Notifying the Users

CS443 – Mobile Applications
Instructor: Bo Sheng

Sending Notifications to the User

• A service can notify the user of events using Toast Notifications or Status Bar Notifications.

• Toast notification: a message that appears on the surface of the current window for a moment then disappears.

• Status bar notification: an icon in the status bar with a message, which the user can select to take an action.
  – Usually a status bar notification is the best when some background work has completed and the user can now act on it.

Notification UI

Notifications in the notification area
Notifications in the notification drawer

Notification Display Elements

• Notifications in the notification drawer can appear in one of two visual styles, depending on the version and the state of the drawer.

• Normal view
  – The standard view of the notifications.

• Big view
  – A large view that's visible when the notification is expanded.
  – Big view is part of the expanded notification feature available as of Android 4.1.

Normal View

A notification in normal view appears in an area that’s up to 64 dp tall.
1. Content title
2. Large icon
3. Content text
4. Content info
5. Small icon
6. Time that the notification was issued.

Big View

Available in Android 4.1 or higher.
1. Content title
2. Large icon
3. Content text
4. Content info
5. Small icon
6. Time that the notification was issued.
7. Details area
Create Toast Notifications

- A toast notification pops up on the surface of the window and automatically fades in and out. It does not accept interaction events.
- A toast can be created and displayed from an Activity or Service. If created from a Service, it appears in front of the Activity currently in focus.
- How to: Instantiate a Toast object with `makeText()` and display it with `show()`.

```
context = getApplicationContext();  // context
mText = this.getString(R.string.toast_text);  // mText
Toast toast = Toast.makeText(context, mText, Toast.LENGTH_SHORT);
toast.show();
```

- A standard toast notification appears near the bottom of the screen, centered horizontally. You can change this position with `setGravity(int, int, int)`:
  - A `Gravity` constant, an x-position offset, and a y-position offset.

Create a Status Bar Notification

- Create a Notification Builder
- Define the Notification's Action
- Set the Notification's Click Behavior
- Issue the Notification
- Notes
  - We use `NotificationCompat.Builder` class in the v4 Support Library for best compatibility.
  - Class `Notification.Builder` was added in Android 3.0.

Creating a Notification Builder

- Specify the UI content and actions with a NotificationCompat.Builder object.
- A Builder object must include the following:
  - A small icon, set by `setSmallIcon()`
  - A title, set by `setContentTitle()`
  - Detail text, set by `setContentText()`

```
NotificationCompat.Builder mBuilder = new NotificationCompat.Builder(this)
    .setSmallIcon(R.drawable.notification_icon)
    .setContentTitle("My notification")
    .setContentText("Hello World!");
```

Defining Notification's Action

- An action takes users directly to an Activity, where they can look at the event that caused the notification or do further work.
- Actions are optional, but you should add at least one action.
- The action is defined by a PendingIntent containing an Intent that starts an Activity in your app.
- Grant permission to a foreign application to use the contained intent
- When starting an Activity from a notification, you must preserve the user's expected navigation experience.

Setting Click Behavior

- Call the appropriate method of NotificationCompat.Builder to associate the PendingIntent with a click.
- E.g., to start an activity when the user clicks the notification text in the notification drawer, add the PendingIntent by calling `setContentIntent()`.

```
PendingIntent resultPendingIntent;
...  // resultPendingIntent;
```

An Example

```
Intent resultIntent = new Intent(this, ResultActivity.class);
TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);
// Adds the back stack
stackBuilder.addParentStack(ResultActivity.class);
// Adds the Intent to the top of the stack
stackBuilder.addNextIntent(resultIntent);
// Gets a PendingIntent containing the entire back stack
PendingIntent resultPendingIntent = stackBuilder.getPendingIntent(0, PendingIntent.FLAG_UPDATE_CURRENT);
...  // resultPendingIntent;
   .setContentIntent(resultPendingIntent);
```
Issuing the Notification

- Get an instance of `NotificationManager`.
- Call `mBuilder.build()`, which returns a `Notification` object containing your specifications.
- Call `notify()` to issue the notification.
  - When calling `notify()`, specify a notification ID.
  - Use this ID to update the notification if needed.

Updating Notifications

- To update an issued notification, update or create a `NotificationCompat.Builder` object, build a `Notification` object from it, and issue the `Notification` with the same ID.

Removing Notifications

- Notifications remain visible until one of the following happens:
  - The user dismisses the notification either individually or by using "Clear All" (if the notification can be cleared).
  - The user touches the notification, and `setAutoCancel()` was called in creating the notification.
  - Call `cancel()` for a specific notification ID. It also deletes ongoing notifications.
  - Call `cancelAll()`. It removes all of the notifications previously issued.

Notification Channels

- Starting in Android 8.0 (API level 26), all notifications must be assigned to a channel.

Create a notification channel

- Construct a `NotificationChannel` object with a unique channel ID, a user-visible name, and an importance level.
- Optionally, specify the description that the user sees in the system settings with `setDescription()`.
- Register the notification channel by passing it to `createNotificationChannel()`.

Displaying Progress in Notification

- Notifications can include an animated progress indicator.
- Use a progress bar
  - If you can estimate how long the operation takes and how much is complete.
- Use an activity indicator
  - If you can't estimate the length of the operation.
Displaying a Progress Bar

- Add the bar to the notification by calling `setProgress(max, progress, false)` and then issue the notification.
- The 3rd argument indicates whether the progress bar is indeterminate (true) or determinate (false).
- As the operation proceeds, increments `progress`, and updates the notification.
  - At the end of the operation, `progress` should equal `max`.
- Usually `max` is set to 100 and then `progress` is incremented as a "percent complete" value for the operation.
- To remove the progress bar, call `setProgress(0, 0, false)`.

```java
mNotifyManager = (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
mBuilder = new NotificationCompat.Builder(this);
mBuilder.setSmallIcon(R.drawable.ic_notification);
```

```java
new Thread(new Runnable()
{
    @Override
    public void run()
    {
        int incr;
        for(incr = 0; incr <= 100; incr+=5)
        {
            mBuilder.setProgress(100, incr, false);
            mNotifyManager.notify(0, mBuilder.build());
            try
            {
                Thread.sleep(5*1000);
            }
            catch(InterruptedException e)
            {
                Log.d(TAG, "sleep	failure");
            }
        }
    }
}).start();
```

// When the loop is finished, updates the notification
```
mbuilder.setContentText("Download complete");
// Removes the progress bar.
setProgress(0, 0, false);
```

Displaying an Activity Indicator

- Use `setProgress(0, 0, true)` and issue the notification.
- Issue the notification at the beginning of the operation.
- The animation runs until the notification is modified.
- When the operation is done, call `setProgress(0, 0, false)` and update the notification to remove the activity indicator.