# Implementing Side Panels

## Overview

This document describes how to manage side panel navigation controls using touch gestures.

### What is a side panel?

A side panel is a left or right panel widget positioned off of the device screen. When a user swipes inward from an edge or tilts the Fire phone, the panel slides into view from the side. The direction of the swipe gesture determines whether a right or left panel will be displayed. For example, to show the left panel, start at the left edge of the screen and swipe to the right.

Loading a panel uses the following sequence:

1. On page load, the panel is placed on the left-hand side of the app – off screen. ***Figure 1.1***
2. When the user swipes from the left to the right, the left panel slides to the right – into view. ***Figure 1.2 above***
3. When the panel is visible and the user swipes from right to left, the panel slides back to its original position out of view to the left.

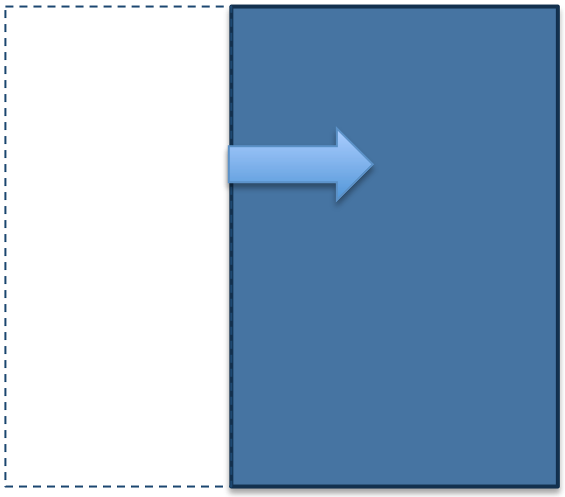
*Figure 1.1*

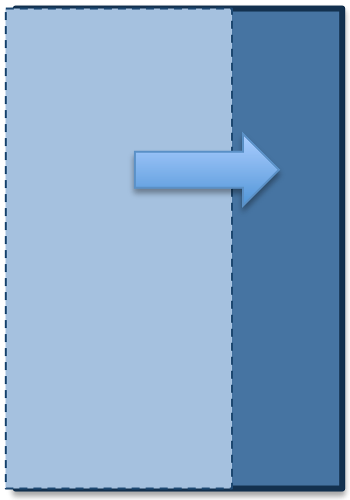
*Figure 1.2*

### Intended Audience

The intended audience for this document is web developers who are developing Amazon WebView apps using either the Amazon Mobile App SDK or Amazon WebView API SDK. The information in this document supplements the side panel guide:
*https://developer.amazon.com/appsandservices/solutions/devices/fire-phone/docs/implementing-sidepanellayout*

Because the side panel developer guide targets Android developers, not all of the information in that guide is relevant to web app developers.

## Detect and manage gesture interactions

This section focuses on detecting and interacting with various types touch gestures in your web application:

- Swipe
- Single tap
- Touch and hold
- Double tap

Note that side panels can also be controlled with one-handed shortcuts similarly to the touch gestures described in this document. For information on using one-handed shortcuts in your app, please review the *"Using One-Handed Shortcuts"* document, in addition to using this guide.

### Swipe

This section describes how to detect a simple swipe for displaying side panels.

To transition the panels from outside of the screen area to inside the screen, first detect a swipe event that begins on the right or left margin and moves inward toward the center of the screen. Use the following determinations:

- Margin size = 10px

  The margin is the target starting coordinate for touchstart events. On the left side this will be between 0px and 10px , and on the right side, between window.offsetWidth and window.offsetWidth minus 10px.

- Minimum length of the swipe = 100px

  To be a valid swipe, the horizontal distance between the touchstart and touchend event must be at least 100px.

- Maximum time allowed for a swipe = 300ms

  The time between the touchstart and touchend events cannot be longer than 300ms for a swipe.

Once you set your parameters, handle the touch events to determine whether an event is a swipe or not:

1. Listen for a touchstart event:
   - Determine if the touch was inside one of the margins.

    - Store the X coordinate.
    - Store the current timestamp.
2. Listen for the touchend event:
    - Calculate the distance between the touchstart & touchend events, which should be greater than the minimum swipe length (100px).
    - Calculate the time between the start of the touchstart & touchend events, which should be less than the max swipe time (300ms).

If the event satisfies all of your parameters, the event is a swipe.

## Single Tap

App users frequently select an item by tapping a menu item or link. You can add support for taps in your web application can by listening for the touchstart, touchmove and touchend events and setting a flag to see if touchmove is invoked. The following sample code shows this approach:

```
var isMoving;

function handleTouchstart(e) {
    isMoving = false;
}

function handleTouchmove(e) {
    isMoving = true;
}

function handleTouchend(e) {
    if(!isMoving) {
        //handle tap
    }
}
```

As an alternative, you could listen for just the touchstart and touchend events, and based on the starting and ending position of the gesture, determine whether the touch has moved.

## Touch and Hold

Detecting a touch and hold event is similar to detecting a single tap. However, for a touch and hold event, you will need to calculate the time difference between the touchstart and touchend events to determine if the gesture is tap or a touch and hold. Get the timestamp using the `Date.now()` API, which returns a timestamp in milliseconds.

Use the following sample code to detect a touch and hold event:

```
var isMoving, startTime;

function handleTouchstart(e) {
    isMoving  = false;
    startTime = performance.now();
}

function handleTouchmove(e) {
    isMoving = true;
}

function handleTouchend(e) {
    if(!isMoving) {
        var lapsTime = startTime – performance.now();
        var gesture = lapsTime <= 300 ? “tap” : “touchhold”;
    }
}
```

## Double Tap

Detecting a double tap gesture begins with detecting a first tap. Once you detect a single tap, set a flag to indicate that you detected an initial tap. Next set a timeout to see if you get another tap within 300ms:

```
var tapTimeout;
var lastGesture;

function handleTouchend(e) {
    var now     = performance.now();
    lastGesture = (lastGesture || now);
    var delta   = now - lastGesture;

    clearTimeout(tapTimeout);

    if(delta < 250 && delta > 0) {
        //double tap
    } else {
        lastGesture = now;
        tapTimeout = setTimeout(function(e) {
            //tap
        }, 250);
    }
    lastGesture = now;
}
```

## Sample App Walkthrough

*The complete source code for this example can be found in Amazon WebView API SDK in the "cordova/samples/MotionGestureManager" directory.*

The included sample is a simple sketchpad app that lets a user draw on the screen. The app employs the use of several touch gestures, such as tap, touchmove, and swipe. Swipe helps to illustrate more complex interaction, although this example only focuses on the use of swipe for handling panels.

### Swipe Gesture

This section describes how to detect a swipe gesture, and use that touch event to display a panel:

1. Detect a swipe that begins on the left or right side of the screen, within the defined margin. The sample's margin is 10px on each side from 0 to 10px on the left, and on the right it will be the width of the window minus 10px from the width of the window.
   Verify that the touchstart began in one of your margins, make sure that the event happens within the maximum time allowed for a swipe, in this case, 500ms.

   Test for this time value by invoking a setTimeout method at your maximum swipe time. If you get a touchend event before the maximum time, the setTimeout will be canceled, so you will test your event data for a swipe. If the setTimeout event fires, cancel your test for swipe and let the application continue with other events. For example, drawing a series of lines.
   When the touchend event fires, confirm whether the gesture was a swipe.

You will already know whether you are within our time limit for swipe because of the setTimeout event. So the last thing you need to test is the distance between the start of the swipe and the end. Expect the swipe to be greater than 100px along the X-axis.

The following markup snippet has an element with the id of "panel_left". Test for a swipe using the `touchstart` and `touchend` events and will modify the CSS of this element to show and hide the panel.

```
<style>
  .box_panelLeft {
      position: absolute;
      top     : 0px;
      width   : 640px;
      height  : 100%;
      z-index : 20;
      left    : -400px;
      transition:all 0.5s ease-in-out;
      -webkit-transition:all 0.5s ease-in-out;
  }

  .act_slideLeft {
    -webkit-transform : translateX(320px);
  }

<body>
    <div id="panel_left" <div>
    <!—Application Body -- >
</body>
```

Register listeners for touchstart and touchend events.

```
//Add listener for touch event
document.addEventListener("touchstart", handleEventStart);
document.addEventListener("touchend", handleEventEnd);
```

## `handleEventStart()`

The handleEventStart method determines if you have a touch or a click event. If you have a touch event, call the method that will test to see if you have the beginning of an edge-swipe:

```
/**
 * Touch & Mouse start event handler
 * @param {Event} e
 */
function handleEventStart(e) {

    if(e.targetTouches){
        //first see if we are starting a swipe
        detectBeginSwipe(e);
    }

} //end handleEventStart
```

## `detectSwipeMargin()`

This method will check for single touch gesture, then grab the start position's X-coordinate of the touch from the event data. If the start position is within your left or right margin, start the timeout that will test whether the gesture happened within the allowable time limit.

```
/**
 * See if we are within one of our side margins (right or left only)
 * if we are then we put our drawing on hold and open up test for swipe
 * @param {Event} e
 */
function detectSwipeMargin(e) {

    //make sure it's a single touch
    if(e.touches.length >=2) {
        return;
    };

    startX = e.touches[0].screenX;

    //If we are within a margin start our timer
    if(startX <= TOUCH_LEFT_MAX_X || startX >= TOUCH_RIGHT_MIN_X) {
        swipeTimer = setTimeout(function() {
           //the user is still moving
           //so this is not a swipe
        }, TOUCH_SWIPE_MAXTIME);
    }
} //end detectSwipeMargin
```

## `handleEventEnd()`

Clear the timeout and call the method that tests the event data for a swipe.

```
/**
 * Touch & Mouse end event handler
 * @param {Event} e
 */
function handleEventEnd(e) {
    clearTimeout(swipeTimer);

    //see where we ended up and check for swipe
    if(e.changedTouches) {
        var dir = swipeGesture(e.changedTouches[0].screenX);

       if(dir) {
           slidePanel(dir);
       }
    }

} //end handleEventEnd
```

## `swipeGesture()`

The swipeGesture method tests the direction of the gesture (left to right or right to left), and determines if that gesture is longer than the minimum swipe distance (100px). If you have a swipe gesture, the method will return the direction of the swipe.

```
/**
 * Check to see if we have a synthetic swipe
 * event and if so which direction are we going?
 * @param {Number} curX X-coordinate
 * @return {String}
 */
function swipeGesture(curX) {
    var swipeDir = getSwipeDirection(curX);
    var swipeDistance = getSwipeDistance(curX);

    if(swipeSpace >= TOUCH_SWIPE_THRESHOLD) {
        return swipeDir;
    } else {
        return undefined;
    }
}
```

## slidePanel()

The slidePanel method tests whether we have a panel currently visible, and makes the call to show or hide the appropriate panel based on the direction of the swipe.

The curPanel variable is a global variable that maintains the current state of the panels. When either the left or right panel is visible, this variable keeps track of which one is displayed. If no panel is currently visible, this variable will be undefined.

To hide the panel, swipe in the opposite direction.

```
/**
 * Decide if we need to show or hide a panel
 * @param {String} dir the direction of the tilt or swipe
 */
 function slidePanel (dir) {
    if(!curPanel) { //show panel

        //swap the direction because the opposite
        //direction displays the panel
        dir === "left" ? dir = "right" : dir = "left";

       //apply the CSS style to slide
        togglePanel(dir);

    } else if(curPanel === dir) { //hide panel

        //apply the CSS style to slide
        togglePanel(dir);
    }
 } //end slidePanel
```

## togglePanel()

The `togglePanel` method toggles the class that shows and hides the panel, and then sets the "curPanel" variable appropriately.

```
/**
 * Apply/Remove class that shows and hides the panels
 * @param {String} whichPanel the right or left panel
 */
function togglePanel(whichPanel) {
    whichPanel = whichPanel || curPanel;

    document.getElementById("panel_"
whichPanel).classList.toggle("act_slide"+whichPanel);

    //reset our curPanel variable
    if(curPanel) {
        curPanel = undefined;
    } else {
        curPanel = whichPanel;
    }
} //end togglePanel
```

## Additional Resources

A number of JavaScript libraries are available to handle touch gestures and increase the response time of touch gestures in mobile web applications. For example, FastClick is a library aimed at reducing the 300ms delay on click events in mobile browsers. The example source code does not use click events but touchstart and touchend events instead and then based on test criteria determines which kind of event we have (i.e. tap, swipe etc).

For more information on FastClick, search for “FastClick” on the github web site.