Legacy - iOS SDK Interface Customization

Our iOS SDK is 100% open source, and as such, you can modify the UI and underlying code at any time. However, the Alchemer Mobile iOS SDK is built in such a way that simple UI style changes are very easy to make without having to modify our code.

Customizing the User Interface

There are four main ways of customizing the Alchemer Mobile iOS UI, from least to most complex:

1. Using `tintColor` and `UIAppearance`
2. Using the Alchemer Mobile style sheet object
3. Using a custom style object
4. Forking the Alchemer Mobile iOS SDK

Using `tintColor` and `UIAppearance`

`tintColor`

By default, the Alchemer Mobile UI uses the tint color of your app's main window. To customize the appearance of the tint color within the Alchemer Mobile UI only, you can set the tint color on `UIView`'s `UIAppearance` proxy when contained in instances of `ApptentiveNavigationController`.

For iOS versions prior to 9.0, use the `-appearanceWhenContainedIn:` method in Objective-C:

```
[UIView appearanceWhenContainedIn:[ApptentiveNavigationController class], nil].tintColor = <#custom tint color#>;
```

For projects targeting iOS 9.0 and later, use the `appearanceWhenContainedInInstancesOfClasses(_:)` method (Swift projects targeting iOS versions prior to 9.0 will have to use the aforementioned Objective-C snippet):

```
UIView.appearanceWhenContainedInInstancesOfClasses([ApptentiveNavigationController.self]).tintColor = <#tint color#>
```

Note that the navigation bars used in the Alchemer Mobile UI will default to a white background color (technically a white `barTintColor`) if no further customizations are made. This can cause white status bar text and icons, as well as white navigation item titles and bar button items to be invisible. In this case you should make sure to set a contrasting bar tint or background color for the navigation bar as described above.

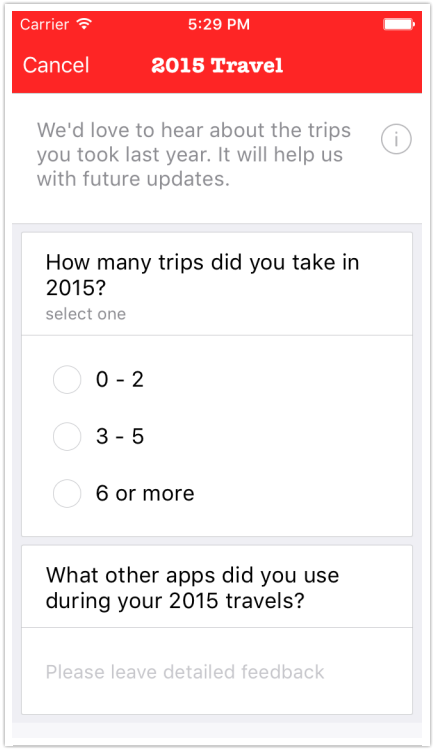
Global Appearance

You can use iOS's built-in `UIAppearance` capabilities to customize a number of global color and font attributes. By default, the Alchemer Mobile UI will use these settings if they are set globally or when contained in instances of `ApptentiveNavigationController` (Note that if no appearance overrides are set, the navigation bar in the Alchemer Mobile UI will default to a white bar tint color).

For instance, you can customize your app's navigation bar as follows, and the Alchemer Mobile UI will use these same values:

```
 UINavigationBar.appearance().barTintColor = UIColor.redColor()
UINavigationBar.appearance().titleTextAttributes = [
    NSForegroundColorAttributeName: UIColor.whiteColor(),
    NSFontAttributeName: UIFont(name: "AmericanTypewriter-Bold", size: 17.0)!
]
UIBarButtonItem.appearance().tintColor = UIColor.whiteColor()
UIApplication.sharedApplication().statusBarStyle = .LightContent // Also set UIViewControllerBasedStatusBarA
ppearance to NO in Info.plist
```

This will set the navigation bar color to red, with a white title and buttons, and a typewriter-style font:



If your app requires view controller-based status bar styles, you can extend `ApptentiveNavigationController` to specify `.lightContent` as its preferred status bar style:

```
extension ApptentiveNavigationController {
 open override var preferredStatusBarStyle: UIStatusBarStyle {
  get {
   return .lightContent
  }
 }
}
```

Using a Style Sheet Object

For non-global styles, using `UIAppearance` quickly becomes cumbersome or impossible (for instance, consider a standard `UITableViewCell` with two labels; there is no way to individually target those labels using `UIAppearance`). To supplement `UIAppearance`, we have added a style sheet object to the 3.0 release of the Alchemer Mobile iOS SDK.

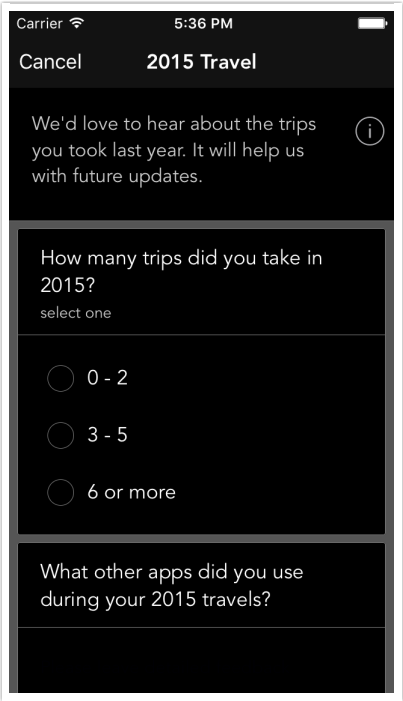
The style sheet object is a read-write property of the `Apptentive` singleton that conforms to the `ApptentiveStyle` protocol. You can either use the `ApptentiveStyleSheet` object created when the singleton is initialized, or set your own object that conforms to the `ApptentiveStyle` protocol after initialization.

Using `ApptentiveStyleSheet`

The style sheet object provides a series of colors and fonts for use by various parts of the Alchemer Mobile UI. Many of the fonts are described by the `UIFontTextStyle` string constants, such as `UIFontTextStyleBody`. In addition, a number of text styles and colors have custom keys that you can find in the `ApptentiveStyleSheet.h` file. All font sizes respect the Dynamic Type settings for the device.

Below is an example of how you might create a “dark mode” theme, using a font other than the built-in system font:

```
 UINavigationBar.appearance().barTintColor = UIColor.blackColor()
UINavigationBar.appearance().titleTextAttributes = [
    NSForegroundColorAttributeName: UIColor.whiteColor(),
    NSFontAttributeName: UIFont(name: "Avenir-Heavy", size: 17.0)!
]
UIBarButtonItem.appearance().tintColor = UIColor.whiteColor()
UIApplication.sharedApplication().statusBarStyle = .LightContent // Also set UIViewControllerBasedStatusBarA
ppearance to NO in Info.plist

if let style = Apptentive.sharedConnection().styleSheet as? ApptentiveStyleSheet {
    style.fontFamily = "Avenir"
    style.regularFaceAttribute = "Book"
    style.boldFaceAttribute = "Heavy"

    style.backgroundColor = UIColor.blackColor()
    style.primaryColor = UIColor.whiteColor()
    style.secondaryColor = UIColor(white: 0.8, alpha: 1.0)
    style.separatorColor = UIColor.grayColor()
    style.failureColor = UIColor(red: 1, green: 0.5, blue: 0.5, alpha: 1.0);
    style.collectionBackgroundColor = UIColor.darkGrayColor()
}
```



Fonts

To use a non-system font, set the `ApptentiveStyleSheet`'s `fontFamily` property to the name of the font you would like to use. You can use the iOS Fonts website to find the font family name for the fonts included with iOS (you'll want to use the name in the heading above the list of typefaces in each font family).

Because the naming of various font weights is not standardized, you may have to customize the modifiers used for identifying each weight of your chosen font. You do this by setting the corresponding face attribute property. For example, the built-in Avenir font's regular weight is called "Avenir-Book", so the regular face attribute is set to "Book". Some fonts omit the suffix entirely for their regular weight. In that case, set the face attribute to `nil`. Available attribute properties are `lightFaceAttribute`, `regularFaceAttribute`, `mediumFaceAttribute`, and `boldFaceAttribute`, which default to "Light", "Regular", "Medium" and "Bold", respectively.

Some fonts may not include all four weights used by the style sheet object. In that case you may need to set one or more of them to use another weight.

Finally, you can globally adjust the font size used by the Alchemer Mobile UI by modifying the `sizeAdjustment` property, which acts as a global multiplier to the font sizes that would otherwise be used.

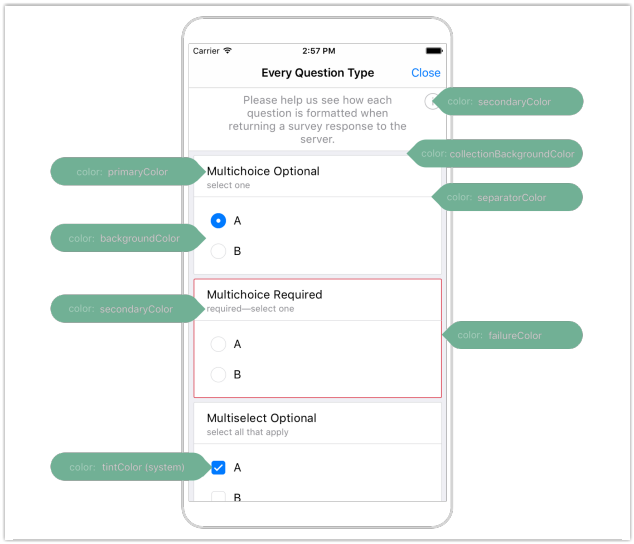
Colors

There are six colors that can be modified to change the appearance of the Alchemer Mobile UI:

- `primaryColor` (defaults to `UILabel`'s `textColor` appearance property)
- `secondaryColor` (defaults to #8E8E93)

- `failureColor` (defaults to #DA3547)
- `backgroundColor` (defaults to `UITableViewCell`'s `backgroundColor` appearance property)
- `separatorColor` (defaults to `UITableView`'s `separatorColor` appearance property)
- `collectionBackgroundColor` (defaults to `UITableView`'s `backgroundColor` appearance property)

The primary, secondary, and failure colors need to contrast well with the background color. The separator and collection background colors are mostly a matter of aesthetics. This survey shows an example of how theses colors are used:



Initializing from a Property List

You can also create an instance of the ApptentiveStyleSheet class using values from a property list. Use the `initWithContentsOfURL:` initializer and set `Apptentive.shared.styleSheet` property to the result. The Alchemer Mobile Cordova integration for iOS does this for you.

The property list consists of a top-level dictionary with keys that are as follows:

- `FontFamily`
- `LightFaceAttribute`
- `RegularFaceAttribute`
- `MediumFaceAttribute`
- `BoldFaceAttribute`
- `PrimaryColor`
- `SecondaryColor`
- `FailureColor`
- `BackgroundColor`
- `SeparatorColor`
- `CollectionBackgroundColor`
- `PlaceholderColor`
- `SizeAdjustment`

- `ColorOverrides`
- `FontOverrides`

All but the last three have string values. Color values should be specified as web color in the format of a pound sign followed by three two-digit hexadecimal numbers specifying red, green and blue (e.g. `#FFFFFF` for white).

The value for the `SizeAdjustment` key is a number specifying a global change to the size of the text in the Alchemer Mobile UI, where 1.0 is normal size. This is useful if a particular font family at the default size is too small or too large everywhere.

The `ColorOverrides` and `FontOverrides` override specific styles and should have dictionary values. The following keys work for both font and color sub-dictionaries:

- `com.apptentive.body`
- `com.apptentive.header.title`
- `com.apptentive.header.message`
- `com.apptentive.message.date`
- `com.apptentive.message.sender`
- `com.apptentive.message.status`
- `com.apptentive.messageCenter.status`
- `com.apptentive.survey.question.instructions`
- `com.apptentive.doneButton`
- `com.apptentive.button`
- `com.apptentive.submitButton`
- `com.apptentive.textInput`

The keys below are only used when specifying colors:

- `com.apptentive.color.header.background`
- `com.apptentive.color.footer.background`
- `com.apptentive.color.failure`
- `com.apptentive.color.separator`
- `com.apptentive.color.cellBackground`
- `com.apptentive.color.collectionBackground`
- `com.apptentive.color.textInputBackground`
- `com.apptentive.color.textInputPlaceholder`
- `com.apptentive.color.messageBackground`
- `com.apptentive.color.replyBackground`
- `com.apptentive.color.contextBackground`

For color overrides, the values again are six-digit hex web colors with a preceding pound sign.

For font overrides, the values are dictionaries. These sub-sub-dictionaries should have keys of `font` and `size`. The value for the `font` e>font key is a string describing the specific typeface (both font and weight—see http://iosfonts.com for a list of those included by the operating system). The value for the `size` key is a number describing the desired size in points.

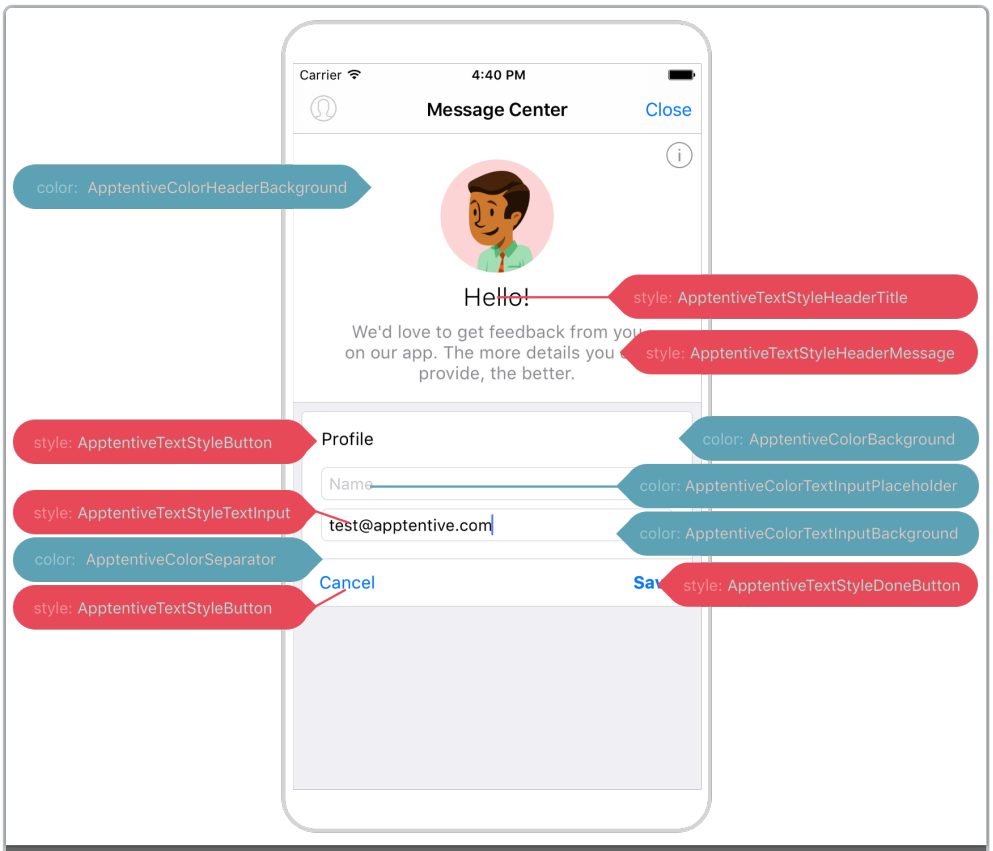
Using Your Own Style Sheet Object

To further customize the fonts and colors used by the Alchemer Mobile UI, you can subclass `ApptentiveStyleSheet` or create your own implementation from scratch that conforms to the `ApptentiveStyle` protocol. Your implementation will have to return appropriate fonts and colors based on the keys listed in the `ApptentiveStyleSheet.h` file.

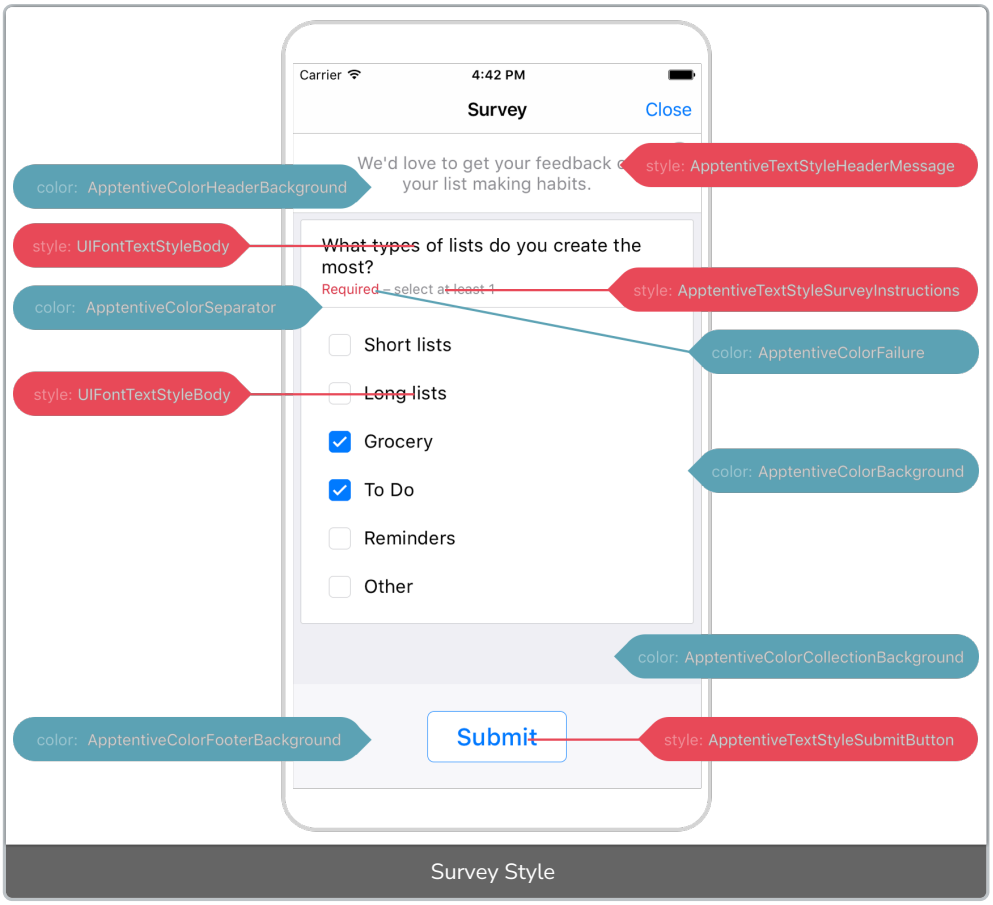
To use your custom style object, simply set the `styleSheet` property on the `Apptentive` singleton to an instance of your style sheet class before displaying Alchemer Mobile UI.

```
class MyStyleSheet: ApptentiveStyle {
    func fontForStyle(textStyle: String) {
        // ...
    }

    func colorForStyle(style: String) {
        // ...
    }
}

// In your App Delegate's application(_:didFinishLaunchingWithOptions:)
Apptentive.sharedConnection().styleSheet = MyStyleSheet()
```

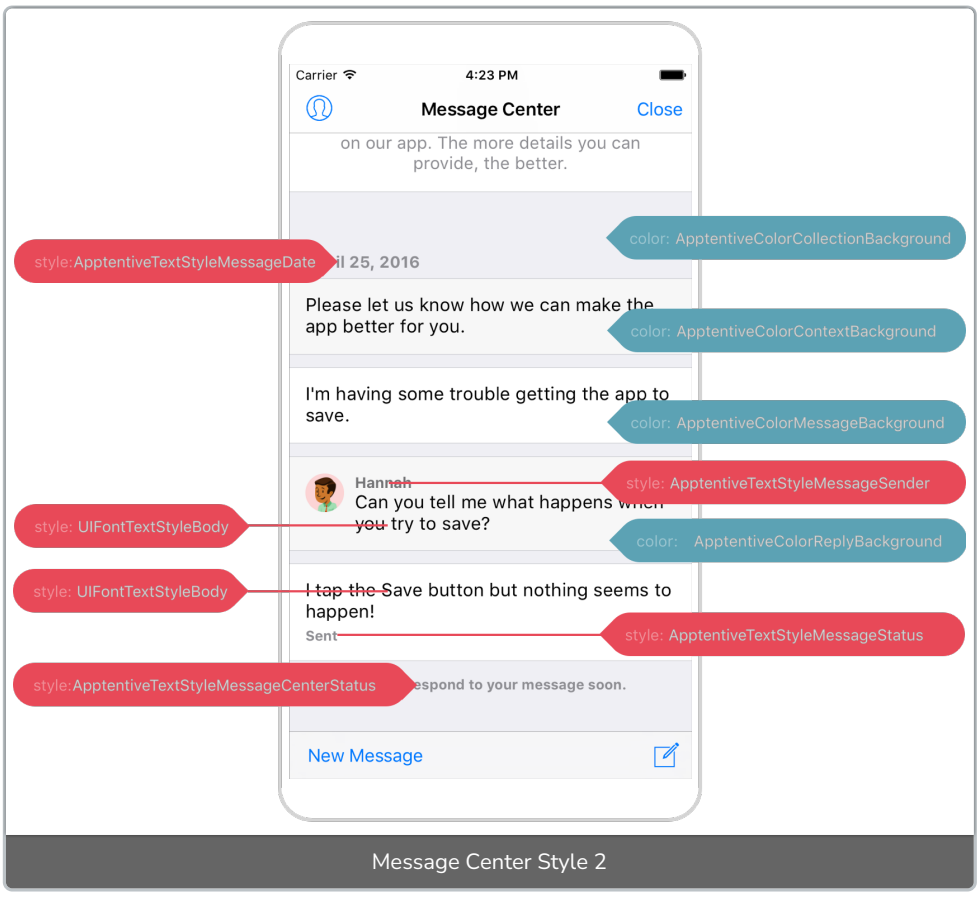
If you would like the Alchemer Mobile UI to use Dynamic Type, your style sheet class will have to adjust font sizes appropriately. For reference the elements that can be styled, with their corresponding style/textStyle keys are illustrated below:



Message Center Style 1



Survey Style



Message Center Style 2

Forking the Alchemer Mobile iOS SDK

In general, we recommend against forking the Alchemer Mobile iOS SDK, since it requires that changes to the main Alchemer Mobile repository be merged into the fork when the former is updated.

We plan to migrate the Alchemer Mobile iOS SDK's architecture to a Model-View-ViewModel system, which will allow forks of the SDK to customize the look and feel of the SDK without re-implementing the logic. The first component that adheres to this architecture is the Surveys component.

As stated above, we encourage you to explore using the built-in capabilities for customization, and contacting us if you have questions or feature requests, before creating a fork and modifying it.

Related Articles