ntroduction

Taking advantage of device orientation (landscape vs. portrait) is one way to enhance the user experience of your app. You need to consider how your app might look and behave differently depending on the orientation of the device. For example, some mail programs provide a list in portrait, and a list on the left side with a preview of the email messages on the right side. You may also wish to lock the orientation so that your app does not rotate along with the device.

When a device is rotated during runtime, or when new peripherals are added such as a keyboard, it is a configuration change. Normally the running Activity restarts whenever there is a configuration change.

On Activity restart, the onDestroy() method is called followed by the onCreate() method. These methods are responsible for destroying the existing Activity and creating a new one. This default behavior helps make it suitable for an application to adapt to the new screen orientation, as well as other configuration changes.

Sometimes this behavior should be overridden in order to provide optimal performance.

Handling Orientation Programmatically

Orientation changes by default force the Activity to lose its current state and restart. To save the state of the application, Android calls the OnSaveInstanceState(Bundle bundle) method before it destroys the Activity. The bundle object is passed in the OnSaveInstanceState() callback method to save small amount of data. Bundle is not designed to handle large amount of data. Restoring the saved state is done with the onCreate() or onRestoreInstanceState () method.

Restoring the complete activity state may not be possible if there is large amount of data to be restored as a Bundle. Handling large amount of data with the Bundle may restart the Activity. There are two ways to bypass Activity re-initialization:

1. Override the onRetainNonConfigurationInstance () method to return the stateful object.
2. When the activity is created after orientation/configuration change, call getLastNonConfigurationInstance () to recover object.

The following code bypasses the activity destruction process:

```
…
@Override
public Object onRetainNonConfigurationInstance() {
    //restore all your data here
    final MyResources data = collectMyResourceData();
    return data;
}
…
…
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
```

```
    setContentView(R.layout.main);

    final MyResources data = (MyResources) getLastNonConfigurationInstance();
    if (data == null) {
        data = collectMyResourceData ();
    }
}
...
   private MyResources collectMyResourceData(){
...
            MyResources myResource = new MyResources();
        ...
   return myResource;
   }
```

[Code 1]

Handling Orientation Itself

If the app's resources do not need to be updated after an orientation change, or if you prefer to control configuration changes yourself, then you can program the Activity to handle its own configuration changes.

To give the Activity control over its own configuration changes , edit the <activity> [for more details visit <activity> field of the manifest file by including the android: configChanges attributes:

```
<activity
        android:name="MyActivity"
android:configChanges = "orientation|keyboard"  <!—multiple values are separated by “|”
character -->
        android:label="@string/app_name">
</activity>
```

[Code 2]

These attributes prevent the Activity from being restarted and on subsequent device rotation onConfigurationChanged() method of activity gets called. The onConfigurationChanged() method passes the configuration object that specifies new device configuration.

```
@Override
public void onConfigurationChanged(Configuration newConfig) {
    super.onConfigurationChanged(newConfig);

    // Checks the orientation of the screen
    if (newConfig.orientation == Configuration.ORIENTATION_LANDSCAPE) {
        Toast.makeText(this, "landscape", Toast.LENGTH_SHORT).show();
    } else if (newConfig.orientation == Configuration.ORIENTATION_PORTRAIT){
```

```
        Toast.makeText(this, "portrait", Toast.LENGTH_SHORT).show();
    }
}
```

[Code 3]

It is not mandatory to implement the onConfigurationChanged() method even if the android: configChanges attributes are added in activity field.

To update the resource object upon an orientation chage, such as updating the ImageView with new image resources, implement onConfigurationChanged().

Lock Screen Orientation Using Manifest File

To block Android Activity from rotating on screen orientation change, add android: screenOrientation ("portrait" or "landscape") field into the <activity> element,

```
<activity
        android:name=".MyActivity"
        android:label="@string/app_name"
        android:screenOrientation="portrait">    <!--activity follow only portrait
                                                                        mode-->
</activity>
```

[Code 4]

Lock Screen Orientation Programmatically

If the event can be evoked by the system or by the user, then the screen orientation can be locked programmatically. The following code snippet shows how to lock phone screen orientation:

```
...
@Override
public void onConfigurationChanged(Configuration newConfig) {
    super.onConfigurationChanged(newConfig);
        switch (newConfig.orientation)
        {
        case Configuration.ORIENTATION_PORTRAIT:
                // taking action on event
                lockScreenRotation(Configuration.ORIENTATION_PORTRAIT);
        break;
        case Configuration.ORIENTATION_LANDSCAPE:
                // taking action on event
                lockScreenRotation(Configuration.ORIENTATION_LANDSCAPE);
        break;
        case Configuration.ORIENTATION_SQUARE:
                // taking action on event
                lockScreenRotation(Configuration.ORIENTATION_SQUARE);
```

```
        break;
        default:
                throw new Exception("Unexpected orientation!!!");
        break;
 }


private void lockScreenRotation(int orientation)
{
  // Stop the screen orientation changing during an event
    switch (orientation)
    {
         ...
  case Configuration.ORIENTATION_PORTRAIT:
    this.setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
    break;
  case Configuration.ORIENTATION_LANDSCAPE:
    this.setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
    break;
         ...
  case default:
   this.setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_UNSPECIFIED);
   break;
    }
}
```

[Code 5]

Orientation on Sensor

To change the screen orientation automatically based on the position, add the android:screenOrientation = "sensor" attribute in the manifest file as follows:

```
...
<activity android:name=".MyActivity"
android:screenOrientation="sensor"
android:label="@string/app_name">
...
```

[Code 6]

Orientation of Android Emulator

To test your app’s behavior on different orientations and screen sizes on the Android Emulator, use the following keys:

To switch to the previous layout orientation (for example, portrait, landscape) - KEYPAD_7 or Ctrl-F11

To switch to next layout orientation (for example, portrait, landscape) KEYPAD_9 or Ctrl-F12

If you prefer, to use KEYPAD_7 or KEYPAD_9 then turn off Num Lock.

[Figure 1: Portrait Mode]

[Figure 2: Landscape Mode]

ref:http://developer.samsung.com/android/technical-docs/Handling-Orientation-in-Android