SproutCore UI
Specification

Majd Taby
Version 1.2
June 22, 2011

Changelog

Version 1.2

- Explain the purpose of the views

Version 1.1

- Added “Superset of SproutCore” section
- Updated Roadmap

Overview

About the Project

SproutCore UI is the next generation User Interface layer built specifically for SproutCore 2.0. It provides a standardized set of components which can be used to build a consistent user experience that scales to different form factors.

Motivation

Today’s users live not just on one or two devices, but on a wide array of devices from their phone, to the tablet, to the computer, and even the TV. It is common for application developers to want to deliver an application to not just one of those platforms, but to multiple, or even all. The ability to share code between those different form factors is essential in lowering cost, and speeding up delivery time.

One of the major problems application developers face when trying to solve this problem is building a UI that is both consistent, and scalable. SproutCore UI attempts to address both problems head on as it was conceived with those requirements in mind.

Audience

SproutCore UI is designed both for the advanced SproutCore developer who is familiar with the intricacies of framework, as well as more novice developers who are either experimenting, or taking on their first SproutCore project. It allows the novice developer to build complex interactions easily, as well as the ability to bootstrap a UI with no design work. It also allows the advanced SproutCore developer to easily create their own reusable components and theme them easily to create a unique experience for their users.

Platform Support

In order to deliver a consistent experience across multiple form factors, SproutCore UI is built with support for the following browsers:

- Mobile
    - iOS Phone and Tablet
    - Android Phone and Tablet
    - Windows Phone and Slate
    - RIM PlayBook

    - WebOS Phone and Tablet
- Desktop
    - Safari 5+
    - Chrome (n-1)
    - IE8+ (IE7?)
    - FireFox 4+

Requirements

Superset of SproutCore

All the components of SproutCore UI must be supersets of their counterparts in SproutCore. That is to say, if a developer has an SC.ButtonView in their template, they should be able to change the SC namespace to UI and have the template still be valid.

Opt-in

A major feature of SproutCore 2.0 is its ability to be used alongside other applications, as well as its ability to be used to build small, html based applications. In order to maintain that requirement, SproutCore UI will be shipped as a separately installed package. As of the time of this writing, the ongoing assumption is that SproutCore UI will ship as a “spade” – part of the Spade packaging system.

Decoupling

In an effort to modularize SproutCore UI and make it possible to whitelist which files are added and blacklist which files are ignored, SproutCore UI will attempt to decouple the components and the various parts of the project as much as possible.

Documentation

SproutCore 1.X suffered from a perpetual lack of documentation that plagued it its entire life. In order to avoid that same pitfall, code will not be accepted into the master branch until it is documented according to the SproutCore Documentation Guidelines document.

Testing

In order to be able to move and iterate rapidly on SproutCore UI, every feature will be checked to ensure that it is unit tested before integration into the master branch. Proper and extensive unit testing will allow us to confidently refactor components, as well as ensuring quality.

Along with unit testing, integration testing is an integral part of the testing story with SproutCore UI. The components are designed to work well with each other, and to test them, sample applications will be built and tested to ensure the components work harmoniously.

Architecture

SproutCore UI makes a distinction between two different categories of views: Structural views, and component views. They serve different purposes, and handle layout differently.

Component Views

The basic set of widgets which can be used to add standard app-like behavior to an application. These widgets include views such as ButtonView, SliderView, CheckboxView, ImageView, etc. They rely on the browser's rendering engine to lay out the views, and scale to fit the size of their contents. Component views are all designed to provide a foundation to build on. For example, an application that requires a custom button may override the default template, but still use the target/action dispatching behavior of UI.ButtonView.

Structural views

These views provide a the structure and chrome of a web app. Generally speaking, they behave like containers of HTML and component views. The set of Structural views includes views such as: SplitView, NavigationView, CardView, PopoverView, etc.

Since component views rely on the HTML and CSS for their presentation and structure. and since that is standard across any web-connected device, the scalability of an application across form factors is provided by controller-backed views.

This class of views is associated with a special kind of controller called a View-Controller. View controllers are tied to a specific view instance and manage its display and its behavior. One of the main differentiator of controller-backed views from content views is how layout is managed.

Layout

The layout of views is managed differently depending on the two types of views. For content view, layout is dictated by the browser's layout engine, and can be modified using CSS. For controller-backed views however, layout is managed by SproutCore UI.

By default, controller-backed views fill their container. An implication of that behavior is that they define `position:absolute`. SproutCore UI exposes properties to the Handlebars helpers to allow developers to customize the positioning of the views relative to their parents. Only the position and dimension of controller-backed views is defined in the handlebars template. All other presentational styles are defined in CSS.

View Controllers

In a traditional MVC architecture, the model wraps and maintains the data layer, the view defines the visual and interaction, and the controller defines the domain-specific interaction between the model and the view. View-Controllers blur the line between the view and the controller since they expose a lot of the view's responsibility and its behavior to the controller.

The reason View Controllers are a powerful and useful concept is that they allow a single entity to create and manage a set of views that work to serve a single purpose. For example. A NavigationViewController not only manages the navigation stack, but also the creation and state of the NavigationBar and its items.

Scalability

One of the main requirements of SproutCore UI was that it scales beyond a single platform and form factor. The rendering engine of the browser already solves most of the rendering and layout problems. What SproutCore UI has to build however, is a way for concepts and conventions to translate from one form factor to another. For example, on the iPhone UINavigationView is one of the primary forms of interaction across applications. What Apple has successfully done, is to create a mapping to the iPad with Popovers. That way, a universal application can be built for both the iPhone and the iPad and share the same codebase. SproutCore UI needs to provide a toolkit that similarly translates interactions between phones, tablets, and computers.

Controller-backed views play a major role in this scalability requirement. Because SproutCore UI can dictate the behavior and presentation of those views, we can automate the translation process. There will be, of course, platform-specific components. For example, a SplitView makes a lot of sense for master-detail UIs for the desktop and tablet form factors, but is not suitable for phones.

Components

ButtonView

Extends SproutCore's built-in SC.ButtonView by adding the ability to theme the look and feel of the SproutCore UI themed button. It also adds toggle capability to the view.

TextFieldView

Extends SproutCore's built-in SC.TextFieldView by adding theming support.

CheckboxView

Extends SproutCore's built-in SC.CheckboxView by adding theming support and the ability to be in a "mixed" state.

RadioView

Extends SproutCore's built-in SC.RadioView by adding theming support and the ability to be in a "mixed" state.

ImageView

Uses Canvas to display images on the screen and adds the ability to specify the resizing behavior.

SliderView

Given a range and at a step amount, UI.SliderView will display a slider with a draggable thumb.

SegmentedButtonView

Displays a list of buttons next to each other. Used to switch between “modes” in an app.

ProgressView

Used to display progress of an event.

DropDownView

Allows the user pick an item from a list

CardViewController

Allows the user to scroll through pages of content. Supports swiping and panning gestures to navigate through the pages.

NavigationViewController

Used to navigate through a hierarchy of content horizontally.

TabViewController

SplitViewController

PopoverViewController