Technical Disclosure Commons

Defensive Publications Series

April 16, 2015

Cross-platform Styling of Native Input Widgets on Mobile Browsers

Ben Greenwood

Joey Scarr

Cross-platform Styling of Native Input Widgets on Mobile Browsers

Abstract

The subject disclosure relates to modifying or "stylizing" the graphical display of a native input widget (also referred to as "standard input widgets", "native semantic input types", or "pickers") in a web page displayed in a web browser, particularly a mobile web browser, so that the native input widget can be configured, for example, to appear to a user to match the visual style (e.g., font size, color, localization format) of the rest of the web page. Specifically, the graphical display of the native input widget appears to the user to be modified by layering a stylized façade element ("façade element") over the location of the native input widget. The façade element can be configured to appear to the user to match the visual theme (e.g., font size, color, localization format) of the rest of the web page. The façade element is connected to the native input widget via an implemented event bridge between the two elements to facilitate responses to user interactions with the façade element, and consequently interaction by the user with the façade element triggers an operating system widget to appear for the selection of values by the user for the standard input widget.

Background

There are very limited, if any, existing mechanisms to customize the appearance of a web browser's native input widget, particularly in mobile web browser. When a user activates a native input widget in the web page displayed in the mobile web browser, such as by clicking or touching an input element of the native input widget, the native input widget commonly activates an operating system level application programming interface (API) for selecting values ("native operating system widget") for the native input widget that appears at the operating system or mobile web browser level. The native operating system widget cannot, however, be stylized and customized by the web page designer, as such stylization of native operating system widgets are commonly restricted to design at the application level. This is due, for example, to a separation between JAVASCRIPT APIs and the application-level native operating system widget API.

Furthermore, after the user has selected an input value using the native operating system widget for display in the native input widget in the web page, the resulting display of the native input widget, such as the styling, commonly cannot be configured by a web page designer so as to, for example, style consistently with the rest of the web page. Very limited approaches, if any, exist for attempting to stylize the native input widget, including replicating semantic capture by using multiple input fields (such as a combination of <select> dropdowns for Red, Green, Blue color channels to imitate a color picker), or imitating a web browser's native input widget using JAVASCRIPT (e.g., such as by using gesture detection and Cascading Style Sheet ("CSS") animations to mimic the user interface of a native input widget).

Features of the subject technology

Mobile web browsers provide mechanisms to capture strong semantics in user input through markup capabilities using HTML-coded native input widgets such as date-time pickers, color chart pickers, etc. Using these native input widgets, developers can often access the performance and native operating system widgets provided by an operating system on the device running the mobile web browser. The subject technology proposes layering a styleable façade element on top of a native input widget, thereby creating the illusion of a styled input widget. The façade element is connected to the native input widget via an implemented event bridge between the two elements to facilitate responses to user interactions with the façade element. Specifically, as illustrated in FIG. 1 below, a z-index 101 layering approach is used so as to maintain the appropriate native browser behavior for the native input widget 102 when the underlying input selector of the native input widget receives focus by the user activating a façade element 103 (e.g., in this case, for a date-time picker) due to the native input widget 102 having a lower z-index value than the façade element 103, as illustrated:

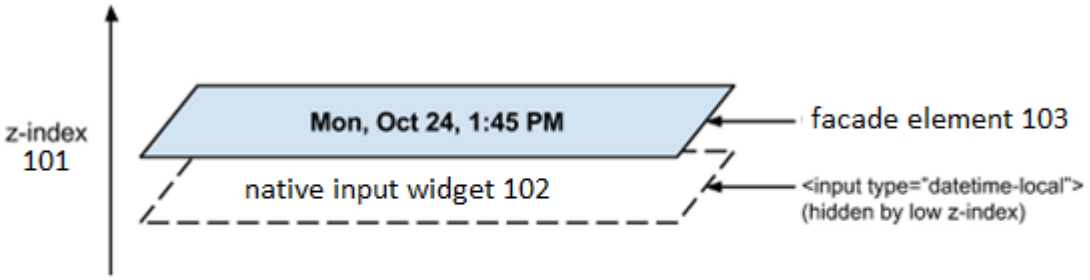

FIG. 1

FIG. 2 provides an example illustration of a webpage in a mobile web browser having a native input widget 102:

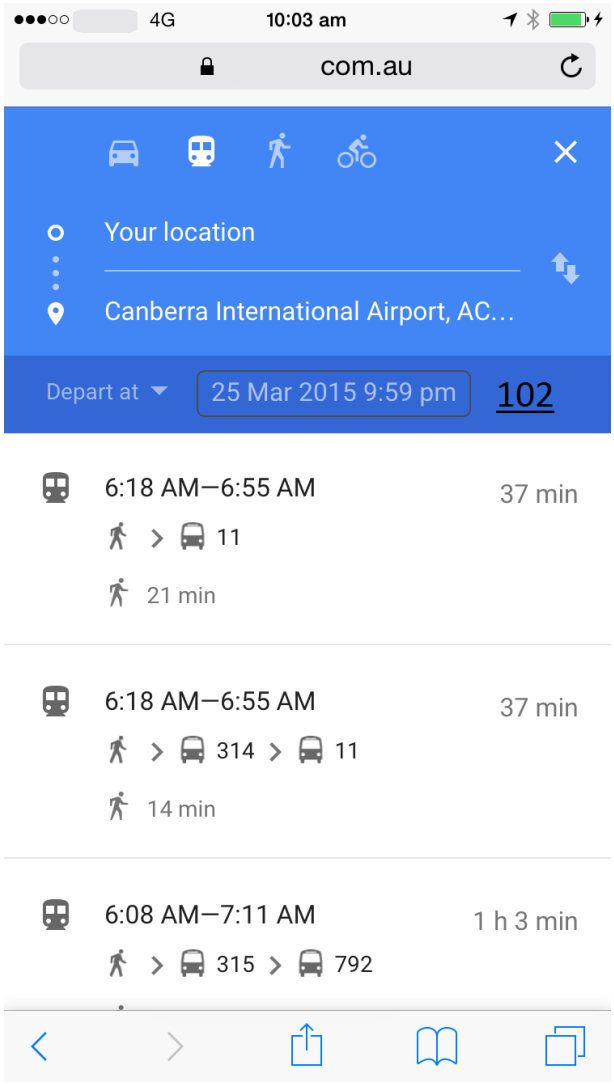

FIG. 2

FIG. 3 provides an example illustration of the webpage of FIG. 2, except with a façade element 103 layered above the native input widget 102:

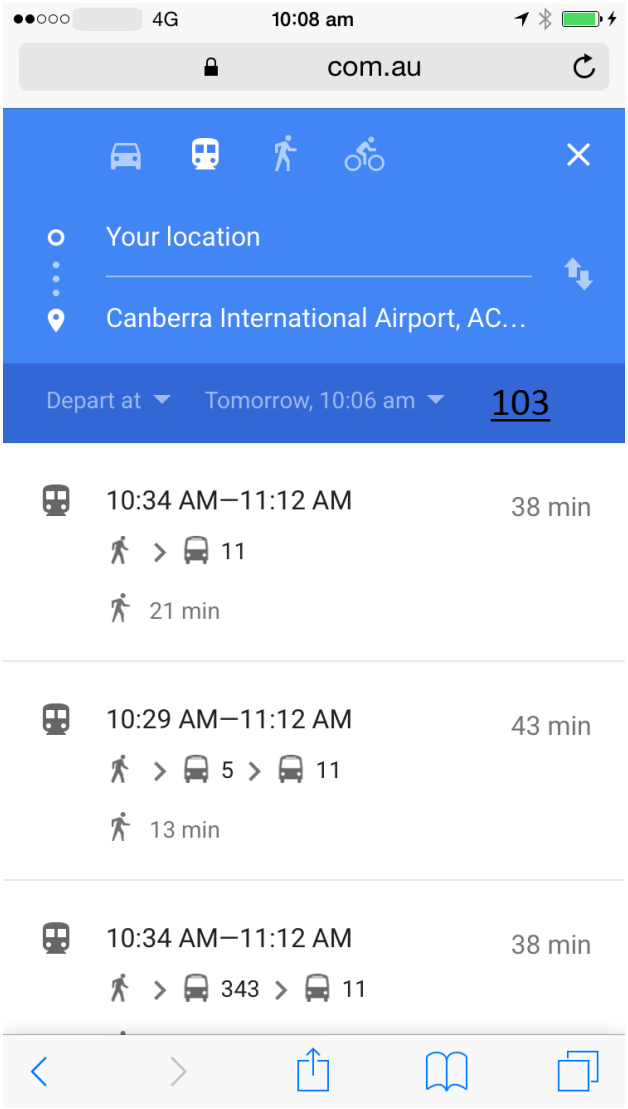

FIG. 3

Onscreen styling of the façade element 103 may be performed using a regular " " tag that is dynamically updated to contain a custom user interface. The façade element 103 may be styled, for example, using color, font (e.g, size and type), and format. The façade element 103 is generated by the " " tag, which acts as a façade that proxies events to and from the native

input widget 102. For example, when the " " tag associated with the façade element 103 receives an "onclick" event, the façade element 103 focuses the hidden native input widget 102 to trigger appearance of the native operating system widget 401 as illustrated in FIG. 4:

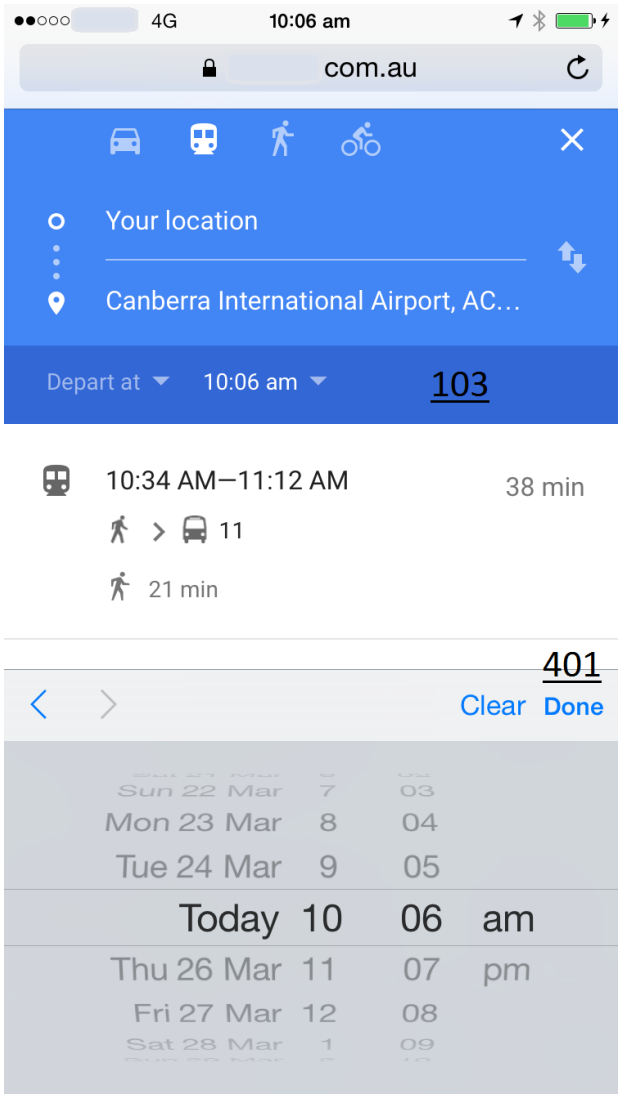

FIG. 4

When the hidden native input widget 102 receives "onchange" events (e.g., from the native operating system widget 401 activated in response to the "onclick" event which focused the native input widget 102), the façade element 103 can respond to the "onclick" events by updating the display of the façade element 103 accordingly. Similarly, "onfocus" and "onblur" events can

be listened to in order to change the styling of the façade element 103 when an input field of the native input widget 102 has focus.

The event bridge can be implemented in two ways. In one approach, the hidden native input widget 102 may be placed directly below the façade element 103, and the web page may use a web browser's event bubbling mechanism to allow click events on the façade element 103 to "fall through" to the hidden native input widget 102. In another approach, an event listener may be added to the "onclick" method of the façade element 103, and in response to the web browser identifying an "onclick" action for the façade element 103, the façade element 103 can explicitly call a "click()" or "focus()" action on the hidden native input widget 102.

Other methods of attempting to hide the native input widget 102, such as adding a "display: none" CSS style, would disallow the hidden native input widget 102 from receiving focus, and therefore prevent the native operating system widget from appearing.

In order for mobile web browsers to correctly scroll the façade element 103 into view on the web page in the mobile web browser when the native input widget 102 is focused, the native input widget 102 should be placed at the same top coordinate of the web page as the façade element 103 on the web page.

For some aspects, certain web-browser-specific behaviors may need to be implemented. For example, scrolling the browser viewport such that the façade element 103 is in view may be done by simulating an event on the hidden native input widget 102 to trigger focus; the specific event to trigger may differ depending on the browser. Similarly, responding to updates to fields of the native input widget 102 may be done in different ways depending on the mobile web browser.