Let's Make a Toast

Toasts are transient Dialog boxes that remain visible for only a few seconds before fading out. Toasts don't steal focus and are non-modal, so they don't interrupt the active application.

Toasts are perfect for informing your users of events without forcing them to open an Activity or read a Notifi cation. They provide an ideal mechanism for alerting users to events occurring in background Services without interrupting foreground applications.

The Toast class includes a static makeText method that creates a standard Toast display window. Pass the application Context, the text message to display, and the length of time to display it (LENGTH_SHORT or LENGTH_LONG) in to the makeText method to construct a new Toast. Once a Toast has been created, display it by calling show, as shown in the following snippet:

```
Context context = getApplicationContext();
String msg = "To health and happiness!";
int duration = Toast.LENGTH_SHORT;
Toast toast = Toast.makeText(context, msg, duration);
toast.show();
```

Figure 8-1 shows a Toast. It will remain on screen for around 2 seconds before fading out. The application behind it remains fully responsive and interactive while the Toast is visible.

Figure 8-1

Customizing Toasts

The standard Toast text message window is often suffi cient, but in many situations you'll want to customize its appearance and screen position. You can modify a Toast by setting its display position and assigning it alternative Views or layouts.

The following snippet shows how to align a Toast to the bottom of the screen using the setGravity method:

```
Context context = getApplicationContext();
String msg = "To the bride an groom!";
int duration = Toast.LENGTH_SHORT;
Toast toast = Toast.makeText(context, msg, duration);
int offsetX = 0;
int offsetY = 0;
toast.setGravity(Gravity.BOTTOM, offsetX, offsetY);
toast.show();
```

When a text message just isn't going to get the job done, you can specify a custom View or layout to use a more complex, or more visual, display. Using setView on a Toast object, you can specify any View (including layouts) to display using the transient message window mechanism.

For example, the following snippet assigns a layout, containing the CompassView widget from Chapter 4 along with a TextView, to be displayed as a Toast.

```
Context context = getApplicationContext();
String msg = “Cheers!”;
int duration = Toast.LENGTH_LONG;
Toast toast = Toast.makeText(context, msg, duration);
toast.setGravity(Gravity.TOP, 0, 0);
LinearLayout ll = new LinearLayout(context);
ll.setOrientation(LinearLayout.VERTICAL);
TextView myTextView = new TextView(context);
CompassView cv = new CompassView(context);
myTextView.setText(msg);
int lHeight = LinearLayout.LayoutParams.FILL_PARENT;
int lWidth = LinearLayout.LayoutParams.WRAP_CONTENT;
ll.addView(cv, new LinearLayout.LayoutParams(lHeight, lWidth));
ll.addView(myTextView, new LinearLayout.LayoutParams(lHeight, lWidth));
ll.setPadding(40, 50, 0, 50);
toast.setView(ll);
toast.show();
```

The resulting Toast will appear as shown in Figure 8-2.

Figure 8-2

Using Toasts in Worker Threads

As GUI components, Toasts must be opened on the GUI thread or risk throwing a cross thread exception. In the following example, a Handler is used to ensure that the Toast is opened on the GUI thread:

```
private void mainProcessing() {
Thread thread = new Thread(null, doBackgroundThreadProcessing, “Background”);
thread.start();
}
private Runnable doBackgroundThreadProcessing = new Runnable() {
public void run() {
backgroundThreadProcessing();
}
};
private void backgroundThreadProcessing() {
handler.post(doUpdateGUI);
} // Runnable that executes the update GUI method.
private Runnable doUpdateGUI = new Runnable() {
public void run() {
Context context = getApplicationContext();
String msg = “To open mobile development!”;
int duration = Toast.LENGTH_SHORT;
Toast.makeText(context, msg, duration).show();}};
```