Using multi-window feature as differentiation on Android*

Overview
Multi-window is a feature in the Android* OS that can differentiate your apps. Many OEM and ODMs such as Samsung, Ramos, and Huawei use the feature to promote their products and make them stand out from the rest. In this article, we will introduce the multi-window function and show you how to implement it in your apps.

Figure 1. Multi-window use cases

Introduction
In June, 2012, the first open source multi-tasking framework, called Cornerstone, was developed. In August 2012, Samsung launched the first multi-window commercial product. And from 2013 until now, there has been an explosion of multi-window solutions on the market (see Figure 2).
The two types of multi-window styles are: floating style and docked style. The multi-window feature usually includes the open/close, resize, and swap functions. The open/close function starts/stops the feature. The resize function allows users to change the window sizes. The swap function exchanges the window positions.

In 2013, many solutions, developed by OEM/ODMs, by ISVs, and by the open-source community, emerged on the market. The following table compares the different multi-window solutions.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Cornerstone</th>
<th>Standout</th>
<th>Xposed</th>
<th>Tieto</th>
</tr>
</thead>
</table>

Figure 2. Multi-window evolution

Figure 3. Multi-window window styles
<table>
<thead>
<tr>
<th>Description</th>
<th>Multi-tasking framework for the Android* OS</th>
<th>An open source library that can be used to create floating apps</th>
<th>A multi-window application that supports docked windows style</th>
<th>The project aims to create desktop-like user experience</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open/close, resize, maximize</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
</tr>
<tr>
<td>Windows style</td>
<td>Docked</td>
<td>Floating</td>
<td>Docked</td>
<td>Docked/Floating</td>
</tr>
<tr>
<td>Code modification</td>
<td>Android framework</td>
<td>Application layer</td>
<td>Android framework</td>
<td>Android Framework</td>
</tr>
<tr>
<td>Application support</td>
<td>Support all applications, but SurfaceView cannot dynamically be adjusted</td>
<td>Some assistant applications, for example, calculator, etc.</td>
<td>Application compatibility and stability need improvement</td>
<td>Support all applications</td>
</tr>
<tr>
<td>Android version</td>
<td>Android 4.1 ~Android 4.4</td>
<td>Android 4.1~Android 4.4</td>
<td>Android 4.4</td>
<td>Android 4.4</td>
</tr>
</tbody>
</table>

**Software Architecture**

You can modify the Android framework code to adapt more functions. The Android OS architecture is divided into layers.

For Android 4.2 and Android 4.3, the launcher and other applications are all run on one stack, called “main stack”. As we know, multi-window needs more stacks to contain the multiple windows, so we need to modify the class ActivityManagerService of the framework to add stack creation and stack management interface. To modify the class WindowManagerService of the framework to adapt the view, we need to modify the inputManager of the framework to dispatch the touch event to the corresponding windows.

Stack management was changed significantly when Android 4.4 and Android 5.0 were released. The launcher and other applications can run on different stacks. Stacks and stack management functions were added. Below shows the stack differences between Android revisions.
Let’s focus on Android 5, codenamed Lollipop. As we know, the Android OS uses a callback method to trigger the activity interface function. But the main function is realized on the framework, so we will introduce two important classes ActivityManagerService and WindowManagerService.
Lollipop Activity Management
Because the multi-window feature depends on the stack, the following shows how to create a stack and how to start an activity on a stack.

In Lollipop, the IActivityManager.java added the following interface functions.

Table 1. Lollipop source code changes

<table>
<thead>
<tr>
<th>IactivityManager.java new add interface function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void moveTaskToStack(int taskId, int stackId, boolean toTop)</td>
<td>Move task to another stack</td>
</tr>
<tr>
<td>public void resizeStack(int stackBoxId, Rect bounds)</td>
<td>Resize the stack size</td>
</tr>
<tr>
<td>public void setFocusedStack(int stackId)</td>
<td>Set the current focus stack</td>
</tr>
<tr>
<td>Public Boolean isInHomeStack(int taskId)</td>
<td>get the task whether is in the HomeStack</td>
</tr>
</tbody>
</table>

After the startup, the SystemServer process will launch the activity management services and window management services. We can add the RuntimeException sentences to trace the process.

![Progress of stack creation in Lollipop](image)

Figure 6. Progress of stack creation in Lollipop

Now let’s show how to start an activity on a stack.
In Lollipop, adb (android debug bridge) added the following commands.

**Table 3. Lollipop’s new adb commands**

<table>
<thead>
<tr>
<th>ADB COMMANDS</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adb shell am stack start</td>
<td>Start a new activity on &lt;DISPLAY_ID&gt; using Intent</td>
<td>Kitkat 4.4, adb commands contain adb shell am stack create. Lollipop 5.0, adb commands adb shell am stack create deletion</td>
</tr>
<tr>
<td>Adb shell am stack movetask</td>
<td>Move &lt;TASK_ID&gt; from its current stack to the top or bottom of STACK_ID</td>
<td>Usage: adb shell am stack movetask task_id stackid true/false Notice: it is ok on KitKat, but it doesn’t work on Lollipop.</td>
</tr>
<tr>
<td>Adb shell am stack resize</td>
<td>Change &lt;STACK_ID&gt; size and position to &lt;LEFT, TOP, RIGHT, BOTTOM&gt;</td>
<td>Usage: adb shell am stack resize task_id weight</td>
</tr>
</tbody>
</table>
Lollipop window management

WindowManagerService is the core window manager. Its function contains input event dispatching, screen layout, and surface management.

Figure 8. WindowManagerService role in the graphics architecture

2
Some common questions on multi-window

Multi-window has a resize function. We have seen cases where a gaming animation can’t be resized. The root cause of this is that the Android function SurfaceFlinger can’t dynamically adjust the size.

![Figure 9](image)

Figure 9. *Games using SurfaceFlinger can’t dynamically adjust the windows size*

Another issue is that some applications don’t display correctly in the multi-window. An example below shows the calculator not displaying correctly in the multi-window because the application uses a bad compatibility configuration.

![Figure 10](image)

Figure 10. *Calculator with bad configuration*
**Will Android’s next version support multi-window?**

Will Google release the multi-window feature in its next version of the OS. I found the following log information in the Lollipop source code. We use the following command to search the multi-window log.

```
git log --grep “multiwindow”
```

The log content contains a line that reads “defer tap outside stack until multiwindows”. So we conclude that multi-window may be on Google’s roadmap.

```
commit ed7698c9ad6e6b959d723ca47bfe01530d306fb8d
Author: Craig Mautner <cmautner@google.com>
Date:   Tue Jan 27 11:44:59 2015 -0800

    Defer tap outside stack until multiwindows

    Taps outside of the stack boundary were causing the current app to lose focus. This led to timeouts waiting for the app to respond.

    Disabling the tap recognition keeps the focus from changing. It will be reenabled for multiwindows.

    Fixes bug 17721767.

    Change-Id: Iad854b19979390e73b97299e489ad7ef82e03d47
```

*Figure 11. Lollipop log about multi-window*

**Case study: Cornerstone**

Onskreen developed Cornerstone, the first multi-window framework solution. It is for large screen devices and tablets. You can download the source code from GitHub³. It is only supported on the Android 4.1 and Android 4.2 versions. It not released yet on higher Android versions. But we can analyze the source code of Android 4.2 to get more technical details.
Figure 12. Cornerstone modifications on Jelly Bean
Summary
Many mobile devices now use Intel® processors running the Android OS. How can developers improve the user experience? How can their products best compete? These questions push us to constantly improve our products on Intel® Architecture (IA) devices. Multi-window is a good differentiating feature. It is convenient and allows consumers to do two or more things simultaneously. They can watch a video and IM friends with video feedback. They can play a game and read reviews of it. Several devices now support the multi-window feature: Ramos i12 tablet, Teclast x98 tablet, and Cube i7, which uses the Remix OS.

Figure 13. Multi-window feature on IA devices

Reference

Resource

About the Author
Li Liang earned a Master’s degree in signal and information processing from Changchun University of Technology. He joined Intel in 2013 as an application engineer in the Developer Relations Division CCE (Client Computing Enabling). He focuses on helping developers to differentiate their apps on the Android platform.
Notices

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

The products and services described may contain defects or errors known as errata which may cause deviations from published specifications. Current characterized errata are available on request.

Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or by visiting www.intel.com/design/literature.htm.

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others

© 2015 Intel Corporation.