The Android Application Life Cycle

Unlike most traditional environments, Android applications have no control over their own life cycles. Instead, application components must listen for changes in the application state and react accordingly, taking particular care to be prepared for untimely termination.

As mentioned before, by default, each Android application is run in its own process that's running a separate instance of Dalvik. Memory and process management of each application is handled exclusively by the run time.

While uncommon, it's possible to force application components within the same application to run in different processes or to have multiple applications share the same process using the android:process *attribute on the affected component nodes within the manifest.*

Android aggressively manages its resources, doing whatever it takes to ensure that the device remains responsive. This means that processes (and their hosted applications) will be killed, without warning if necessary, to free resources for higher-priority applications — generally those that are interacting directly with the user at the time. The prioritization process is discussed in the next section.

Understanding Application Priority and Process States

The order in which processes are killed to reclaim resources is determined by the priority of the hosted applications. An application's priority is equal to its highest-priority component.

Where two applications have the same priority, the process that has been at a lower priority longest will be killed fi rst. Process priority is also affected by interprocess dependencies; if an application has a dependency on a Service or Content Provider supplied by a second application, the secondary application will have at least as high a priority as the application it supports.

All Android applications will remain running and in memory until the system needs its resources for other applications.

Figure 3-3 shows the priority tree used to determine the order of application termination.

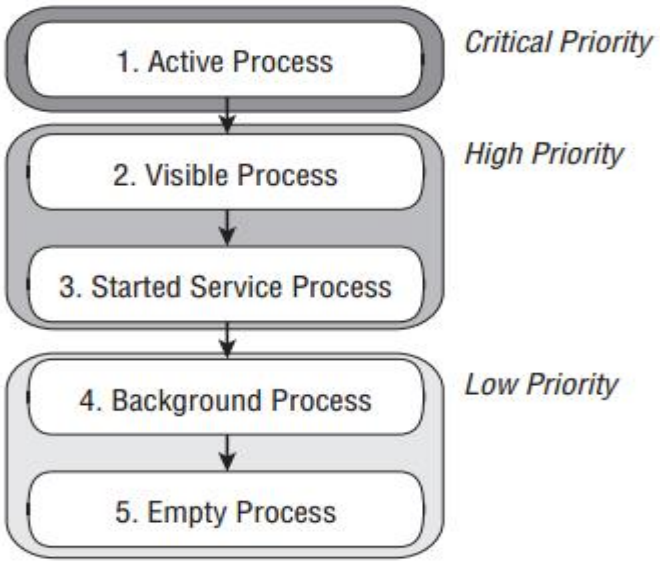

Figure 3-3

It's important to structure your application correctly to ensure that its priority is appropriate for thework it's doing. If you don't, your application could be killed while it's in the middle of something important.

The following list details each of the application states shown in Figure 3-3, explaining how the state is determined by the application components comprising it:

❑ **Active Processes** Active (foreground) processes are those hosting applications with components currently interacting with the user. These are the processes Android is trying to keep responsive by reclaiming resources. There are generally very few of these processes, and they will be killed only as a last resort.

Active processes include:

❑ Activities in an "active" state; that is, they are in the foreground and responding to user events. You will explore Activity states in greater detail later in this chapter.

❑ Activities, Services, or Broadcast Receivers that are currently executing an onReceive event handler.

❑ Services that are executing an onStart, onCreate, or onDestroy event handler.

❑ **Visible Processes** Visible, but inactive processes are those hosting "visible" Activities. As the name suggests, visible Activities are visible, but they aren't in the foreground or responding to user events. This happens when an Activity is only partially obscured (by a non-full-screen or transparent Activity). There are generally very few visible processes, and they'll only be killed in extreme circumstances to allow active processes to continue.

❑ **Started Service Processes** Processes hosting Services that have been started. Services support ongoing processing that should continue without a visible interface. Because Services don't interact directly with the user, they receive a slightly lower priority than visible Activities. They are still considered to be foreground processes and won't be killed unless resources are needed for active or visible processes. You'll learn more about Services in Chapter 8.

❑ **Background Processes** Processes hosting Activities that aren't visible and that don't have any Services that have been started are considered background processes. There will generally be a large number of background processes that Android will kill using a last-seen-fi rst-killed pattern to obtain resources for foreground processes.

❑ **Empty Processes** To improve overall system performance, Android often retains applications in memory after they have reached the end of their lifetimes. Android maintains this cache to improve the start-up time of applications when they're re-launched. These processes are routinely killed as required.

Externalizing Resources

No matter what your development environment, it's always good practice to keep non-code resources like images and string constants external to your code. Android supports the externalization of resources ranging from simple values such as strings and colors to more complex resources like images (drawables), animations, and themes. Perhaps the most powerful resources available for externalization are layouts.

By externalizing resources, they become easier to maintain, update, and manage. This also lets you easily defi ne alternative resource values to support different hardware and internationalization.

You'll see later in this section how Android dynamically selects resources from resource trees that let you defi ne alternative values based on a device's hardware confi guration, language, and location. This lets you create different resource values for specifi c languages, countries, screens, and keyboards. When an application starts, Android will automatically select the correct resource values without your having to write a line of code.

Among other things, this lets you change the layout based on the screen size and orientation and customize text prompts based on language and country.