November 12th, 2019

Choices I’ve considered for Savitar’s Text Engine:

NSTextView

WKWebView

iTerm2Lib

I started down the path of using NSTextView until I got to the point where I wanted to start rendering ANSI codes and also thought about how things like <code> would need to be implemented. I wanted both of course. So, If I used NSTextView, I’d have to do an ANSI-to-attributedString parser as well as an HTML-to-attributedString parser. I found an ANSI-to-HTML parser (aha.c). This got me thinking of using WKWebView, because itcould handle the HTML natively and aha.c would provide the ANSI code support. Also, it gave room to doing other media types easily, including links of course.

I then thought WKWebView would be just too darn slow. So, I set my sights on iTerm2Lib, for it was performant, could handle ANSI codes, and well, I could live without HTML… I guess.

The problem with iTerm2Lib is I’m carving it up into smaller pieces, and the network aspect isn’t going to be a joy to deal with.

What if WKWebView was in fact performant? Wouldn’t that be the ideal means to render text for the next generation of Savitar? WKWebView claims to be: “*Boasting responsive 60fps scrolling, built-in gestures, streamlined communication between app and webpage, and the same JavaScript engine as Safari”*.

Hey, JavaScript would be a nice addition too. I think I’m back to exploring WKWebView.

November 13th, 2019

Next thing I want to do is do some local testing, get some ANSI-to-HTML conversion working.

November 29th, 2019

I moved over to using WkWebView for output and am successfully processing ANSI escape codes using a hacked-up aha.c. The Appearance settings tab is fully operational, allowing the setting of background and foreground colors, as well as body and code fonts and sizes, all through <head> CSS style. This has proven to be a great success for me! I’m unblocked from pondering how to display output and can now focus on the meat of the application.

November 30th, 2019

I got IAC telnet command parsing working. The next task is to add some internal logging support and then onto triggers.

December 10th, 2019

I started a laundry list to get into start of beta:

√ Started a private github repo

√ App is 64bit only, runs on Catalina

√ App is integrated with AppCenter, handles crash reporting and basic analytics

√ Reading Sav 1.x world settings, opening sessions

√ Integrated WKWebView as the output pane

√ World settings Appearance tab is operational

√ Output triggers are working

\_ Load Sav 1.x app settings (includes triggers)

\_ Load Sav 1.x world triggers

\_ Input pane command recall and local commands supported

\_ Triggers Window implemented

\_ Implement World settings Starting tab

\_ Connect/disconnect session handling

\_ Implement remaining World settings tabs

\_ Logging

\_ Add check for updates support (Sparkle?)

\_ Add bug reporting support

\_ Enhanced analytics

### start of beta

\_ Move github repo to public

\_ Release alpha to select testers, start getting feedback

\_ rewrite Aha

\_ ANSI Color Settings window implemented

\_ input triggers (? does anyone use these?)

\_ Macro Clicker

\_ MCP (? does anyone use this?)

\_ Audio & Speech

\_ File upload

\_ Polish

### Post first release

These features take Savitar 2.0 beyond what 1.6.x provides:

\_ SSL support

\_ Javascript ?

\_ ???

January 4th, 2020

Happy New Year.

The most current laundry list:

### start of alpha

```

√ Started a private github repo

√ App is 64bit only, runs on macOS 10.12 and later, including Catalina

√ App is integrated with AppCenter, handles crash reporting and basic analytics

√ Reading Sav 1.x world settings, opening sessions

√ Integrated WKWebView as the output pane

√ World settings Appearance tab is operational

√ Output triggers are working

√ Load Sav 1.x app settings (includes triggers)

√ Load Sav 1.x world triggers

\_ Input pane command recall and local commands supported

\_ Implement input triggers

\_ Implement Triggers Window

\_ Implement World settings Starting tab

\_ Connect/disconnect session handling

\_ Implement remaining World settings tabs

\_ Menubar finalized

\_ Logging

\_ Add check for updates support (Sparkle?)

\_ Add bug reporting support

```

### start of beta

```

\_ Move github repo to public

\_ Release alpha to select testers, start geting feedback

\_ rewrite Aha

\_ Enhanced analytics

\_ ANSI Color Settings window implemented

\_ Macro Clicker

\_ MCP (? does anyone use this?)

\_ Audio & Speech

\_ File upload

\_ Polish

```

### Post first release

These features take Savitar 2.0 beyond what 1.6.x provides:

```

\_ SSL support

\_ Dark Mode support

\_ Javascript scripting?

\_ ???

January 11th, 2020

I wrote the following in PR #6 this morning: “Wrapping up this current work. Going to explore using Core Data instead of SavitarManager and NSCopyable. That'll clean-up the code and allow more idiomatic use of Swift, as well as open the door to possible SwiftUI work too.”

Then I thought about it and determined rewriting into Core Data is outside the scope of the initial release of Savitar 2.0.

January 12th, 2020

I’m on my way into the Events Window now, modifying a world’s triggers for the first time (editing trigger names and enabled/disabled right now). This brings up the question of “How does NSDocument get marked dirty?” Well, it’s tied into the Undo Manager. I suppose it would be fantastic to support undo for world trigger edits. Which is a separate thing from modifying universal triggers.

I guess I need to do a full study of the NSDocument…

First thing: NSUndoManager is fairly straightforward to deal with. However, it’ll require associating editable things, like a world’s triggers, to the world’s document’s undo manager.