Using Background Worker Threads

To ensure that your applications remain responsive, it's good practice to move all slow, time-consuming operations off the main application thread and onto a child thread.

All Android application components — including Activities, Services, and Broadcast Receivers — run on the main application thread. As a result, time-consuming processing in any component will block all other components including Services and the visible Activity.

Using background threads is vital to avoid the "Application Unresponsive" Dialog box described in Chapter 2. Unresponsiveness is defi ned in Android as Activities that don't respond to an input event (such as a key press) within 5 seconds and Broadcast Receivers that don't complete their onReceive handlers within 10 seconds.

Not only do you want to avoid this scenario, you don't want to even get close. Use background threads for all time-consuming processing, including fi le operations, network lookups, database transactions, and complex calculations.

Creating New Threads

You can create and manage child threads using Android's Handler class and the threading classes available within java.lang.Thread. The following skeleton code shows how to move processing onto a child thread:

```
// This method is called on the main GUI thread.
private void mainProcessing() {
// This moves the time consuming operation to a child thread.
Thread thread = new Thread(null, doBackgroundThreadProcessing, "Background");
thread.start();
}
// Runnable that executes the background processing method.
private Runnable doBackgroundThreadProcessing = new Runnable() {
public void run() {
backgroundThreadProcessing();
}
};
// Method which does some processing in the background.
private void backgroundThreadProcessing() {
[ ... Time consuming operations ... ]
}
```

Synchronizing Threads for GUI Operations

Whenever you're using background threads in a GUI environment, it's important to synchronize child threads with the main application (GUI) thread before creating or modifying graphical components.

The Handler class allows you to post methods onto the thread in which the Handler was created. Using the Handler class, you can post updates to the User Interface from a background thread using the Post method. The following example shows the outline for using the Handler to update the GUI thread:

```
// Initialize a handler on the main thread.
private Handler handler = new Handler();
private void mainProcessing() {
Thread thread = new Thread(null, doBackgroundThreadProcessing, "Background");
thread.start();
}
private Runnable doBackgroundThreadProcessing = new Runnable() {
public void run() {
backgroundThreadProcessing();
}
};
// Method which does some processing in the background.
private void backgroundThreadProcessing() {
[ ... Time consuming operations ... ]
handler.post(doUpdateGUI);
}
// Runnable that executes the update GUI method.
```

```
private Runnable doUpdateGUI = new Runnable() {
public void run() {
updateGUI();
}
};
private void updateGUI() {
[ ... Open a dialog or modify a GUI element ... ]
}
```

The Handler class lets you delay posts or execute them at a specifi c time, using the postDelayed and postAtTime methods, respectively.

In the specifi c case of actions that modify Views, the UIThreadUtilities class provides the runOnUIThread method, which lets you force a method to execute on the same thread as the specified View, Activity, or Dialog.

Within your application components, Notifi cations and Intents are always received and handled on the GUI thread. In all other cases, operations that explicitly interact with objects created on the GUI thread (such as Views) or that display messages (like Toasts) must be invoked on the main thread.