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. This can be done via the SavitarManager (TriggerManager).

The problem with the Events Window is that there are multiple UndoManagers involved. Each open world has its document’s undo manager, and then the universal triggers/variables have the app-wide undo manager. Simply registering undo with each triggerman’s document undomanager isn’t enough, because the menu item won’t change when the Events Window is in the foreground, it only appears when the document is in the foreground. **windowWillReturnUndoManager** apparently will help withthis. Alternatively, I’d have to tie the triggers/variables right to the document window, like how settings works. If I went this route, I would not have to deal with doing an NSOutlineView-based Events Window. Simplifying the work, giving up drag and drop for closer association to the world, but also on the flip side, giving up UI parity with Savitar1’s Event Window. Another downside of settings sheet-based triggers/variables is you can’t test them as easily, you’d have to keep dismissing the entire sheet to get back to text input (the sheet is modal).

The bottom line is: the world’s document needs to go dirty when a trigger is enabled/disabled or edited. Having undo/redo is gravy on top of that, and comes almost for free if done the right way.

January 16th, 2020

Next Savitar challenge:

When a World’s title changes, the change must propagate to the trigger manager and var manager, which in turn will signal a change to any open Event Window.

One way to simplify things is to remove the name attribute off of SavitarManager (triggerman and varMan) and instead have Event Manager enumerate open worlds as needed.

Also noted: A world’s document window is listing its file name, not its world name. That is actually a good thing, makes it easy to find the document. Perhaps one doesn’t rename a world in settings at all, just use the filename to refer to it.

So, I removed the ‘name’ field from Savitar Manager and instead taught the Event Window how to name trigger managers by world or universally (app wide).

January 20th, 2020

I decided over the weekend to use a Flux design pattern for Savitar2’s data model. It’ll be implemented using ReSwift.

Things that go into the Store:

World Document

Actions: add\_world, remove\_world

Listeners: Event Window

So, good. Event Window knows to update automatically its groups. But, what happens when a world’s trigger is edited at the document? How does Event Window listen for that?

Trigger

Actions: add\_trigger, remove\_trigger, enable\_trigger, disable\_trigger

Those are the easiest actions... then there are:

Rename\_trigger, change\_trigger, etc.

Rename and change should be the same, I think. The granularity is at the trigger as a whole.

So, along with observing a world document, should Event Windows observe a world’s triggers individually? And if so, then how does the trigger equate back to its trigger man? Or… is a Trigger Man even necessary any longer? Surely one is needed to preserve order of triggers, meaning we need a “move\_trigger” action. May as well have “add\_trigger” and “remove\_trigger” be peers to “move\_trigger” then.

These are considerations general to Flux/Redux. Probably every TODO app ever made has this problem to solve (add, remove, move, change TODOs).

A google search for redux+add+remove+objects+change reveals…

“**Redux Patterns: Add/Edit/Remove Objects in an Array**” <https://hackernoon.com/redux-patterns-add-edit-remove-objects-in-an-array-6ee70cab2456> -- covers using a `byId` and `byHash` approach and is made reference to as a best practice here: <https://www.reddit.com/r/reactjs/comments/6o9tfq/redux_patterns_addeditremove_objects_in_an_array/>

“**Immutable Update Patterns**” <https://redux.js.org/recipes/structuring-reducers/immutable-update-patterns/> -- covers handling immutablity of arrays. Copying of nested objects.

“**Immutability in React and Redux: The Complete Guide”** <https://daveceddia.com/react-redux-immutability-guide/> -- general reading

“**Why you should use an object, and not an array, for lists in Redux**” <https://medium.com/javascript-in-plain-english/https-medium-com-javascript-in-plain-english-why-you-should-use-an-object-not-an-array-for-lists-bee4a1fbc8bd> -- this is a worthwhile technic, however, triggers have an ordering requirement. Perhaps I can marry this article up with the “**Redux Patterns: Add/Edit/Remove Objects in an Array**” article of keeping a `byId` array.

If we go this way, then every trigger needs to have a unique id.

Bonus: Savitar1 used trigger names as the unique identifier. Savitar2 can use internal (not displayed to the user) unique ids, which then allows for triggers of the same “name” (a trigger’s “name” is a misnomer – it’s really the text matching string.)

NSOrderedSet gives us the best of both worlds. Fast lookup and ordering.

Need to explore how adding/moving items in the NSOrderedSet is done. Do I employ NSMutableOrderedSet? Likely – there is the wonders of `moveObjects()`.

**ACTION:** move\_trigger(oldIndex: Int, newIndex: Int)

new\_state = NSMutableOrderedSet(state)

new\_state.moveObjects…)

return NSOrderedSet(new\_state)

**ACTION:** remove\_trigger(id)

new\_state = NSMutableOrderedSet(state)

new\_state.remove(id)

return NSOrderedSet(new\_state)

Is it necessary to have a trigger man if all interactions with triggers are done thru actions? Is the trigger man basically the ordered set being operated on?

I think so, unless of course we want to wrap the actions with methods off of trigger man. Not sure what that gains us. Let’s look at what ReSwift action calls look like.

Read this code next: <https://github.com/ReSwift/ReSwift-Todo-Example/blob/master/ReSwift-Todo/ToDoListActions.swift>

January 23rd, 2020

Maybe I should write a simple stand-alone app that demonstrates the management of triggers. But, why? Mainly because current triggers are classes whereas ReSwift and modern Idiomatic Swift prefers structs. They are objects because: @obj dynamic name field.

We’re using @obj dynamic on the controllers, they need not be on the object itself.

I’d like to start the data model over, but before I do, I’d like to get a working knowledge of ReSwift under my belt.

What would the sample app do?

It would allow the opening of multiple documents. In each document, an ordered array of employees is presented, representing a “job site”:

struct employee {

name: String

skills: String

}

There would be a add button on each document. Removal is done by highlighting the employee and hitting the delete key.

There would be an “All Employees” window that lists all the documents by group and allow add/removal, and most importantly MOVE of employees.

February 3rd, 2020

Start of new internal repo, ReSwift example.

Each document will contain a list of employees for a job. (Job)