Intel® Atom™ E3800 Processor Series: Android* Multi-Display Features in Automobiles

Programming White Paper

Document Number: 331489-001

Legal Disclaimer

Contents

Figures

Revision History

Date	Revision	Description
October 2014	001	Alpha Release.

§

1 Executive Summary

Newer cars equipped with several video displays are changing the automotive experience for passengers.

Display technology and interactive features, powered by Android*, are rapidly transforming life on the road.

Intel Inside® powered, Android*-centric multi-display technology leverages Intel® Atom™ processor E3800 Series processors.

Android* vehicles equipped with one default display in the front dashboard, in combination with two additional displays for rear seat passengers, will deliver a fantastic vehicle infotainment system user experience.

Intel Inside® innovation can drive this landmark automobile sea change.

2 Introduction

In Android*-enabled automobiles, the Android*-based IVI system usually has one default main display, and projects the video signal content to additional HDMI video displays through cable or Miracast*.

But, in the recent past in IVI, it’s becoming more and more common to display different applications on different video displays.

In a typical setup, the Navigation application displays Map and/or GPS content on the front display for the driver, and a video player application displays movies on the rear display.

Intel® Atom™ processor E3800 Series processors have two separate display conduits that can attach to any of the display interfaces, since the SoC supports multi-display.

The Intel® Atom™ processor E3800 Series processor leaves multi-display feature development to the upper-layer development environment.

Multi-display feature development leverages Android Kitkat-4.4.2*.

In this white paper, the development of these features are demonstrated through the use of an Intel® Atom™ processor Z3000 series reference platform, with a cable connection to an HDMI display, and one Bluetooth* mouse to control the applications on the HDMI display.

The Intel® Atom™ Z3000 and E3800 Series processors have the same graphics engine. Thus, it is not a problem to port this feature to the Intel® Atom™ processor E3800 Series and future tick-tock SoC generations.

3 Multi-Display with Two Screen Support

Requisite knowledge of two key computing elements is crucial to the development and support of multiple displays with two screens:

1. Understanding the Android* design for external display support. This design resides in WMS/AMS/DMS/Input Subsystem/Surface Flinger.

2. Understanding how to develop the multi-display features that enable rendering, and understanding how to engage the ability to control different applications, simultaneously, based on current Android* design standards.

3.1 Multi-Display Android* Support

Android* has evolved to support multiple external displays and touchscreen technology and usage.

For the development of new features, Android* utilizes WMS/AMS interactions.

The current WMS/AMS interaction architecture has evolved considerably from earlier versions.

The interaction architecture is currently comprised of (in a one-to-one correspondence):

- StackBox, for WMS
- ActivityStack, for AMS

The Activity launched in external displays must be assigned to the target display.

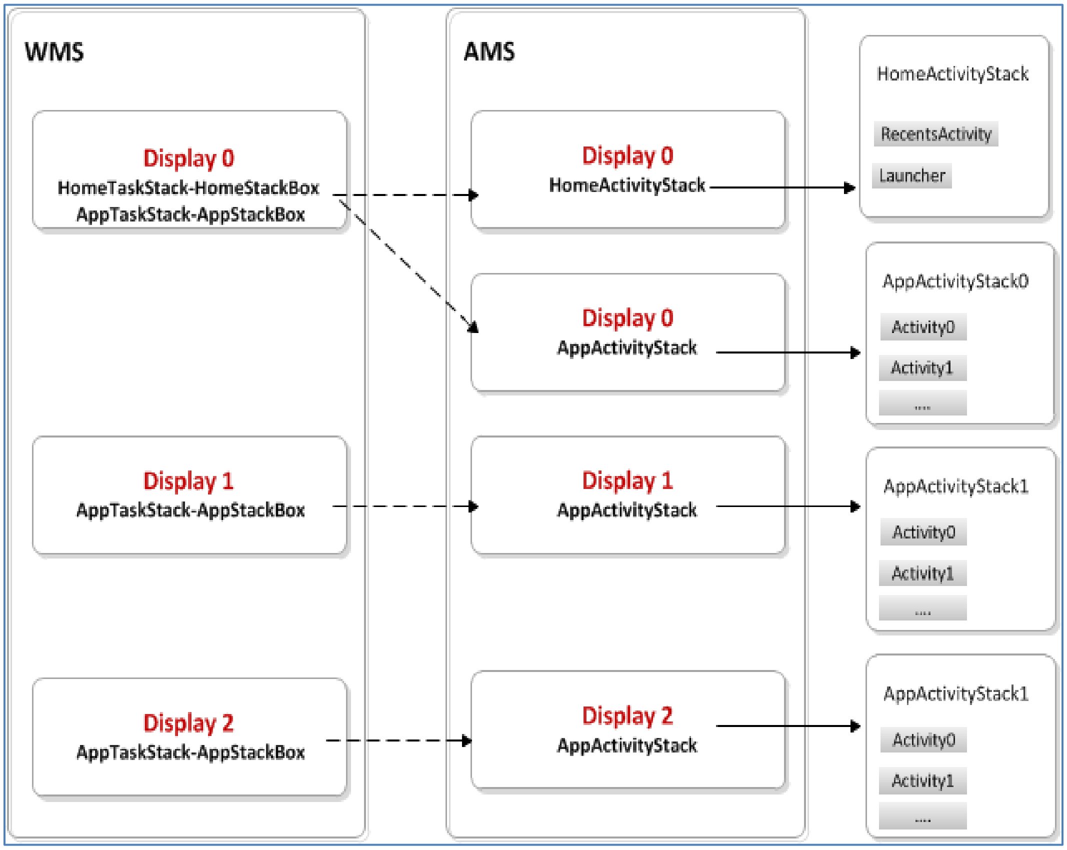
This can set views for activity rendering on the target display.

WindowState from one Activity can get the DisplayId, and then be assigned to the target display.

New Activity stacks from AMS can be maintained for external displays. For example, for external display, one activity stack should be created for it, to manage applications running on it. This is one ActivityStack for one display.

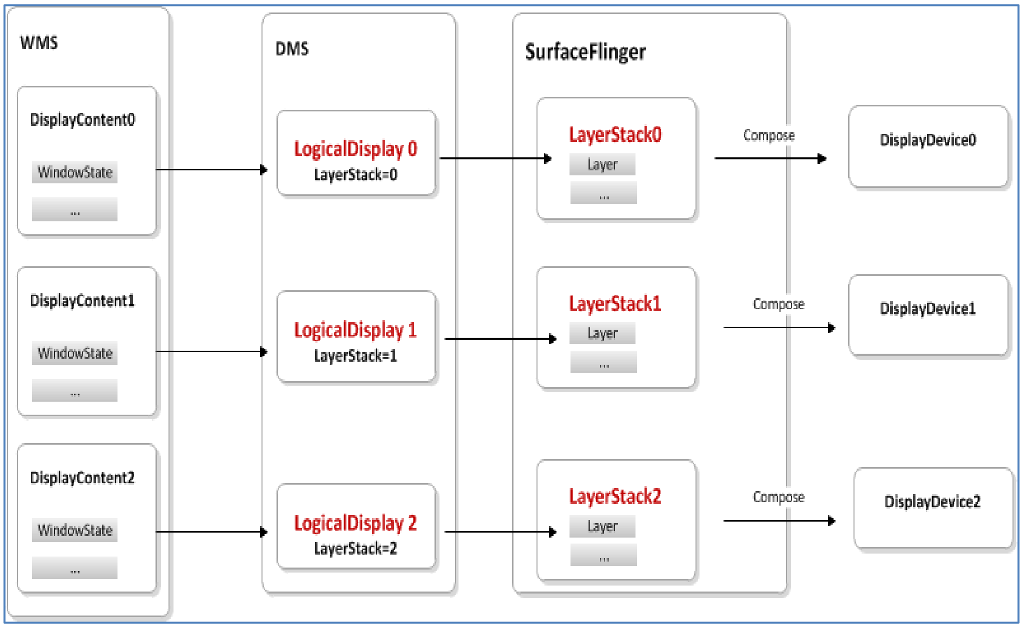
Figure 1 shows the WMS/DMS Surface Flinger Interaction.

Figure 1. WMS/AMS Interaction



WMS/DMS/SurfaceFlinger interaction allows the WMS views assigned to different displays to render on real physical display devices. **Figure 2** shows the WMS/DMS Surface Flinger Interaction.

Figure 2. WMS/DMS Surface Flinger Interaction



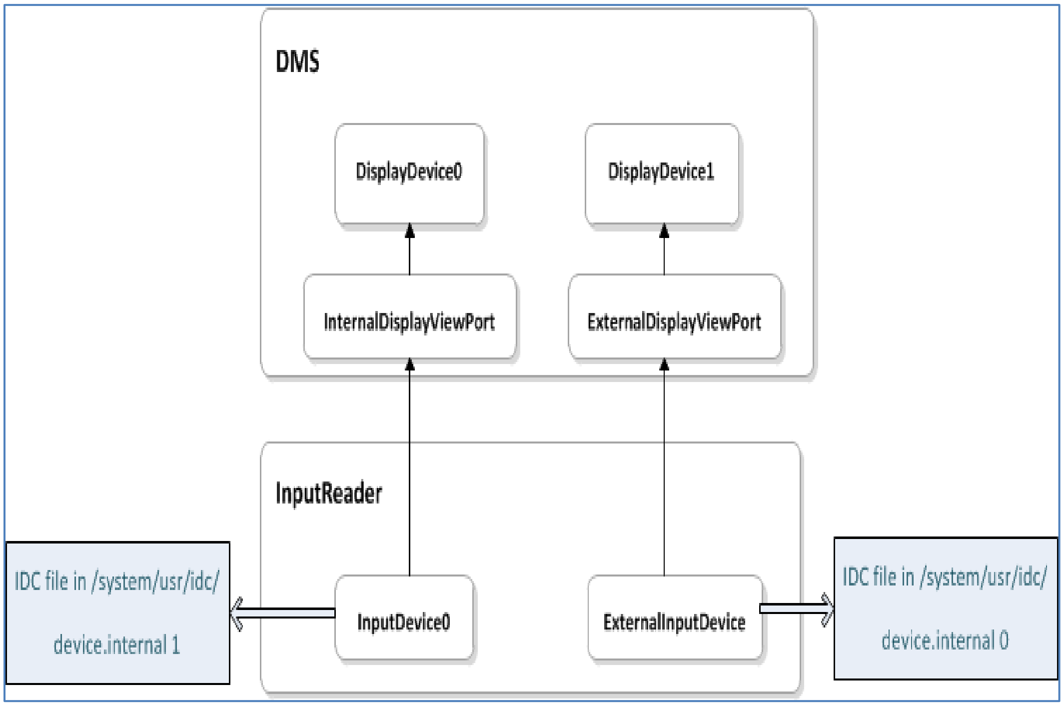
SurfaceFlinger composes Layers with the same LayerStack, and then delivers buffers with different LayerStacks into the correct display device. One displayId equals one LayerStack for one display device.

For two LayerStacks to co-exist, DisplayViewPort is employed. For a device with internal touch, internalDisplayViewPort is utilized. For an input device on external display, externalDisplayViewPort is utilized.

For views only assigned to the default display, the HDMI-only display depicts the same content as the default display. That is, only one LogicalDisplay0 exist. Create a new LogicalDisplay for the display device to mad HDMI display different views.

For the touch device on the external display, the Input Device Configuration (IDC) file must have the appropriate setting, as shown in **Figure 3**.

Figure 3. Input Device Configuration File

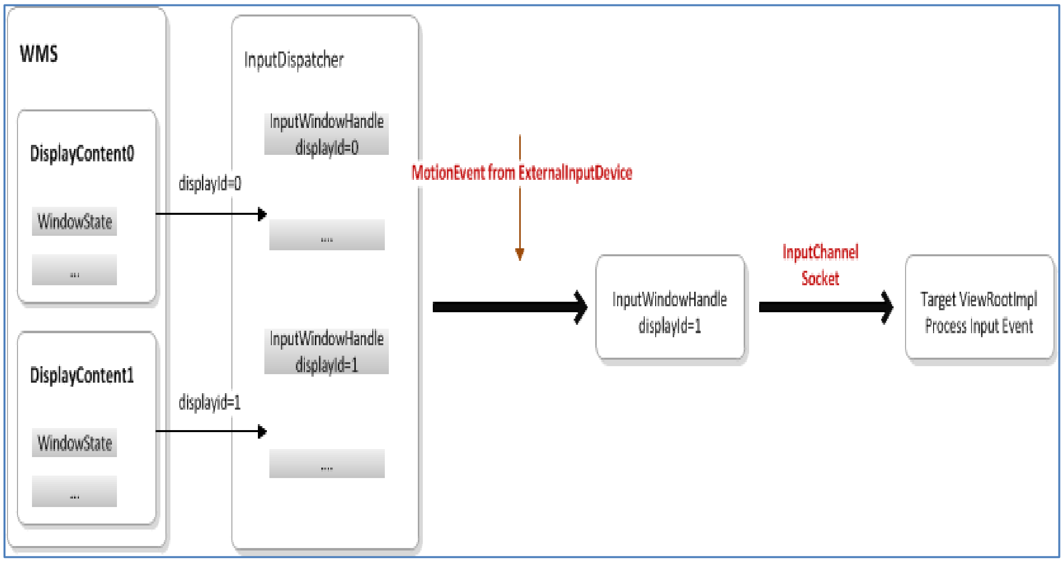


The motion events sent from the external touch device are assigned to the display device to make the views handle them.

Note: Android* Kitkat-4.4.2 only supports two touch devices.

Figure 4 illustrates the current, default Android* design. This Android* Framework will be improved in the near future to implement multi-display rendering of different applications.

Figure 4 Android* Design Framework



The current default Android* design consists of the following:

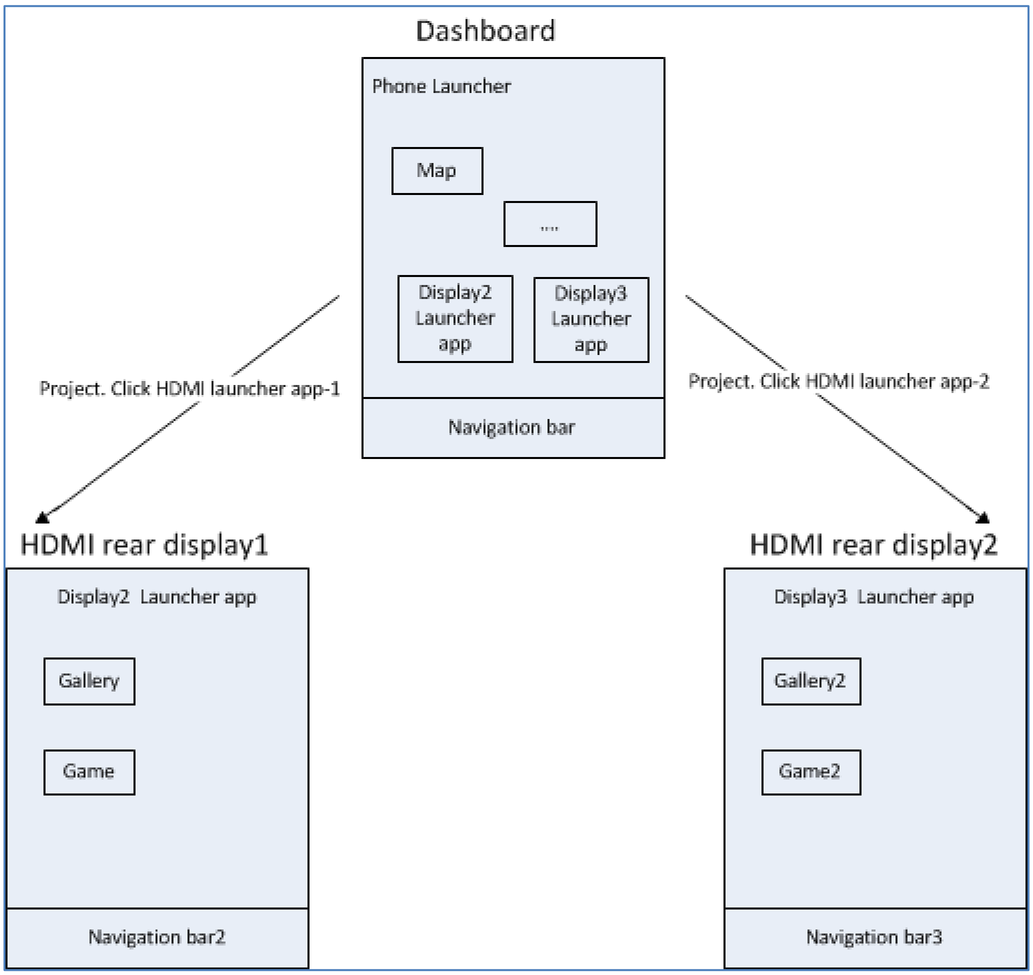
- One application ActivityStack for each display, to control application switching
- Support for a third touch device, including a third DisplayViewPort, DisplayDevice
- One focus window per display device, per touch device
- A Navigation Bar for each display, to enable applications to connect to the home screen

3.1.1 Proof of Concept for Two Displays

Figure 5 illustrates the multi-display scenario usage in an Android enabled vehicle. The illustration depicts the following:

- Launcher Application for each display
- Navigation Bar for each display

Figure 5. Multi-Display Usage Scenario in Android*-enabled Vehicles



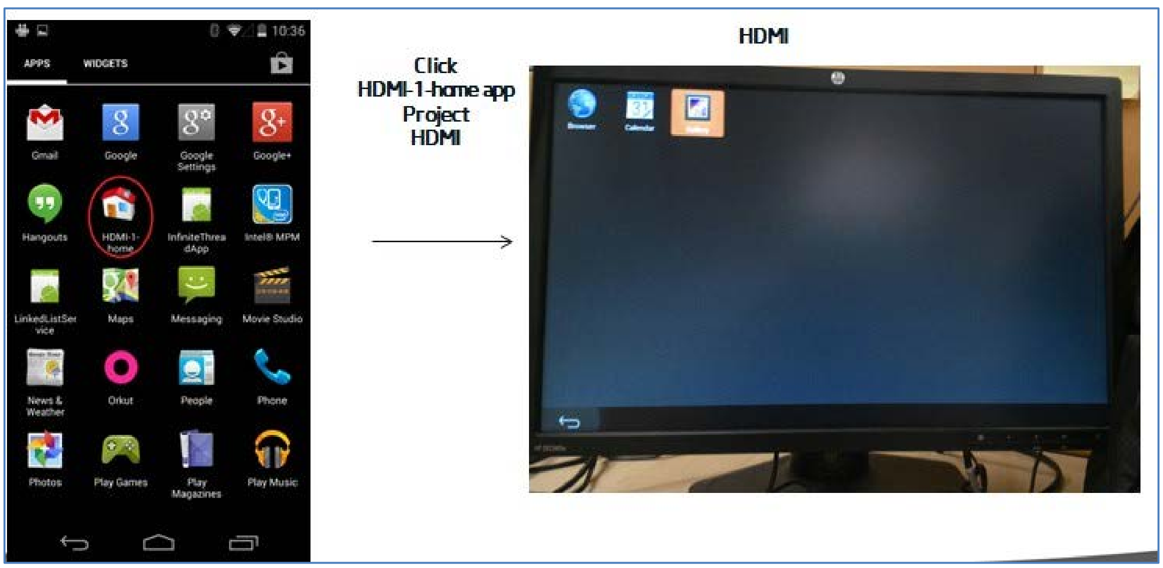
4 Results

The following are key results of the development information detailed in the previous sections.

- `HDMI-1-launcher.apk` is the launcher application for the second display. The Bluetooth* mouse can be used to control the HDMI Display.
- The Back key is used for the HDMI Display, to help the application return to the home screen

The current demo uses BYT_T_FFRD8, with a cable connection to the HDMI Display and a Bluetooth* mouse to control the application on the HDMI Display.

Figure 6. HDMI Display



After clicking the HDMI-1-Home application, the HDMI Display reflects this selection, and the default display is not changed.

The driver can use applications on the Dashboard display.

Rear passengers can launch any one of the three applications.