 1.1
  frame #0: 0x0000000100001165 stepping\stepping.main () -> () at stepping.swift:38
  35   main () -> Void
  36   {
  37       let my_cp = ComputedProperties()
  -> 38       doSomething(my_cp.computed_ivar_1,
  39                       my_cp.computed_ivar_2,
  40                       my_cp.computed_ivar_3)
  41   }
(lldb) step --step-in-target doSomething
Process 4971 stopped
* thread #1: tid = 0x1c8535, function: stepping.main () -> (), stop reason = step in
  frame #0: 0x000000010000117a stepping\stepping.main () -> () at stepping.swift:39
  36   {
  37       let my_cp = ComputedProperties()
  38       doSomething(my_cp.computed_ivar_1,
  -> 39                       my_cp.computed_ivar_2,
  40                       my_cp.computed_ivar_3)
  41   }
  42
Well, at least I didn't end up in the accessor
Targeted Stepping in Complex Situations
Targeted Stepping in Complex Situations

Stepping is by source line
Targeted Stepping in Complex Situations

Stepping is by source line
This call spans multiple lines...
Targeted Stepping in Complex Situations

Stepping is by source line
This call spans multiple lines...
Specify the end line number:

(lldb)
Targeted Stepping in Complex Situations

Stepping is by source line

This call spans multiple lines...

Specify the end line number:

```
(lldb) step -t doSomething --end-linenumber 40
```
Targeted Stepping in Complex Situations

Stepping is by source line

This call spans multiple lines...

Specify the end line number:

(lldb) step -t doSomething --end-linenuumber 40

Easier: use the special token block
Targeted Stepping in Complex Situations

Stepping is by source line
This call spans multiple lines...
Specify the end line number:

(lldb) step -t doSomething --end-linenumber 40

Easier: use the special token block
• Step with a safeguard around the current semantic block
Targeted Stepping in Complex Situations

Stepping is by source line
This call spans multiple lines...
Specify the end line number:

```lldb
step -t doSomething --end-linenumber 40
```

Easier: use the special token `block`

- Step with a safeguard around the current semantic block

There's even an alias for this:
Targeted Stepping in Complex Situations

Stepping is by source line
This call spans multiple lines...
Specify the end line number:

```
(lldb) step -t doSomething --end-linenumber 40
```

Easier: use the special token `block`
• Step with a safeguard around the current semantic block

There's even an alias for this:
```
sif
```
stands for step into function
main () -> Void
{
    let my_cp = ComputedProperties()
    doSomething(my_cp.computed_ivar_1, my_cp.computed_ivar_2, my_cp.computed_ivar_3)
Process 4971 stopped
* thread #1: tid = 0x1c8535, function: stepping.main () -> (), stop reason = breakpoint 1.1
    frame #0: 0x0000000100001165 stepping`stepping.main () -> () at stepping.swift:38
        35   main () -> Void
        36   {
        37       let my_cp = ComputedProperties()
        -> 38       doSomething(my_cp.computed_ivar_1,
        39           my_cp.computed_ivar_2,
        40           my_cp.computed_ivar_3)
        41   }
(lldb) sif doSomething
Process 4971 stopped
* thread #1: tid = 0x1c8535, function: stepping.doSomething (Swift.Int, Swift.Int, Swift.Int) -> Swift.Int, stop reason = step in
    frame #0: 0x00000001000011c0 stepping`stepping.doSomething (Swift.Int, Swift.Int, Swift.Int) -> Swift.Int at stepping.swift:31
        28   func
        29   doSomething(_ one : Int, _ two: Int, _ three: Int) -> Int
        30   {
        -> 31       return one + two + three
        32   }
Troubleshooting
What Binaries Were Loaded?
What Binaries Were Loaded?

Sometimes you need to see exactly what binaries you are running
What Binaries Were Loaded?

Sometimes you need to see exactly what binaries you are running

- I have built Release and Debug; which am I using now?
What Binaries Were Loaded?

Sometimes you need to see exactly what binaries you are running

- I have built Release and Debug; which am I using now?
- I have a dSYM, is it getting read in?
What Binaries Were Loaded?

Sometimes you need to see exactly what binaries you are running

- I have built Release and Debug; which am I using now?
- I have a dSYM, is it getting read in?

The command to query the binaries in your program is:

```
(lldb)
```
What Binaries Were Loaded?

Sometimes you need to see exactly what binaries you are running

• I have built Release and Debug; which am I using now?
• I have a dSYM, is it getting read in?

The command to query the binaries in your program is:

(lldb) image list [<ModuleName>]

What Binaries Were Loaded?

Sometimes you need to see exactly what binaries you are running

- I have built Release and Debug; which am I using now?
- I have a dSYM, is it getting read in?

The command to query the binaries in your program is:

- With no arguments, lists all binaries

(lldb) image list [<ModuleName>]


(lldb) image list Example
[  0] C9F4C7B9-7A81-3428-A1D3-A454B3A3C472 0x0000000100000000 /private/tmp/Example/build/Debug/Example.app/Contents/MacOS/Example
   /private/tmp/Example/build/Debug/Example.app.dSYM/Contents/Resources/DWARF/Example
image list Example

[ 0] 0x000000100000000 /private/tmp/Example/build/Debug/Example.app/Contents/MacOS/Example
/private/tmp/Example/build/Debug/Example.app.dSYM/Contents/Resources/DWARF/Example

This is the path to the binary
(lldb) image list Example
[ 0] C9F4C7B9-7A81-3428-A1D3-A454B3A3C472 0x0000000100000000 /private/tmp/Example/build/Debug/Example.app/Contents/MacOS/Example
/private/tmp/Example/build/Debug/Example.app.dSYM/Contents/Resources/DWARF/Example
(lldb) image list Example
[ 0] C9F4C7B9-7A81-3428-A1D3-A454B3A3C472 0x0000000100000000 /private/tmp/Example/build/Debug/
Example.app/Contents/MacOS/Example
/private/tmp/Example/build/Debug/Example.app.dSYM/Contents/Resources/DWARF/Example

It is the debug build we've loaded!
(lldb) image list Example
[ 0] C9F4C7B9-7A81-3428-A1D3-A454B3A3C472 0x0000000100000000 /private/tmp/Example/build/Debug/
Example.app/Contents/MacOS/Example
 /private/tmp/Example/build/Debug/Example.app.dSYM/Contents/Resources/DWARF/Example
(lldb) image list Example
[ 0] C9F4C7B9-7A81-3428-A1D3-A454B3A3C472 0x0000000010000000 /private/tmp/Example/build/Debug/Example.app/Contents/MacOS/Example
/private/tmp/Example/build/Debug/Example.app.dSYM/Contents/Resources/DWARF/Example

And here is the dSYM
Swift Debug Information
In Swift, the debugger reads type information directly from the Swift module.
Swift Debug Information

In Swift, the debugger reads type information directly from the Swift module

• Ensures greater fidelity – good!
Swift Debug Information

In Swift, the debugger reads type information directly from the Swift module

- Ensures greater fidelity – good!
- Ties the debugger to the compiler that built the module
Swift Debug Information

In Swift, the debugger reads type information directly from the Swift module

- Ensures greater fidelity – good!
- Ties the debugger to the compiler that built the module

The binding between Objective-C modules and Swift is required by the debugger
Swift Debug Information

In Swift, the debugger reads type information directly from the Swift module

- Ensures greater fidelity – good!
- Ties the debugger to the compiler that built the module

The binding between Objective-C modules and Swift is required by the debugger

- LLDB has to reconstruct the Objective-C modules as originally built
Swift Debug Information

In Swift, the debugger reads type information directly from the Swift module

- Ensures greater fidelity – good!
- Ties the debugger to the compiler that built the module

The binding between Objective-C modules and Swift is required by the debugger

- LLDB has to reconstruct the Objective-C modules as originally built

TL;DR?
Swift Debug Information

In Swift, the debugger reads type information directly from the Swift module

• Ensures greater fidelity – good!
• Ties the debugger to the compiler that built the module

The binding between Objective-C modules and Swift is required by the debugger

• LLDB has to reconstruct the Objective-C modules as originally built

TL;DR?

• All Swift code with debug info needs to have been built locally
Optimized Code Debugging
Optimized Code Debugging

Enrico's Rule of Optimized Code Debugging:
Optimized Code Debugging

Enrico's Rule of Optimized Code Debugging:

• Don't do it if you don't have to
Optimized Code Debugging

Enrico's Rule of Optimized Code Debugging:

• Don't do it if you don't have to

Corollary to Enrico's Rule of Optimized Code Debugging:
Optimized Code Debugging

Enrico's Rule of Optimized Code Debugging:

• Don't do it if you don't have to

Corollary to Enrico's Rule of Optimized Code Debugging:

• Most people who do it do it by accident
Optimized Code Debugging

Enrico's Rule of Optimized Code Debugging:
• Don't do it if you don't have to

Corollary to Enrico's Rule of Optimized Code Debugging:
• Most people who do it do it by accident
• LLDB will tell you if a .o file was compiled with optimization
Optimized Code Debugging

Enrico's Rule of Optimized Code Debugging:

- Don't do it if you don't have to

Corollary to Enrico's Rule of Optimized Code Debugging:

- Most people who do it do it by accident
- LLDB will tell you if a .o file was compiled with optimization
- When you stop in it
Optimized Code Debugging

Enrico's Rule of Optimized Code Debugging:
• Don't do it if you don't have to

Corollary to Enrico's Rule of Optimized Code Debugging:
• Most people who do it do it by accident
• LLDB will tell you if a .o file was compiled with optimization
• When you stop in it
• Only once per binary with optimization:
Optimized Code Debugging

Enrico's Rule of Optimized Code Debugging:

• Don't do it if you don't have to

Corollary to Enrico's Rule of Optimized Code Debugging:

• Most people who do it do it by accident
• LLDB will tell you if a .o file was compiled with optimization
• When you stop in it
• Only once per binary with optimization:

Func was compiled with optimization – stepping may behave oddly; variables may not be available.
Clang Module Debug Information
Clang Module Debug Information

Allows compiler to reuse module type repositories for debug information
Clang Module Debug Information

Allows compiler to reuse module type repositories for debug information

• Can also use PCH files
Clang Module Debug Information

Allows compiler to reuse module type repositories for debug information

• Can also use PCH files
• Called *Clang Module Debugging* in Xcode Build Settings
Clang Module Debug Information

Allows compiler to reuse module type repositories for debug information

• Can also use PCH files

• Called Clang Module Debugging in Xcode Build Settings

• Compiler flag `--gmodules`
Clang Module Debug Information

Allows compiler to reuse module type repositories for debug information

- Can also use PCH files
- Called *Clang Module Debugging* in Xcode Build Settings
- Compiler flag `-gmodules`
- Can speed up compile times
Clang Module Debug Information

Caveats, provisos, and quid pro quos:
Clang Module Debug Information

Caveats, provisos, and quid pro quos:

- Debug information depends on the module cache or PCH files
Caveats, provisos, and quid pro quos:

- Debug information depends on the module cache or PCH files
  - Not part of your app or framework
Clang Module Debug Information

Caveats, provisos, and quid pro quos:

- Debug information depends on the module cache or PCH files
  - Not part of your app or framework
  - \texttt{dsymutil} will join all the parts into the dSYM
Caveats, provisos, and quid pro quos:

- Debug information depends on the module cache or PCH files
  - Not part of your app or framework
  - `dSYMutil` will join all the parts into the dSYM
  - Can't use it when shipping static archives
Clang Module Debug Information

Caveats, provisos, and quid pro quos:

- Debug information depends on the module cache or PCH files
  - Not part of your app or framework
  - `dSYMutil` will join all the parts into the dSYM
  - Can't use it when shipping static archives
- Deleted the module cache? Rebuild before debugging
Summary
Summary

LLDB is extremely customizable
Summary

LLDB is extremely customizable

*Many* ways to look at data
Summary

LLDB is extremely customizable

Many ways to look at data

Expressions are flexible, more than just data inspection
Summary

LLDB is extremely customizable

Many ways to look at data

Expressions are flexible, more than just data inspection

Beyond the gutter: breakpoints rock
Summary

LLDB is extremely customizable

*Many ways to look at data*

Expressions are flexible, more than just data inspection

Beyond the gutter: breakpoints rock

*More than source-level debugging*
Summary

LLDB is extremely customizable

*Many ways to look at data*

Expressions are flexible, more than just data inspection

Beyond the gutter: breakpoints rock

More than source-level debugging

• Rich tools for exploring running code
## Related Sessions

<table>
<thead>
<tr>
<th>Session</th>
<th>Location</th>
<th>Date/Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visual Debugging with Xcode</td>
<td>Presidio</td>
<td>Wednesday 4:00PM</td>
</tr>
<tr>
<td>Thread Sanitizer and Static Analysis</td>
<td>Mission</td>
<td>Thursday 10:00AM</td>
</tr>
<tr>
<td>Debugging with LLDB</td>
<td></td>
<td>WWDC 2012</td>
</tr>
<tr>
<td>Advanced Debugging with LLDB</td>
<td></td>
<td>WWDC 2013</td>
</tr>
</tbody>
</table>