This document is an extension of “Guideline for resolving frequent errors when porting apps to Android 4.0”. please refer to the following link:
Guideline for resolving frequent errors when porting apps to Android 4.0

1. Out Of Memory Exceptions

A. Case

Certain applications that functioned correctly in Android GB (Ginger Bread) are causing out of memory (OutOfMemory) exceptions in ICS (Ice Cream Sandwich).

B. Guideline

Android has a limited heap size, which differs for each device. The heap size limits of each device can be checked using the code snippet below.

```
// Returns the memory heap limit size in MB.
int memoryClass = ((ActivityManager)
getSystemService(Context.ACTIVITY_SERVICE)).getMemoryClass();
```

Android attempts to reclaim memory that is no longer in use by the application when the system is low on memory (the developer cannot free memory manually). Out of memory exceptions occur when the system attempts to allocate memory when there is no available free memory, even after garbage collection. Due to the change in the heap memory size of the VM in ICS, out of memory exceptions occur in certain applications that functioned correctly in GB.

To resolve this issue, the application must be revised to use a smaller heap memory, or you must secure more available heap memory through the largeHeap settings.

e.g. 1) An out of memory exception occurs due to an error when an application uses the BitmapFactory class.
The solutions below may be used to resolve the issue.

- •Add recycle() when using a BitmapFactory to release memory.
- •When the image size is large, resize the image to be smaller and revise the application to use less memory.
- •Use supplementary code using a try/catch for out of memory exceptions that occur when using an Android BitmapFactory.
- •Check the maximum image processing size through an image sample test and reflect the maximum size in the application.

e.g. 2) Use the largeHeap option. In Android SDK 3.0 HoneyComb or later,
manifest settings can be adjusted to use larger heap memory than the basic size provided by the device.
- Add the’ largerHeap=True’ value in the application tag of the AndroidManifest.xml file.

```
<application
    ....
    largerHeap = "true">
```

Reference: Google I/O 2011: Memory management for Android Apps
http://www.google.com/events/io/2011/sessions/memory-management-for-android-apps.html

2. Scrolling Speed Issues in ListView

A. Case

Certain applications that functioned correctly in Android GB show slow response speeds or abnormal termination when scrolling in the ListView provided in ICS.

B. Guideline

UI code, such as creating a layout, rendering a view system, or outputting resources through the ListView within an Android application, may result in low performance due to frequent allocation and releasing of internal memory. In most cases that show performance drops with a ListView, the heap memory size increases dramatically within a few seconds after scrolling in a ListView starts.

Repeated creation of objects (drawables, bitmaps, etc.) in the getView method of the adapter causes garbage collection, which results in severe drops in performance and response speeds.

e.g.) When the app fails to recycle 368,656 bytes (720x128x4) occupied by bitmaps,
which make up each row in the ListView, and repeatedly creates new ones, android.widget.
LinearLayout ends up exceeding the maximum heap size (64MB) by creating hundreds of
LinearLayout objects.
This causes a lack of memory and frequently calls garbage collector.

Garbage collection caused by an out of memory exception during list scrolling is the main reason for the performance drop. The same guidelines as in section 1 are also applicable.
Furthermore, optimizing memory usage in the ListView used in the application may improve performance.

Many applications usually override the getView method as shown below.

```
 LayoutInflater mInflater;
      ...
      public View getView(int position, View convertView, ViewGroup parent)
      {
            convertView = mInflater.inflate(R.layout.list_row, null);

            TextView tvData = (TextView) convertView.findViewById(R.id.list_row_tv_data);
            ImageView ivPhoto = (ImageView)
convertView.findViewById(R.id.list_row_iv_photo);

            ...
            // Data Setting

            return convertView;
      }
```

However, the above method creates a new view for each position, and causes performance drops every time a new view is created.
To resolve this problem, the application must be revised to reuse the previously created view instead of repeatedly creating new views; ViewHolder can be used to do this.

```
LayoutInflater mInflater;

...
class ViewHolder
{
        TextView tvData;
        ImageView ivPhoto;
}
...
public View getView(int position, View convertView, ViewGroup parent)
{
        ViewHolder holder;

        if(convertView == null)
        {
                holder = new ViewHolder;
                convertView = mInflater.inflate(R.layout.list_row, null);
                holder.tvData = (TextView)
convertView.findViewById(R.id.list_row_tv_data);
                        holder.ivPhoto = (ImageView)
convertView.findViewById(R.id.list_row_iv_photo);
                        convertView.setTag(holder);
        }
        else
        {
                holder = (ViewHolder) convertView.getTag();
        }       ...
        // Data Setting

        return convertView;
}
```

The above method reuses the views in the adapter. If a view was already created using the above method, it retrieves the child view using the views that have already been created, replaces the data and returns it. Compared to the aforementioned method, this can result in twice the performance improvement and prevents new views from being repeatedly created.

> Reference: Google I/O 2010: The world of ListView
> http://www.google.com/intl/ko-KR/events/io/2010/sessions/world-of-listview-android.html

3. Screen Layout of WebApps Being Displayed Incorrectly

A. Case

Certain applications using a WebView were displayed correctly in Android GB, but do not properly display layouts in pop-up windows and some other elements in ICS.

B. Guideline

Android supports web features to enable applications to include web contents using the WebView class. It also supports the viewport property, which allows the size of the web apps to be adjusted according to size of the device's screen. The Webkit framework allows web pages to specify viewport properties and allows using the screen density for modifying styles or image resources.

Furthermore, Android's WebView implements the wide viewport concept, which adjusts the viewport size in mobile environments.

```
// Resize contents to fit the screen.
   webview.getSettings().setuseWideViewPort( true );
   webview.setInitialScale( 1 );
```

To make sure that the contents are displayed correctly in all screen densities, viewport metadata can be written and read. CSS and Javascript can be used to provide an option for various screen densities as shown below. Here, the viewport's visible area can be adjusted by the developer.

```
<meta name="viewport"
      content="
           height = [pixel_value | device-height] ,
           width = [pixel_value | device-width ] ,
           initial-scale = float_value ,
           minimum-scale = float_value ,
           maximum-scale = float_value ,
           user-scalable = [yes | no] ,
           target-densitydpi = [dpi_value | device-dpi |
                                      high-dpi | medium-dpi | low-dpi]"
/>
```

For example, the HTML code below sets the width of the viewport to exactly match the width of the screen, so the scaling property must be used.

```
<head>
    <title>Example</title>
    <meta name="viewport" content="width=device-width, user-scalable=no" />
</head>
```

When the screen of a web application is created with a wide viewport, the layout may display incorrectly in ICS if the Web contents do not specify the viewport properties.

Due to the difference in the logic of the wide viewport logic between GB and ICS, mobile pages must be written based on the device-width/1.0 scale/medium-DPI to create a WebView page that is compatible with regular Android devices.

Reference:

http://developer.android.com/guide/webapps/targeting.html
http://developer.android.com/reference/android/webkit/WebSettings.html#setUseWideViewPort(boolean)

4. Changing Input Method Selections in Edit Fields

A. Case

Input method selection has changed in ICS. The pop-up for the input method selection that appears when the Text Edit field was long-pressed in GB has been moved to another position in ICS. Therefore, the input selection method must be revised in some IME-related applications (software keyboard applications) for ICS.

B. Guideline

In GB and previous versions, the input method selection pop-up (input method picker) appeared upon a long press in the text edit field, and an input method could be chosen from this pop-up. But in ICS, changing the input method from the text edit field was changed to selecting the input method from the Notification Bar after pulling it down from the top of the screen.

IME-related applications in GB attempt to launch the input method selection pop-up when the user long-presses on a text edit field. As the ICS method of input method selection differs significantly, the two methods are incompatible with each other.

5. Minor Changes in SQLite Keywords in ICS

A. Case

Some SQLite keywords allowed in GB cause SQLiteExceptions.

B. Guideline

When creating a SQL query using the getContentResolver().query(...) method and using GROUP BY to group data, a SQLiteException may occur in ICS. The following code requires android.permission.READ_CONTACTS.

```
String request = Phone.NUMBER + " LIKE ? OR " + Email.DATA + " LIKE ?";
String[] params = {"%test%", "%test%"};

Cursor cursor = getContentResolver().query( Data.CONTENT_URI,
            new String[] { Data._ID, Data.RAW_CONTACT_ID },
                           request + ") GROUP BY (" + Data.RAW_CONTACT_ID,  params,
                                                   "lower(" + Data.DISPLAY_NAME + ")
ASC");
```

To resolve this problem, you must use rawQuery instead of “GROUP BY”, or you must use a workaround, such as MatrixCursor.

```
Cursor cursor = getContentResolver().query( Data.CONTENT_URI,
                                  new String[] { Data._ID, Data.RAW_CONTACT_ID }, null,
null, null);
MatrixCursor result = new MatrixCursor(new String[] { Data._ID, Data.RAW_CONTACT_ID });
Set seen = new HashSet();
while (cursor.moveToNext())
{
        long raw = cursor.getLong(1);
```

```
        if (!seen.contains(raw))
       {
          seen.add(raw);
          result.addRow(new Object[] {cursor.getLong(0), raw});
       }
}
```

In versions after Android Honeycomb, FTS (Full Text Searching) has been supported in SQLite to find specific words in large data or long text. FTS supports quickly finding names, emails, phone numbers, organization names, long memo data, and addresses from information stored in Contacts, and has a large impact on updates and query performance.

Reference:

http://www.sqlite.org/fts3.html

6. NullPointerException in Specific Networks

A. Case

Certain applications that functioned correctly in GB are causing NullPointerExceptions on certain devices with LTE-support or Wi-Fi only devices in ICS.

B. Guideline

Generally, when a java.lang NullPointerException occurs, it is one of the following.

1. Calling the instance method of a null object.
2. Accessing or modifying the field of a null object.
3. Taking the length of null as if it were an array.
4. Accessing or modifying the slots of null as if it were an array.
5. Throwing null as if it were a Throwable value.

To resolve this problem, the developer must check that the object values are correctly referenced, if the received value is Null, and must except the subject code if it is.

```
try { …. }
catch ( NullPointerException ex )  { …. }
```

As Wi-Fi-only devices and devices with LTE and HSPA+ support are increasingly being launched in the market, certain applications may not work properly and throw a NullpointerException when launched on a new device that supports a new type of network. Consider the following:

1) To support LTE, applications that use the following code are not supported in previous versions.

```
NetworkInfo lte_4g = manager.getNetworkInfo(ConnectivityManager.TYPE_WIMAX);
```

2) When the state of the SIM card must be checked:

```
TelephonyManager tm = (TelephonyManager) getSystemService(Context.TELEPHONY_SERVICE);


       if (tm.getSimState() == TelephonyManager.SIM_STATE_ABSENT)
       {
          // When SIM card is missing
```

```
      }
      else
      {
         // When SIM card is present
      }
```

3) When the connection status, such as 3G and Wi-Fi on/off, must be identified:

```
- 3G
         ConnectivityManager conn = (ConnectivityManager)
context.getSystemService(Context.CONNECTIVITY_SERVICE);
         NetworkInfo ni = conn.getNetworkInfo(ConnectivityManager.TYPE_MOBILE);
         return ni.isConnected() || ni.isConnectedOrConnecting();

-wifi
         WifiManager wifiManager = (WifiManager) context.getSystemService
Context.WIFI_SERVICE);
         wifiManager.isWifiEnabled();
```

Furthermore, LTE was added in Android versions 2.3.x and later. Various other network types, such as HSPA+, were added in 4.0 (ICS), and a wider range of network types can be identified.

```
package com.AndroidTelephonyManager;

TelephonyManager tm = (TelephonyManager) getSystemService(Context.TELEPHONY_SERVICE);
int type = tm.getNetworkType();

switch(type)
{
….
case TelephonyManager.NETWORK_TYPE_HSDPA:
         return  “HSPA+”;
case TelephonyManager.NETWORK_TYPE_LTE:
         return  "NETWORK_TYPE_LTE";
….
}
```

Reference:

http://developer.android.com/reference/android/telephony/TelephonyManager.html#NETWORK_TYPE_LTE
http://developer.android.com/reference/android/telephony/TelephonyManager.html#getSimState()

7. Force Quit Within the Main Thread and Applying Strict Mode

A. Case

Certain apps that access files or the network in the main thread are being forced to quit in ICS.

B. Guideline

ICS implements a strong policy where an application is forced to quit when it accesses a file or the network from the main thread. Using the permitAll() method in Strict Mode can prevent the application from being forced to quit. However, this is only a temporary workaround.

Android provides Strict Mode for developers to easily locate and revise code that violates the thread protection policy. The developer must apply Strict Mode in the application to find the parts that violate the policy, and revise the logic. Strict Mode enables detecting and locating code where the application accesses a file or the network in a specific thread. It can also detect the memory location and provides logging, force quit, DropBox, dialogue, and splash notifications for memory leaks.

Reference:

http://developer.android.com/reference/android/os/StrictMode.htmlhttp://android-developers.blogspot.com/2010/12/new-gingerbread-api-strictmode.html

ref:http://developer.samsung.com/android/technical-docs/Guideline-for-resolving-frequent-errors-when-porting-apps-to-Android-4-0-2