Debugging on macOS

If you experience crashes or issues in Electron that you believe are not caused by your JavaScript application, but instead by Electron itself, debugging can be a little bit tricky especially for developers not used to native/C++ debugging. However, using `lldb` and the Electron source code, you can enable step-through debugging with breakpoints inside Electron's source code. You can also use XCode for debugging if you prefer a graphical interface.

Requirements

- **A testing build of Electron**: The easiest way is usually to build it from source, which you can do by following the instructions in the build instructions. While you can attach to and debug Electron as you can download it directly, you will find that it is heavily optimized, making debugging substantially more difficult. In this case the debugger will not be able to show you the content of all variables and the execution path can seem strange because of inlining, tail calls, and other compiler optimizations.
- **Xcode**: In addition to Xcode, you should also install the Xcode command line tools. They include LLDB, the default debugger in Xcode on macOS. It supports debugging C, Objective-C and C++ on the desktop and iOS devices and simulator.
- **.lldbinit**: Create or edit `~/.lldbinit` to allow Chromium code to be properly source-mapped.

  ```
  # e.g: ['~/electron/src/tools/lldb']
  script sys.path[:0] = ['<...path/to/electron/src/tools/lldb>']
  script import lldbinit
  ```

Attaching to and Debugging Electron

To start a debugging session, open up Terminal and start `lldb`, passing a non-release build of Electron as a parameter.

```
$ lldb ./out/Testing/Electron.app
(lldb) target create "./out/Testing/Electron.app"
Current executable set to './out/Testing/Electron.app' (x86_64).
```

Setting Breakpoints

LLDB is a powerful tool and supports multiple strategies for code inspection. For this basic introduction, let's assume that you're calling a command from JavaScript that isn't behaving correctly - so you'd like to break on that command's C++ counterpart inside the Electron source.

Relevant code files can be found in `./shell/`.

Let's assume that you want to debug `app.setName()`, which is defined in `browser.cc` as `Browser::SetName()`. Set the breakpoint using the `breakpoint` command, specifying file and line to break on:

```
(lldb) breakpoint set --file browser.cc --line 117
Breakpoint 1: where = Electron Framework`atom::Browser::SetName(std::__1::basic_string<char,
```

Then, start Electron:

```
(lldb) run
```

The app will immediately be paused, since Electron sets the app's name on launch:

```
(lldb) run
Process 25244 launched: '/Users/fr/Code/electron/out/Testing/Electron.app/Contents/MacOS/Ele
Process 25244 stopped
* thread #1: tid = 0x839a4c, 0x0000000100162db4 Electron Framework`atom::Browser::SetName(th
    frame #0: 0x0000000100162db4 Electron Framework`atom::Browser::SetName(this=0x000000010
   115  }
   116
   117  void Browser::SetName(const std::string& name) {
-> 118    name_override_ = name;
   119  }
   120
   121  int Browser::GetBadgeCount() {
(lldb)
```

To show the arguments and local variables for the current frame, run `frame variable` (or `fr v`), which will show you that the app is currently setting the name to "Electron".

```
(lldb) frame variable
(atom::Browser *) this = 0x0000000108b14f20
(const string &) name = "Electron": {
    [...]
}
```

To do a source level single step in the currently selected thread, execute `step` (or `s`). This would take you into `name_override_.empty()`. To proceed and do a step over, run `next` (or `n`).

```
(lldb) step
Process 25244 stopped
* thread #1: tid = 0x839a4c, 0x0000000100162dcc Electron Framework`atom::Browser::SetName(th
    frame #0: 0x0000000100162dcc Electron Framework`atom::Browser::SetName(this=0x000000010
   116
   117  void Browser::SetName(const std::string& name) {
   118    name_override_ = name;
-> 119  }
```

```
   120
   121  int Browser::GetBadgeCount() {
   122    return badge_count_;
```

NOTE: If you don't see source code when you think you should, you may not have added the `~/.lldbinit` file above.

To finish debugging at this point, run `process continue`. You can also continue until a certain line is hit in this thread (`thread until 100`). This command will run the thread in the current frame till it reaches line 100 in this frame or stops if it leaves the current frame.

Now, if you open up Electron's developer tools and call `setName`, you will once again hit the breakpoint.

Further Reading

LLDB is a powerful tool with a great documentation. To learn more about it, consider Apple's debugging documentation, for instance the LLDB Command Structure Reference or the introduction to Using LLDB as a Standalone Debugger.

You can also check out LLDB's fantastic manual and tutorial, which will explain more complex debugging scenarios.