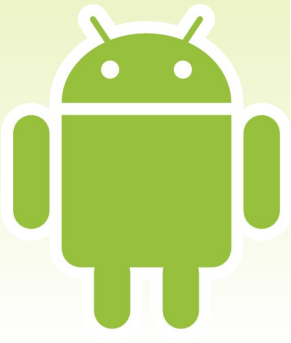
Event Buses

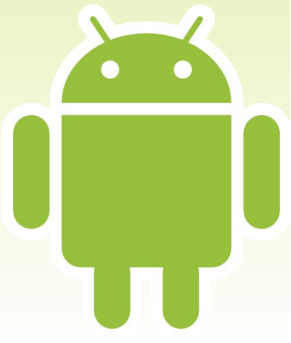
What We Have Here is a Failure to Communicate

- Android Design: Loose Coupling of Components
  - Allows activities, etc. to be implemented internally or by third parties, sometimes at user discretion
  - Reminiscent of Web apps, where pages do not (normally) communicate with other pages
    - GET parameters ~= Intent extras
  - Elegant in theory, clunky in practice
    - Clunkigant?



3 Inter-Component Communication Patterns

- Introduce Tight Coupling
  - Bound services and callbacks
  - Messenger
- Use IPC
  - PendingIntent
  - Regular broadcasts
- Use an In-Process Event Bus
  - ???

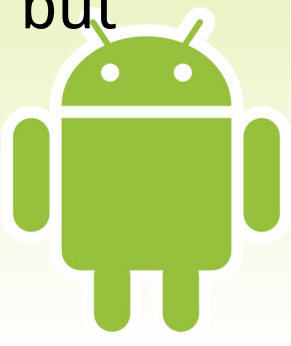


Get On The Bus (Gus)

- Publish/subscribe pattern
  - Components register interest in specific types of events
  - Components publish those events, which are then delivered to the registered parties
- Pros
  - Logically decoupled, if not “physically”
  - Bus implementation can use soft references, etc. to help mitigate GC risks

Where Buses Go

- Example Use Patterns
  - Service → UI layer
    - Done correctly, can also handle cases where there is no relevant UI, so we show a Notification instead
  - Fragment → activity
    - Or fragment → fragment, in cases where they happen to both be on the screen
  - Broadcast receiver → something else
    - For system broadcasts, etc. that require a receiver, but the business logic is best handled elsewhere

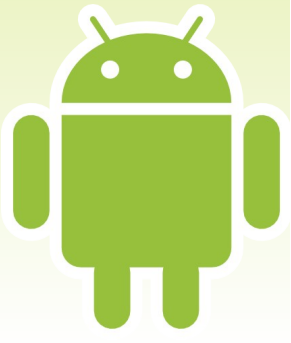


Bus-Like Substances

- Broadcast Intents
  - IPC
  - Security
- ContentObserver
  - Inflexible
  - IPC?

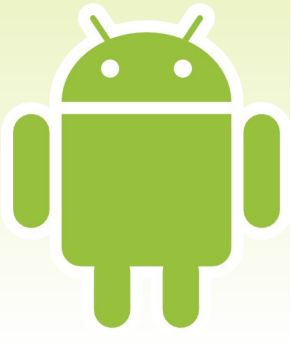
The Big Three Buses

- LocalBroadcastManager
- Square's Otto
- greenrobot's EventBus



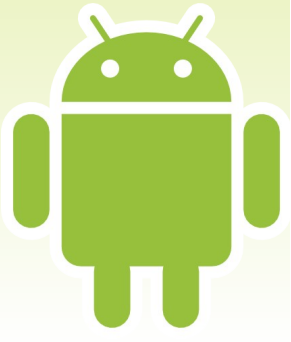
LocalBroadcastManager

- Pros
  - Part of Android Support package
    - Backed by the full faith and credit of Google
    - You're probably already using it, so no new library artifacts, licenses, etc.
  - Uses broadcast Intent semantics and structures
    - BroadcastReceiver
    - IntentFilter



LocalBroadcastManager

- Cons
  - Uses broadcast Intent semantics and structures
    - Limited by Intent and Bundle and such
    - Tends to be more verbose
  - Limited thread options
    - Messages delivered on main application thread only
  - No ordered broadcasts
  - No sticky broadcasts



```
public class EventDemoActivity extends
    SherlockFragmentActivity {
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    if (getSupportFragmentManager().findFragmentById(android.R.id.content) == null) {
      getSupportFragmentManager().beginTransaction()
                                 .add(android.R.id.content,
                                      new EventLogFragment()).commit();

      PollReceiver.scheduleAlarms(this);
    }
  }
}
```

```
public class PollReceiver extends BroadcastReceiver {
  private static final int PERIOD=15000; // 15 seconds
  private static final int INITIAL_DELAY=1000; // 1 second

  @Override
  public void onReceive(Context ctxt, Intent i) {
    if (i.getAction() == null) {
      WakefulIntentService.sendWakefulWork(ctxt, ScheduledService.class);
    }
    else {
      scheduleAlarms(ctxt);
    }
  }

  static void scheduleAlarms(Context ctxt) {
    AlarmManager mgr=
        (AlarmManager)ctxt.getSystemService(Context.ALARM_SERVICE);
    Intent i=new Intent(ctxt, PollReceiver.class);
    PendingIntent pi=PendingIntent.getBroadcast(ctxt, 0, i, 0);

    mgr.setRepeating(AlarmManager.ELAPSED_REALTIME_WAKEUP,
                     SystemClock.elapsedRealtime() + INITIAL_DELAY,
                     PERIOD, pi);

  }
}
```

```
public class ScheduledService extends WakefulIntentService {
  private static int NOTIFY_ID=1337;
  private Random rng=new Random();

  public ScheduledService() {
    super("ScheduledService");
  }
```

```
@Override
protected void doWakefulWork(Intent intent) {
  Intent event=new Intent(EventLogFragment.ACTION_EVENT);
  long now=Calendar.getInstance().getTimeInMillis();
  int random=rng.nextInt();

  event.putExtra(EventLogFragment.EXTRA_RANDOM, random);
  event.putExtra(EventLogFragment.EXTRA_TIME, now);

  if (!LocalBroadcastManager.getInstance(this).sendBroadcast(event)) {
    NotificationCompat.Builder b=new NotificationCompat.Builder(this);
    Intent ui=new Intent(this, EventDemoActivity.class);

    b.setAutoCancel(true).setDefaults(Notification.DEFAULT_SOUND)
     .setContentTitle(getString(R.string.notif_title))
     .setContentText(Integer.toHexString(random))
     .setSmallIcon(android.R.drawable.stat_notify_more)
     .setTicker(getString(R.string.notif_title))
     .setContentIntent(PendingIntent.getActivity(this, 0, ui, 0));

    NotificationManager mgr=
        (NotificationManager)getSystemService(NOTIFICATION_SERVICE);

    mgr.notify(NOTIFY_ID, b.build());
  }
}
```

```
public class EventLogFragment extends SherlockListFragment {
  static final String EXTRA_RANDOM="r";
  static final String EXTRA_TIME="t";
  static final String ACTION_EVENT="e";
  private EventLogAdapter adapter=null;

  @Override
  public void onActivityCreated(Bundle state) {
    super.onActivityCreated(state);

    setRetainInstance(true);
    getListView().setTranscriptMode(ListView.TRANSCRIPT_MODE_NORMAL);

    if (adapter == null) {
      adapter=new EventLogAdapter();
    }

    setListAdapter(adapter);
  }
```

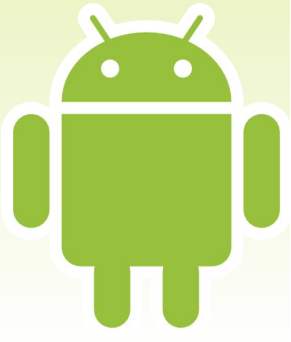
```
class EventLogAdapter extends ArrayAdapter<Intent> {
  DateFormat fmt=new SimpleDateFormat("HH:mm:ss", Locale.US);

  public EventLogAdapter() {
    super(getActivity(), android.R.layout.simple_list_item_1,
          new ArrayList<Intent>());
  }

  @SuppressLint("DefaultLocale")
  @Override
  public View getView(int position, View convertView, ViewGroup parent) {
    TextView row=
        (TextView)super.getView(position, convertView, parent);
    Intent event=getItem(position);
    Date date=new Date(event.getLongExtra(EXTRA_TIME, 0));

    row.setText(String.format("%s = %x", fmt.format(date),
                              event.getIntExtra(EXTRA_RANDOM, -1)));

    return(row);
  }
}
```

```
@Override
public void onResume() {
  super.onResume();

  IntentFilter filter=new IntentFilter(ACTION_EVENT);

  LocalBroadcastManager.getInstance(getActivity())
                       .registerReceiver(onEvent, filter);
}

@Override
public void onPause() {
  LocalBroadcastManager.getInstance(getActivity())
                       .unregisterReceiver(onEvent);

  super.onPause();
}
```

```
private BroadcastReceiver onEvent=new BroadcastReceiver() {
  @Override
  public void onReceive(Context context, Intent intent) {
    adapter.add(intent);
  }
};
```

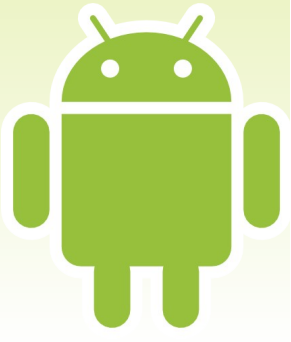
Otto

- Pros
  - It's Square (so it's cool)
  - Annotation based (so it's cool)
  - Flexible events (any class you like)
  - Clean API
  - Event producers
    - Reminiscent of sticky broadcasts
    - Helps to handle the case where the subscriber is newly created and needs to “bootstrap” data



Otto

- Cons
  - Annotation based
    - Overhead, particularly on older devices
  - Event registration not inherited
  - Limited thread options
    - Messages delivered on thread they are sent upon
      - And exception thrown if that's not the main application thread, unless you configure it to allow for any thread
    - May require somebody to fuss around with getting the event over to the main application thread
  - No ordered events
    - ...but can find out if nobody received an event, so you can use this for UI-or-Notification pattern



```java
public class ScheduledService extends WakefulIntentService {
  static final Bus bus=new Bus(ThreadEnforcer.ANY);
  private static int NOTIFY_ID=1337;
  private Random rng=new Random();

  public ScheduledService() {
    super("ScheduledService");
  }

  @Override
  public void onCreate() {
    super.onCreate();

    bus.register(this);
  }

  @Override
  protected void doWakefulWork(Intent intent) {
    bus.post(new RandomEvent(rng.nextInt()));
  }

  @Override
  public void onDestroy() {
    bus.unregister(this);

    super.onDestroy();
  }
```

```java
public class RandomEvent {
   Date when=Calendar.getInstance().getTime();
   int value;

   RandomEvent(int value) {
      this.value=value;
   }
}
```

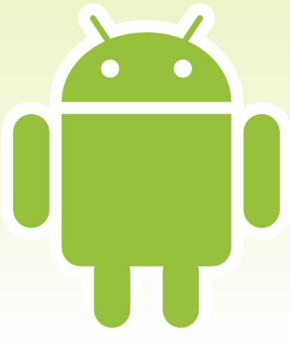
```
@Subscribe
public void onDeadEvent(DeadEvent braiiiiiiinz) {
  RandomEvent original=(RandomEvent)braiiiiiiinz.event;
  NotificationCompat.Builder b=new NotificationCompat.Builder(this);
  Intent ui=new Intent(this, EventDemoActivity.class);

  b.setAutoCancel(true).setDefaults(Notification.DEFAULT_SOUND)
   .setContentTitle(getString(R.string.notif_title))
   .setContentText(Integer.toHexString(original.value))
   .setSmallIcon(android.R.drawable.stat_notify_more)
   .setTicker(getString(R.string.notif_title))
   .setContentIntent(PendingIntent.getActivity(this, 0, ui, 0));

  NotificationManager mgr=
      (NotificationManager)getSystemService(NOTIFICATION_SERVICE);

  mgr.notify(NOTIFY_ID, b.build());
}
```

```java
@Override
public void onResume() {
  super.onResume();

  ScheduledService.bus.register(this);
}

@Override
public void onPause() {
  ScheduledService.bus.unregister(this);

  super.onPause();
}

@Subscribe
public void onRandomEvent(final RandomEvent event) {
  if (getActivity() != null) {
    getActivity().runOnUiThread(new Runnable() {
      @Override
      public void run() {
        adapter.add(event);
      }
    });
  }
}
```

```
class EventLogAdapter extends ArrayAdapter<RandomEvent> {
  DateFormat fmt=new SimpleDateFormat("HH:mm:ss", Locale.US);

  public EventLogAdapter() {
    super(getActivity(), android.R.layout.simple_list_item_1,
          new ArrayList<RandomEvent>());
  }

  @SuppressLint("DefaultLocale")
  @Override
  public View getView(int position, View convertView, ViewGroup parent) {
    TextView row=
        (TextView)super.getView(position, convertView, parent);
    RandomEvent event=getItem(position);

    row.setText(String.format("%s = %x", fmt.format(event.when),
                              event.value));

    return(row);
  }
}
```

greenrobot's EventBus

- Pros
  - Flexible events (any class you like)
  - Clean API
  - Sticky events
  - Ordered event delivery
  - Robust threading options
    - Posting thread
    - Main application thread
    - Dedicated background thread
    - Posting-or-background thread



greenrobot's EventBus

- Cons
  - Naming convention for event receipt methods
    - Not quite as flexible as Otto, but does save on annotation overhead
  - Ordered events only work with “posting thread” option
    - May require somebody to fuss around with getting the event over to the main application thread

```java
@Override
public void onCreate() {
  super.onCreate();

  EventBus.getDefault().register(this, 0);
}

@Override
protected void doWakefulWork(Intent intent) {
  EventBus.getDefault().post(new RandomEvent(rng.nextInt()));
}

@Override
public void onDestroy() {
  EventBus.getDefault().unregister(this);

  super.onDestroy();
}
```

```
public void onEvent(RandomEvent event) {
  NotificationCompat.Builder b=new NotificationCompat.Builder(this);
  Intent ui=new Intent(this, EventDemoActivity.class);

  b.setAutoCancel(true).setDefaults(Notification.DEFAULT_SOUND)
   .setContentTitle(getString(R.string.notif_title))
   .setContentText(Integer.toHexString(event.value))
   .setSmallIcon(android.R.drawable.stat_notify_more)
   .setTicker(getString(R.string.notif_title))
   .setContentIntent(PendingIntent.getActivity(this, 0, ui, 0));

  NotificationManager mgr=
      (NotificationManager)getSystemService(NOTIFICATION_SERVICE);

  mgr.notify(NOTIFY_ID, b.build());
}
```

```java
@Override
public void onResume() {
  super.onResume();

  EventBus.getDefault().register(this, 1);
}

@Override
public void onPause() {
  EventBus.getDefault().unregister(this);

  super.onPause();
}

public void onEvent(final RandomEvent event) {
  getActivity().runOnUiThread(new Runnable() {
    @Override
    public void run() {
      adapter.add(event);
    }
  });

  EventBus.getDefault().cancelEventDelivery(event);
}
```

Choosing a Bus

- LocalBroadcastManager
  - Cannot use third-party libraries
  - Using regular broadcasts now and want to migrate over code with minimal fuss
- Otto
  - If you're using other Square libraries that work particularly well with it
- greenrobot's EventBus
  - General recommended choice