Saving and Loading Files

It's good practice to use Shared Preferences or a database to store your application data, but there are still times when you'll want to use fi les directly rather than rely on Android's managed mechanisms.

As well as the standard Java I/O classes and methods, Android offers openFileInput and openFileOuput to simplify reading and writing streams from and to local fi les, as shown in the code snippet below:

```
String FILE_NAME = "tempfile.tmp";
// Create a new output file stream that's private to this application.
FileOutputStream fos = openFileOutput(FILE_NAME, Context.MODE_PRIVATE);
// Create a new file input stream.
FileInputStream fis = openFileInput(FILE_NAME);
```

These methods only support fi les in the current application folder; specifying path separators will cause an exception to be thrown.

If the fi lename you specify when creating a FileOutputStream does not exist, Android will create it for you. The default behavior for existing fi les is to overwrite them; to append an existing fi le, specify the mode as Context.MODE_APPEND.

By default, fi les created using the openFileOutput method are private to the calling application — a different application that tries to access these fi les will be denied access. The standard way to share a fi le between applications is to use a Content Provider. Alternatively, you can specify either Context.MODE_WORLD_READABLE or Context.MODE_WORLD_WRITEABLE when creating the output file to make them available in other applications, as shown in the following snippet:

```
String OUTPUT_FILE = "publicCopy.txt";
FileOutputStream fos = openFileOutput(OUTPUT_FILE, Context.MODE_WORLD_WRITEABLE);
```

Including Static Files as Resources

If your application requires external fi le resources, you can include them in your distribution package by placing them in the res/raw folder of your project hierarchy.

To access these Read Only fi le resources, call the openRawResource method from your application's Resource object to receive an InputStream based on the specifi ed resource. Pass in the filename (without extension) as the variable name from the R.raw class, as shown in the skeleton code below:

```
Resources myResources = getResources();
InputStream myFile = myResources.openRawResource(R.raw.myfilename);
```

Adding raw fi les to your resources hierarchy is an excellent alternative for large, preexisting data sources (such as dictionaries) where it's not desirable (or even possible) to convert them into an Android database.

Android's resource mechanism lets you specify alternative resource fi les for different languages, locations, or hardware confi gurations. As a result, you could, for example, create an application that dynamically loads a dictionary resource based on the user's current settings.

File Management Tools

Android supplies some basic fi le management tools to help you deal with the filesystem. Many of these utilities are located within the standard java.io.File package. Complete coverage of Java fi le management utilities is beyond the scope of this book, but Android does supply some specialized utilities for fi le management available from the application's Context.

- **deleteFile** Lets you remove files created by the current application.
- **fileList** Returns a String array that includes all the fi les created by the current application.