# Handling Runtime Changes

Some device configurations can change during runtime (such as screen orientation, keyboard availability, and language). When such a change occurs, Android restarts the running [Activity](file:///G:\Studio\SDK\docs\reference\android\app\Activity.html) ([onDestroy()](file:///G:\Studio\SDK\docs\reference\android\app\Activity.html#onDestroy()) is called, followed by [onCreate()](file:///G:\Studio\SDK\docs\reference\android\app\Activity.html#onCreate(android.os.Bundle))). The restart behavior is designed to help your application adapt to new configurations by automatically reloading your application with alternative resources that match the new device configuration.

To properly handle a restart, it is important that your activity restores its previous state through the normal[Activity lifecycle](file:///G:\Studio\SDK\docs\guide\components\activities.html#Lifecycle), in which Android calls[onSaveInstanceState()](file:///G:\Studio\SDK\docs\reference\android\app\Activity.html#onSaveInstanceState(android.os.Bundle)) before it destroys your activity so that you can save data about the application state. You can then restore the state during[onCreate()](file:///G:\Studio\SDK\docs\reference\android\app\Activity.html#onCreate(android.os.Bundle)) or [onRestoreInstanceState()](file:///G:\Studio\SDK\docs\reference\android\app\Activity.html#onRestoreInstanceState(android.os.Bundle)).

To test that your application restarts itself with the application state intact, you should invoke configuration changes (such as changing the screen orientation) while performing various tasks in your application. Your application should be able to restart at any time without loss of user data or state in order to handle events such as configuration changes or when the user receives an incoming phone call and then returns to your application much later after your application process may have been destroyed. To learn how you can restore your activity state, read about the [Activity lifecycle](file:///G:\Studio\SDK\docs\guide\components\activities.html#Lifecycle).

However, you might encounter a situation in which restarting your application and restoring significant amounts of data can be costly and create a poor user experience. In such a situation, you have two other options:

1. [Retain an object during a configuration change](file:///G:\Studio\SDK\docs\guide\topics\resources\runtime-changes.html#RetainingAnObject)

Allow your activity to restart when a configuration changes, but carry a stateful object to the new instance of your activity.

1. [Handle the configuration change yourself](file:///G:\Studio\SDK\docs\guide\topics\resources\runtime-changes.html#HandlingTheChange)

Prevent the system from restarting your activity during certain configuration changes, but receive a callback when the configurations do change, so that you can manually update your activity as necessary.

Retaining an Object During a Configuration Change

If restarting your activity requires that you recover large sets of data, re-establish a network connection, or perform other intensive operations, then a full restart due to a configuration change might be a slow user experience. Also, it might not be possible for you to completely restore your activity state with the [Bundle](file:///G:\Studio\SDK\docs\reference\android\os\Bundle.html) that the system saves for you with the [onSaveInstanceState()](file:///G:\Studio\SDK\docs\reference\android\app\Activity.html#onSaveInstanceState(android.os.Bundle))callback—it is not designed to carry large objects (such as bitmaps) and the data within it must be serialized then deserialized, which can consume a lot of memory and make the configuration change slow. In such a situation, you can alleviate the burden of reinitializing your activity by retaining a [Fragment](file:///G:\Studio\SDK\docs\reference\android\app\Fragment.html) when your activity is restarted due to a configuration change. This fragment can contain references to stateful objects that you want to retain.

When the Android system shuts down your activity due to a configuration change, the fragments of your activity that you have marked to retain are not destroyed. You can add such fragments to your activity to preserve stateful objects.

To retain stateful objects in a fragment during a runtime configuration change:

1. Extend the [Fragment](file:///G:\Studio\SDK\docs\reference\android\app\Fragment.html) class and declare references to your stateful objects.
2. Call [setRetainInstance(boolean)](file:///G:\Studio\SDK\docs\reference\android\app\Fragment.html#setRetainInstance(boolean)) when the fragment is created.
3. Add the fragment to your activity.
4. Use [FragmentManager](file:///G:\Studio\SDK\docs\reference\android\app\FragmentManager.html) to retrieve the fragment when the activity is restarted.

For example, define your fragment as follows:

public class RetainedFragment extends Fragment {  
  
    // data object we want to retain  
    private MyDataObject data;  
  
    // this method is only called once for this fragment  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        // retain this fragment  
        setRetainInstance(true);  
    }  
  
    public void setData(MyDataObject data) {  
        this.data = data;  
    }  
  
    public MyDataObject getData() {  
        return data;  
    }  
}

**Caution:** While you can store any object, you should never pass an object that is tied to the[Activity](file:///G:\Studio\SDK\docs\reference\android\app\Activity.html), such as a [Drawable](file:///G:\Studio\SDK\docs\reference\android\graphics\drawable\Drawable.html), an [Adapter](file:///G:\Studio\SDK\docs\reference\android\widget\Adapter.html), a [View](file:///G:\Studio\SDK\docs\reference\android\view\View.html) or any other object that's associated with a [Context](file:///G:\Studio\SDK\docs\reference\android\content\Context.html). If you do, it will leak all the views and resources of the original activity instance. (Leaking resources means that your application maintains a hold on them and they cannot be garbage-collected, so lots of memory can be lost.)

Then use [FragmentManager](file:///G:\Studio\SDK\docs\reference\android\app\FragmentManager.html) to add the fragment to the activity. You can obtain the data object from the fragment when the activity starts again during runtime configuration changes. For example, define your activity as follows:

public class MyActivity extends Activity {  
  
    private RetainedFragment dataFragment;  
  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.main);  
  
        // find the retained fragment on activity restarts  
        FragmentManager fm = getFragmentManager();  
        dataFragment = (DataFragment) fm.findFragmentByTag(“data”);  
  
        // create the fragment and data the first time  
        if (dataFragment == null) {  
            // add the fragment  
            dataFragment = new DataFragment();  
            fm.beginTransaction().add(dataFragment, “data”).commit();  
            // load the data from the web  
            dataFragment.setData(loadMyData());  
        }  
  
        // the data is available in dataFragment.getData()  
        ...  
    }  
  
    @Override  
    public void onDestroy() {  
        super.onDestroy();  
        // store the data in the fragment  
        dataFragment.setData(collectMyLoadedData());  
    }  
}

In this example, [onCreate()](file:///G:\Studio\SDK\docs\reference\android\app\Activity.html#onCreate(android.os.Bundle)) adds a fragment or restores a reference to it. [onCreate()](file:///G:\Studio\SDK\docs\reference\android\app\Activity.html#onCreate(android.os.Bundle)) also stores the stateful object inside the fragment instance. [onDestroy()](file:///G:\Studio\SDK\docs\reference\android\app\Activity.html#onDestroy()) updates the stateful object inside the retained fragment instance.

Handling the Configuration Change Yourself

If your application doesn't need to update resources during a specific configuration change *and* you have a performance limitation that requires you to avoid the activity restart, then you can declare that your activity handles the configuration change itself, which prevents the system from restarting your activity.

**Note:** Handling the configuration change yourself can make it much more difficult to use alternative resources, because the system does not automatically apply them for you. This technique should be considered a last resort when you must avoid restarts due to a configuration change and is not recommended for most applications.

To declare that your activity handles a configuration change, edit the appropriate [<activity>](file:///G:\Studio\SDK\docs\guide\topics\manifest\activity-element.html)element in your manifest file to include the [android:configChanges](file:///G:\Studio\SDK\docs\guide\topics\manifest\activity-element.html#config) attribute with a value that represents the configuration you want to handle. Possible values are listed in the documentation for the [android:configChanges](file:///G:\Studio\SDK\docs\guide\topics\manifest\activity-element.html#config) attribute (the most commonly used values are "orientation" to prevent restarts when the screen orientation changes and "keyboardHidden" to prevent restarts when the keyboard availability changes). You can declare multiple configuration values in the attribute by separating them with a pipe | character.

For example, the following manifest code declares an activity that handles both the screen orientation change and keyboard availability change:

<activity android:name=".MyActivity"  
          android:configChanges="orientation|keyboardHidden"  
          android:label="@string/app\_name">

Now, when one of these configurations change, MyActivity does not restart. Instead, theMyActivity receives a call to [onConfigurationChanged()](file:///G:\Studio\SDK\docs\reference\android\app\Activity.html#onConfigurationChanged(android.content.res.Configuration)). This method is passed a [Configuration](file:///G:\Studio\SDK\docs\reference\android\content\res\Configuration.html)object that specifies the new device configuration. By reading fields in the [Configuration](file:///G:\Studio\SDK\docs\reference\android\content\res\Configuration.html), you can determine the new configuration and make appropriate changes by updating the resources used in your interface. At the time this method is called, your activity's [Resources](file:///G:\Studio\SDK\docs\reference\android\content\res\Resources.html) object is updated to return resources based on the new configuration, so you can easily reset elements of your UI without the system restarting your activity.

**Caution:** Beginning with Android 3.2 (API level 13), **the "screen size" also changes** when the device switches between portrait and landscape orientation. Thus, if you want to prevent runtime restarts due to orientation change when developing for API level 13 or higher (as declared by the [minSdkVersion](file:///G:\Studio\SDK\docs\guide\topics\manifest\uses-sdk-element.html#min) and [targetSdkVersion](file:///G:\Studio\SDK\docs\guide\topics\manifest\uses-sdk-element.html#target) attributes), you must include the "screenSize" value in addition to the "orientation" value. That is, you must decalareandroid:configChanges="orientation|screenSize". However, if your application targets API level 12 or lower, then your activity always handles this configuration change itself (this configuration change does not restart your activity, even when running on an Android 3.2 or higher device).

For example, the following [onConfigurationChanged()](file:///G:\Studio\SDK\docs\reference\android\app\Activity.html#onConfigurationChanged(android.content.res.Configuration)) implementation checks the current device orientation:

@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();  
    }  
}

The [Configuration](file:///G:\Studio\SDK\docs\reference\android\content\res\Configuration.html) object represents all of the current configurations, not just the ones that have changed. Most of the time, you won't care exactly how the configuration has changed and can simply re-assign all your resources that provide alternatives to the configuration that you're handling. For example, because the [Resources](file:///G:\Studio\SDK\docs\reference\android\content\res\Resources.html) object is now updated, you can reset any [ImageView](file:///G:\Studio\SDK\docs\reference\android\widget\ImageView.html)s with[setImageResource()](file:///G:\Studio\SDK\docs\reference\android\widget\ImageView.html#setImageResource(int)) and the appropriate resource for the new configuration is used (as described in[Providing Resources](file:///G:\Studio\SDK\docs\guide\topics\resources\providing-resources.html#AlternateResources)).

Notice that the values from the [Configuration](file:///G:\Studio\SDK\docs\reference\android\content\res\Configuration.html) fields are integers that are matched to specific constants from the [Configuration](file:///G:\Studio\SDK\docs\reference\android\content\res\Configuration.html) class. For documentation about which constants to use with each field, refer to the appropriate field in the [Configuration](file:///G:\Studio\SDK\docs\reference\android\content\res\Configuration.html) reference.

**Remember:** When you declare your activity to handle a configuration change, you are responsible for resetting any elements for which you provide alternatives. If you declare your activity to handle the orientation change and have images that should change between landscape and portrait, you must re-assign each resource to each element during [onConfigurationChanged()](file:///G:\Studio\SDK\docs\reference\android\app\Activity.html#onConfigurationChanged(android.content.res.Configuration)).

If you don't need to update your application based on these configuration changes, you can instead*not* implement [onConfigurationChanged()](file:///G:\Studio\SDK\docs\reference\android\app\Activity.html#onConfigurationChanged(android.content.res.Configuration)). In which case, all of the resources used before the configuration change are still used and you've only avoided the restart of your activity. However, your application should always be able to shutdown and restart with its previous state intact, so you should not consider this technique an escape from retaining your state during normal activity lifecycle. Not only because there are other configuration changes that you cannot prevent from restarting your application, but also because you should handle events such as when the user leaves your application and it gets destroyed before the user returns to it.

For more about which configuration changes you can handle in your activity, see the[android:configChanges](file:///G:\Studio\SDK\docs\guide\topics\manifest\activity-element.html#config) documentation and the [Configuration](file:///G:\Studio\SDK\docs\reference\android\content\res\Configuration.html) class.