Managing Breakpoints

As described earlier, a *breakpoint* interrupts execution of a program at a specified point in execution.

Breakpoints are the primary means by which a developer uses the debugger to start interacting with a program.

Setting a Breakpoint

You use the `breakpoint set` command to set a breakpoint, specifying either a function name, by passing the `--name` (`-n`) option, or a source file and line number by passing the `--file` (`-f`) and `--line` (`-l`) options. In the following example, breakpoints are set on any function named `sayHello` and in the `main.swift` file at line `5`:

```
(lldb) breakpoint set -n sayHello

Breakpoint 1: where = main`main.sayHello () -> () + 15 at main.swift:2, address =
0x00000001000014cf

(lldb) breakpoint set -f main.swift -l 5

Breakpoint 2: where = main`main + 70 at main.swift:5, address = 0x00000001000014a6
```

You can also set a breakpoint to occur anytime a Swift error is thrown or an Objective-C exception is raised by passing the `--language-exception` (`-E`) option with `Swift` or `objc` as the value. To additionally configure the breakpoint to stop only when a particular type of error is thrown or an exception is raised, pass the `--exception-typename` (`-O`) option with the name of the type as the value.

```
(lldb) breakpoint set -E Swift -O EnumErrorType
```

```
(lldb) breakpoint set -E objc
```

Listing Breakpoints

You use the `breakpoint list` command to show all breakpoints that have been set.

```
(lldb) breakpoint list

Current breakpoints:

1: name = 'sayHello', locations = 1

  1.1: where = main`main.sayHello () -> () + 15 at main.swift:2, address =
main[0x00000001000011cf], unresolved, hit count = 0


2: file = 'main.swift', line = 4, exact_match = 0, locations = 1

  2.1: where = main`main + 70 at main.swift:7, address = main[0x00000001000011a6],
unresolved, hit count = 0
```

Setting a breakpoint creates a *logical breakpoint*, which may resolve to one or more *locations*.

Each logical breakpoint has a sequential assigned integer ID, starting from `1`. Each location is identified by the ID of its logical breakpoint, followed by a dot (`.`), and then its own sequential

assigned integer ID starting from `1`. For example, the second location of the first set breakpoint has the ID `1.2`.

Logical breakpoints are *live*, meaning that if any new code is loaded into the program, new locations are automatically created. If you set a breakpoint that doesn't resolve to any locations in the program, a *pending* breakpoint is created. A pending breakpoint may indicate a typo in the breakpoint specification or that a file or shared library isn't loaded.

```
>> (lldb) breakpoint set --file main.swift --line 12
Breakpoint created: 2: file ='main.swift', line = 12, locations = 0 (pending)
WARNING: Unable to resolve breakpoint to any actual locations.
```

Modifying a Breakpoint

You use the `breakpoint modify` command to modify a logical breakpoint or individual location by passing the breakpoint ID or location ID as an argument, and any of the following configuration options:

- `--condition` (`-c`) Specifies an expression that must evaluate to true in order for the breakpoint to stop
- `--ignore-count` (`-i`) Specifies the number of times the breakpoint is skipped before stopping
- `--one-shot` (`-o`) Removes the breakpoint the first time it stops
- `--queue-name` (`-q`) Specifies the name of the queue on which the breakpoint stops
- `--thread-name` (`-T`) Specifies the name of the thread on which the breakpoint stops
- `--thread-id` (`-t`) Specifies the ID (TID) of the thread on which the breakpoint stops
- `--thread-index` (`-x`) Specifies the index of the thread on which the breakpoint stops

For example, the following code snippet shows how to modify the first breakpoint to be removed the first time it stops at any of its locations:

```
(lldb) breakpoint modify --one-shot 1
```

Note: Many of these configuration options can also be passed when a breakpoint is initially set. Enter the `help breakpoint set` command for a complete list of available options.

Running Commands at a Breakpoint

When a set breakpoint is reached, it stops execution of the program and allows for LLDB commands to be run. You can also specify commands to run each time a breakpoint is reached by using the `breakpoint command add` command, which takes a breakpoint ID or a location ID as an argument. For example, to add a command to the first location of the first breakpoint:

```
(lldb) breakpoint command add 1.1
Enter your debugger command(s). Type 'DONE' to end.
> thread backtrace
> DONE
```

By default, the `breakpoint command add` command uses the LLDB command interpreter and opens an interactive prompt with lines beginning with a right angle bracket (`>`). Enter one command per line.

When you are finished entering commands, type `DONE` to exit the interactive prompt.

> **Important:** Commands are not validated when initially added to a breakpoint. If a breakpoint command does not appear to be executing, be sure to check the command syntax, to ensure it's correct.

If you enter the `process continue` command as the last breakpoint command, the debugger automatically continues running the program after executing all of the preceding commands. This is especially convenient for logging information about program state without disrupting user interactivity with the program.

```
(lldb) breakpoint command add 1.1
Enter your debugger command(s). Type 'DONE' to end.
> frame variable
> process continue
> DONE
```

To specify a command to execute for a specified breakpoint inline, rather than in an interactive prompt, you pass the `--one-liner` (`-o`) option with a single-line command surrounded by quotes as its value, to the `breakpoint command add` command.

```
(lldb) breakpoint command add 1.1 -o "bt"
```

Disabling and Enabling a Breakpoint

When a logical breakpoint is disabled, it does not stop at any of its locations. To disable a breakpoint from stopping without removing it, use the `breakpoint disable` command.

Disable a logical breakpoint by passing a breakpoint ID as an argument.

```
(lldb) breakpoint disable 1
1 breakpoints disabled.
```

Disable an individual breakpoint location using the `breakpoint disable` command, passing a location ID.

```
(lldb) breakpoint disable 2.1
1 breakpoints disabled.
```

Enable a logical breakpoint or a breakpoint location using the `breakpoint enable` command, passing a breakpoint ID or a location ID. In the following example, the first breakpoint is enabled, and then the first location of the second breakpoint is enabled:

```
(lldb) breakpoint enable 1
1 breakpoints enabled.
(lldb) breakpoint enable 2.1
1 breakpoints enabled.
```

To enable only certain locations of a logical breakpoint, use the `breakpoint disable` command, passing the breakpoint ID followed by a dot-separated wildcard character (`*`), and then use the `breakpoint enable` command, passing any individual breakpoint locations that you want to enable.

In the following example, all locations of the first breakpoint are disabled, and then the first location of the first breakpoint is enabled:

```
(lldb) breakpoint disable 1.*
2 breakpoint disabled*
(lldb) breakpoint enable 1.1
1 breakpoint enabled.
```

Deleting a Breakpoint

Deleting a breakpoint disables it and prevents it from being reenabled. Delete a breakpoint using the `breakpoint delete` command, passing a breakpoint ID as an argument. For example, to delete the first breakpoint:

```
(lldb) breakpoint delete 1
1 breakpoints deleted; 2 breakpoint locations disabled.
```

Watchpoints

A *watchpoint* is a kind of a breakpoint that you set on an address or variable, to stop anytime a value is accessed, rather than being set at a point of execution.

Watchpoints are limited by the number of registers on the hardware that is running the program being debugged.

You use a watchpoint to isolate when a variable is changed during execution, which can be especially useful for debugging state that is shared across multiple components in code. Once you know where and how a variable is changed, you create breakpoints at points in execution that you want to investigate, and then remove the watchpoint.

Setting a Watchpoint

You use the `watchpoint set variable` command to set a watchpoint on a variable, and the `watchpoint set expression` command to set a watchpoint on an address from an expression.

```
(lldb) watchpoint set variable places
Watchpoint created: Watchpoint 1: addr = 0x100004a40 size = 8 state = enabled type =
w
    declare @ 'main.swift:7'
    watchpoint spec = 'places'
    new value: 1 value
(lldb) watchpoint set expression -- (int *)$places + 8
Watchpoint created: Watchpoint 2: addr = 0x100005f33 size = 8 state = enabled type =
w
    new value: 0x0000000000000000
```

By default, a watchpoint monitors write access to a variable or address. Specify the type of access to monitor by passing the `--watch` (`-w`) option with a value of `read`, `write`, or `read_write`.

A watchpoint monitors reads and writes using the target's pointer byte size by default. Change the number of bytes used to watch a region by passing the `--size` (`-s`) option with a value of `1`, `2`, `4`, or

8.

Listing Watchpoints

As with breakpoints, you use the `watchpoint list` command to show all watchpoints that have been set.

```
(lldb) watchpoint list
Current watchpoints:
Watchpoint 1: addr = 0x100004a50 size = 8 state = disabled type = w
    declare @ 'main.swift:7'
    watchpoint spec = 'places'
```

Like a breakpoint, a watchpoint has an assigned integer ID. Unlike a breakpoint, a watchpoint does not resolve to any specific locations, as it instead monitors a variable or an address for any changes.

Modifying a Watchpoint

Use the `watchpoint modify` command to configure when a set watchpoint stops, by passing a watchpoint ID as an argument along with the `--condition` (`-c`) option and an expression as its value.

```
(lldb) watchpoint modify --condition !places.isEmpty
1 watchpoints modified.
```

Adding Commands to Watchpoints

To add commands to run whenever a watchpoint is hit, use the `watchpoint command add` command, passing the watchpoint ID as an argument.

```
(lldb) watchpoint command add 1
Enter your debugger command(s).  Type 'DONE' to end.
> bt
> DONE
```

Deleting a Watchpoint

Because watchpoints are hardware-limited, it is important to delete them after they are no longer needed. You delete a watchpoint using the `watchpoint delete` command, passing the watchpoint ID as an argument.

```
(lldb) watchpoint delete 1
1 watchpoints deleted.
```