Topics

- Architecture
- Activity & Intent
- Broadcast Receiver
- Service
- Content Provider

- And stuff in between…

© 2009 - 2010 Digia Plc

The main topics covered by this course cover:

- Using Eclipse IDE for Android development
- Android architecture
    - Linux
    - Modules
    - Processes & threads
- AndroidManifest.xml
    - Semantics
    - apk

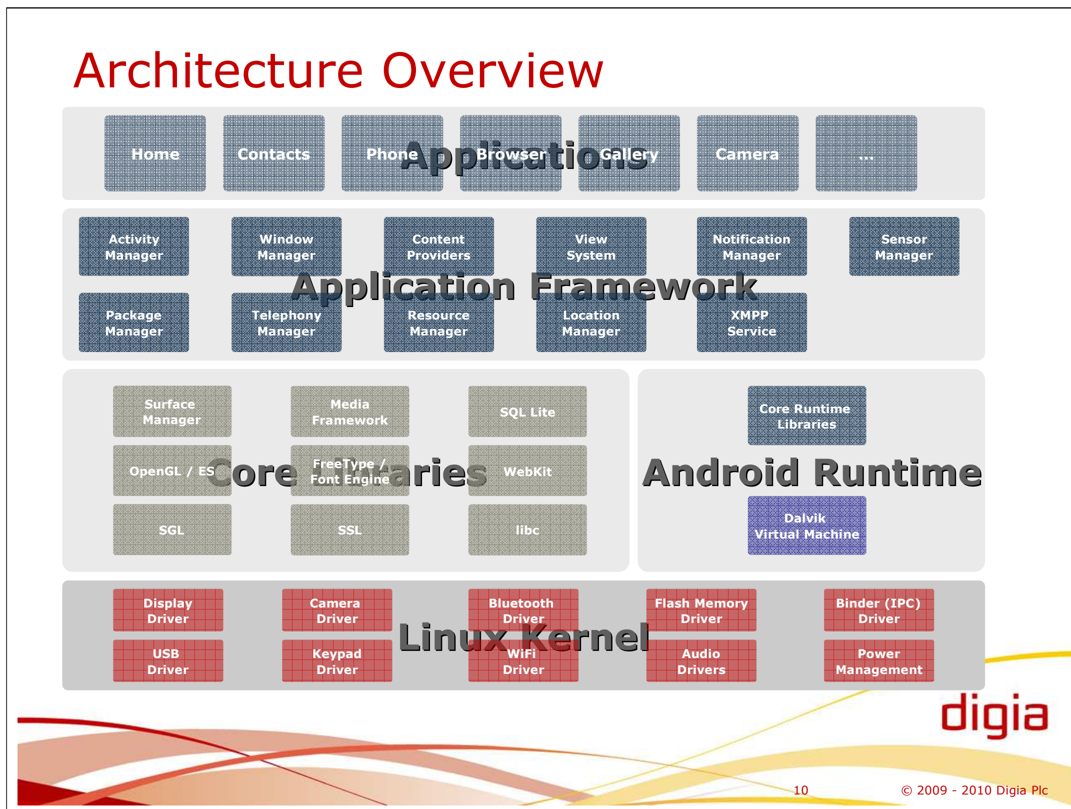
- Activity & Intent
    - Matching
    - Hook method sequences
    - ListActivity
- Content Provider
    - Cursor
    - SimpleCursorAdapter
- Service
    - AIDL
    - Thread model
    - Starting and stopping a service
    - Implementing a service
    - Using a service from an application
- Views
- Dialogs
- Menus
- Handler & Message
    - Implementing responsive UI
    - Hand-over/synchronization of threads
- Broadcast Receiver

Open Source

All the source code of Android is available from its git repository. You can access source code also through web pages at http://android.git.kernel.org. The repository can be valuable place to check out some detailed functionality of a UI component, for example.

The repository includes all the releases of Android. Also, the most recent version of codes in the repository does not necessarily correspond to the most recent release of the android.

Particular device may have its own additions made by the mobile device manufacturer. The android git repository contains only the “official” set of codes. Also, device drivers may or may not be located in the repository.



The architecture of Android can be divided into four layers:

1. Applications, system built-in applications and 3rd party applications
2. Application Framework provides components and services for applications.
3. Core Libraries provides “low-level” functionality that is wrapped and used by the Application Framework.
4. Linux Kernel provides the memory, process/thread, device driver, and security management

Linux kernel is based on 2.6 version. There is no particular Linux distribution inside/for Android.

Binder driver is Android specific device driver for IPC (Inter-Process Communication) based on shared memory.

Between the Linux Kernel and Libraries layers Android specifies its hardware abstraction layer providing interfaces, for example, for Wi-Fi, GPS, Radio, Bluetooth, Camera, Audio, and Graphics. That abstraction specifies what is required from the under lying operating system (Linux kernel and device drivers) by Android.

Libc is dedicated, BSD-based, library for the purposes of Android. Android libc is optimized for the memory foot print. There is no glibc in Android.

Own native, non-Java, application development is possible with Android by using Android NDK (Android Native Development Kit). The NDK contains cross-compiler, build files, libraries, headers required for compiling and linking C/C++ applications.

Dalvik is the java virtual machnine of Android, which is optimized (memory foot print) for mobile devices. Dalvik executes so-called dex files (dalvik executables). Dex files are converted from the jave compiler output with the dx tool of Android.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8" ?>
- <manifest xmlns:android="http://schemas.android.com/apk/res/android" package="com.example.android.sequenceactivity"
    android:versionCode="1" android:versionName="1.0">
  - <application android:icon="@drawable/icon" android:label="@string/app_name">
    - <activity android:name=".SequenceActivity" android:label="@string/app_name">
      - <intent-filter>
          <action android:name="android.intent.action.MAIN" />
          <category android:name="android.intent.category.DEFAULT" />
          <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
      </activity>
    - <activity android:name=".MyList">
      - <intent-filter>
          <action android:name="android.intent.action.MAIN" />
          <category android:name="android.intent.category.SAMPLE_CODE" />
        </intent-filter>
      </activity>
    + <service android:name="com.example.android.sequenceservice.SequenceService" android:process=":service">
    - <receiver android:name=".MyReceiver" android:exported="true" android:enabled="true">
      - <intent-filter>
          <action android:name="jep" />
        </intent-filter>
      </receiver>
      <provider android:name="com.example.android.fibonacci.FibonacciContentProvider"
        android:authorities="com.example.android.fibonacci" />
    </application>
    <uses-sdk android:minSdkVersion="6" />
</manifest>
```

digia

© 2009 - 2010 Digia Plc

AndroidManifest.xml is a specification for your application. It specifies what are the elements of your application: application name and icon, activities, services, content providers, and broadcast receivers. In other words, it lists the elements that could be used, or referenced, by others.

Android tools, or Eclipse plug-in, creates the AndroidManifest for you.

Example:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.example.android.sequenceactivity"
      android:versionCode="1"
      android:versionName="1.0">

    <application android:icon="@drawable/icon" android:label="@string/app_name">

                        <activity android:name=".SequenceActivity" android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.DEFAULT" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <activity android:name=".MyList">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.SAMPLE_CODE" />
            </intent-filter>
        </activity>

                        <service
                                        android:name="com.example.android.sequenceservice.SequenceService"
                                        android:process=":service">
            <intent-filter>
                <!-- These are the interfaces supported by the service, which
                     you can bind to. -->
                <action android:name="servicestub.ISequence" />
            </intent-filter>

                        </service>
```

Code Insert: The First Activity

- How to get started
- Eclipse IDE
- Creating an Android project
- Running an activity
- Parts of project

© 2009 - 2010 Digia Plc

- Architecture
- Activity & Intent
- Broadcast Receiver
- Service
- Content Provider

digia

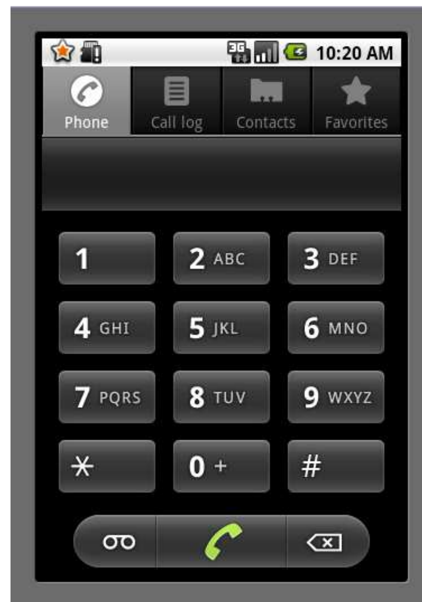
© 2009 - 2010 Digia Plc

The main topics covered by this course cover:

- Using Eclipse IDE for Android development
- Android architecture
  - Linux
  - Modules
  - Processes & threads
- AndroidManifest.xml
  - Semantics
  - apk

- Activity & Intent
  - Matching
  - Hook method sequences
  - ListActivity
- Content Provider
  - Cursor
  - SimpleCursorAdapter
- Service
  - AIDL
  - Thread model
  - Starting and stopping a service
  - Implementing a service
  - Using a service from an application
- Views
- Dialogs
- Menus
- Handler & Message
  - Implementing responsive UI
  - Hand-over/synchronization of threads
- Broadcast Receiver

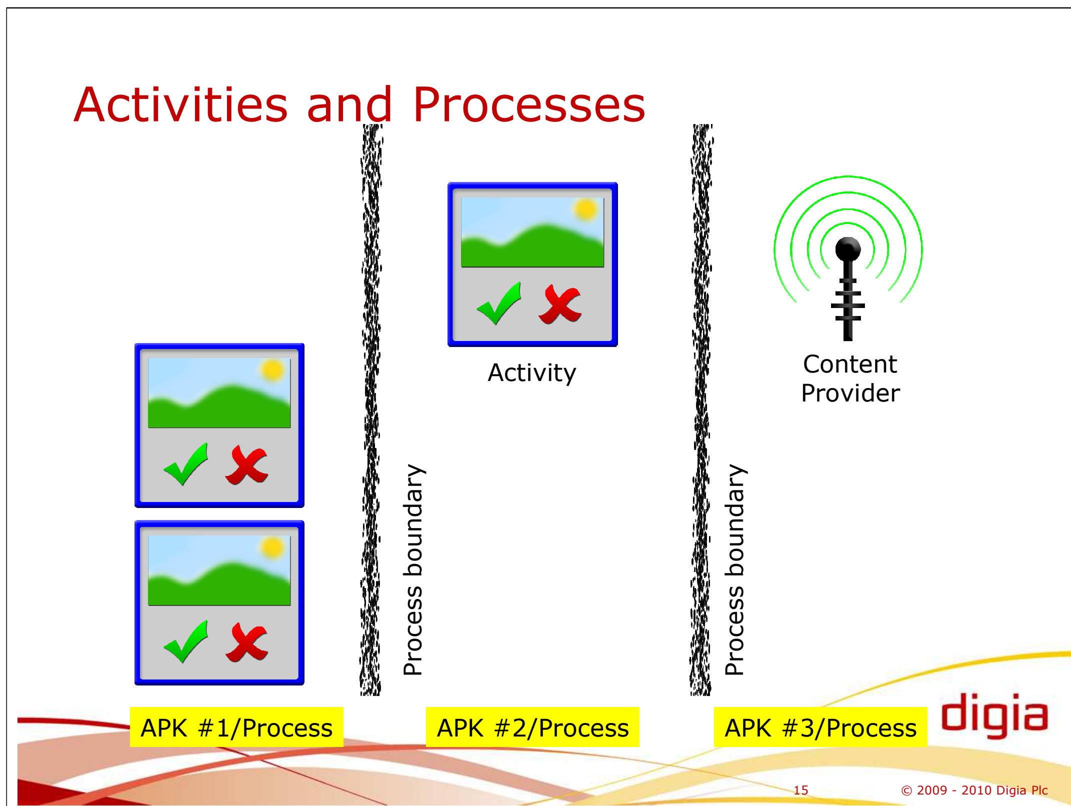
Activity



digia

© 2009 - 2010 Digia Plc

An activity represents a set of functionality of the user-interface of an application, not the whole user-interface. In other words, the user-interface consists of multiple activities.



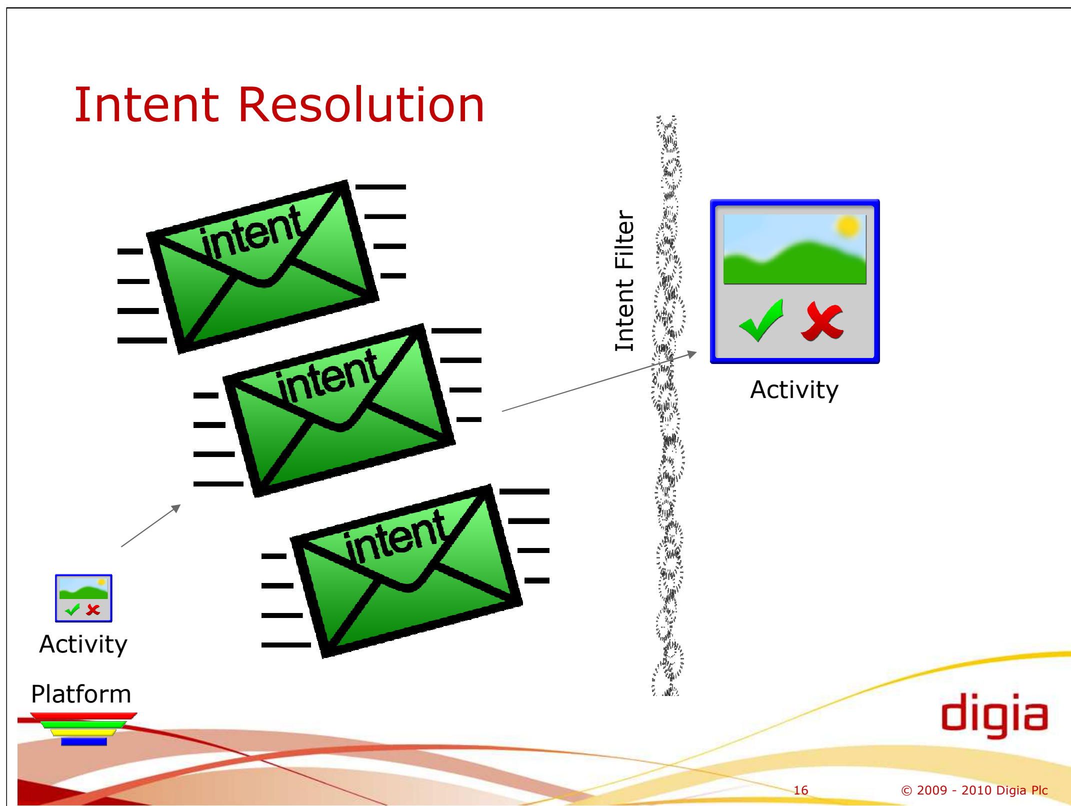
Each APK (Android Package) corresponds to one process by default. The start and stop of a process is transparent for the user and the application developer. If an activity use another activity from another APK, a process is launched for it, if it does not already exist.

By default there is one thread (main thread) per a process. The main thread is running the Looper class (see Activity Hooks slide). However, the process have other threads and they are used by IPC mechanism.

Every APK is assigned with a Linux user id meaning that every application is running as a different user because of security.

Android platform may choose to kill processes that do not have visible activities in cases of low memory.

Activities can pass resulted data back to the caller activity inside an intent.



Intent can be seen as a request for something. One (or zero) activity in the platform is chosen to serve that intent. Intent is used to resolve an activity, broadcast receiver, and service. Content resolver is used to refer to content provider.

There are two ways to specify an intent:

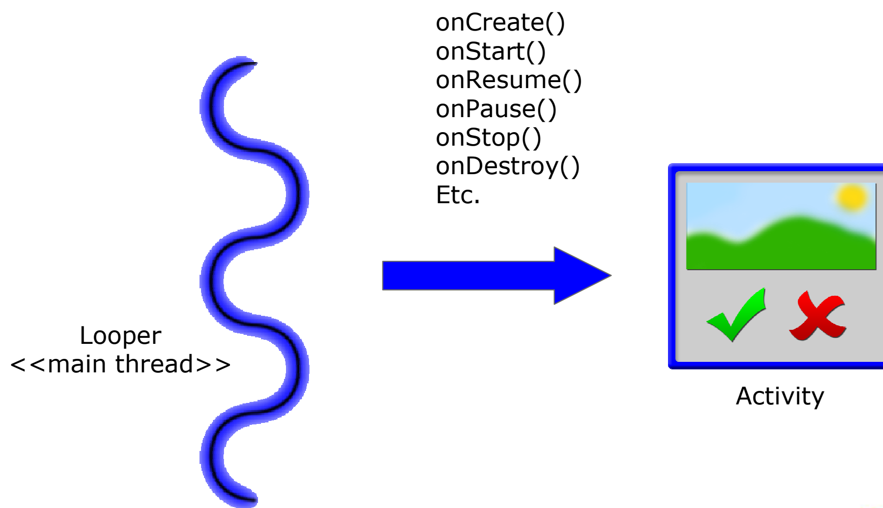
1. Explicit intent – the exact class is specified, which is activated.
2. Implicit intent – information is provided by the intent that is then used in matching the activity by the platform.

In the case of implicit intent, the intent information used in intent resolution is provided in AndroidManifest.xml files of activities/applications. The platform then choose the best activity, or component, to be activated. Intent resolution can be based on:

- Action
- Type
- URI
- Category

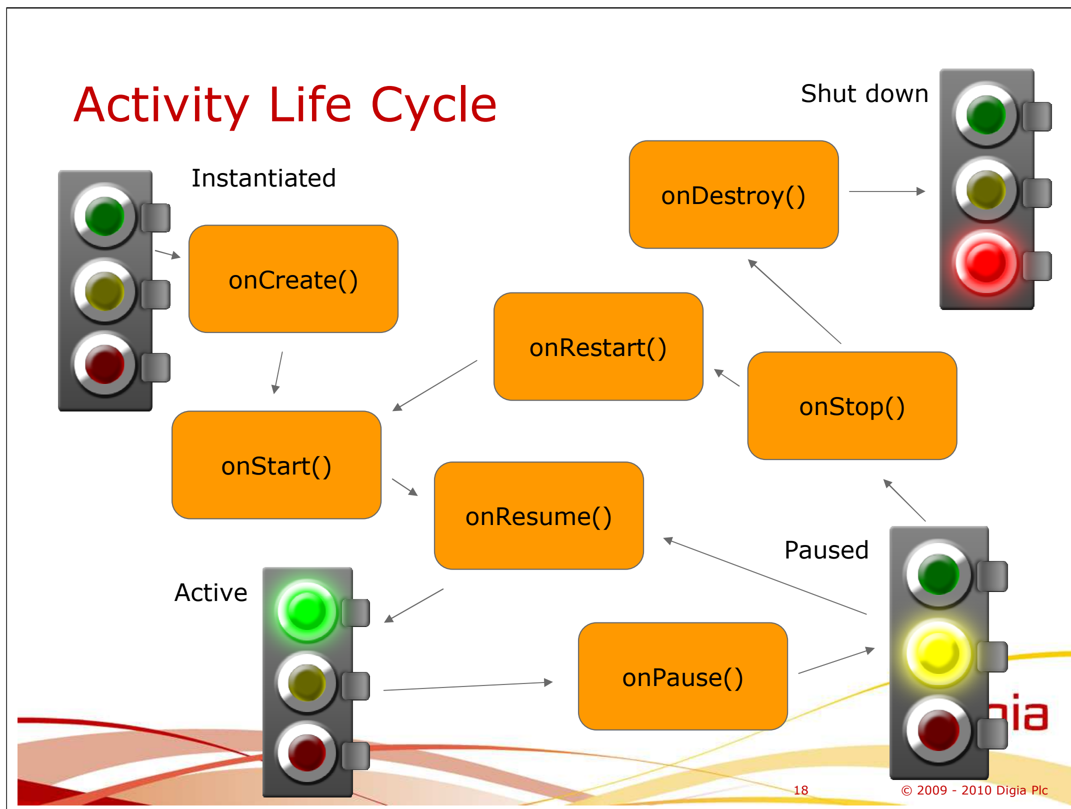
References:

- http://developer.android.com/reference/android/content/Intent.html



Activity is used by the platform by calling the hook methods of that activity. Hook methods are called always by the same thread (the main/UI thread). There are a number of hook methods specified in the interface of activity. Hook methods exist, for example, for handling the life cycle of activity, managing dialogs and menus, and handling touch events.

The time spend in these hook methods, especially in the event handling functions, should be minimized to keep the application responsive.



Activity is paused if there is another activity on top of it. The paused activity can be visible because the top most activity is transparent or it has a window not covering the whole screen. The paused activity can be resumed to be active.

Activity is stopped if there is another activity on top of it covering it totally. The stopped activity can be restarted to be active again.

Hook methods of activity:

-onCreate() – called when an activity is created. Most initialized can be done in this hook method.

-onStart() – called just before the activity becomes visible to the user..

-onResume() – called just before an activity starts interacting with the user.

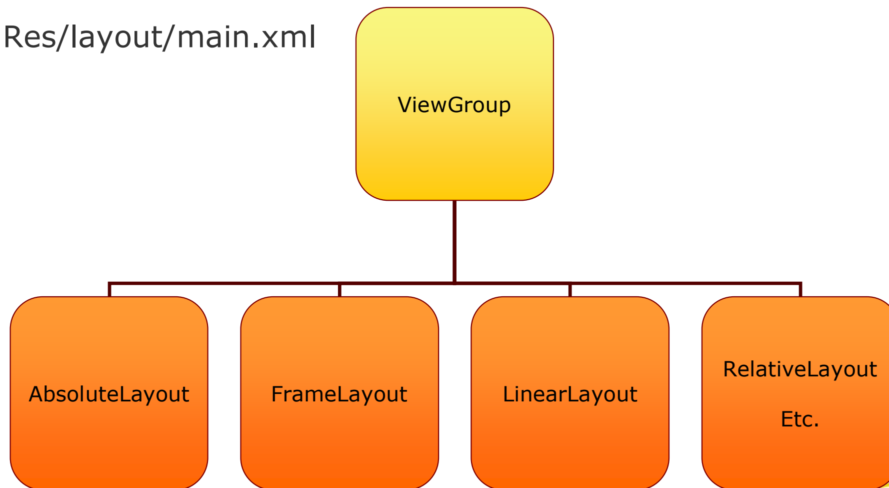
-onPause() – called when another activity is started on top of this. This his the moment to freeze animations, for example. After onPause() the platform may kill the process where the activity resides. It is recommended that the application commits changes on any persistent data when onPause() is called.

-onStop() – another activity is covering the activity making the activity invisible. Next: onRestart() or onDestroy().

-onDestroy() – activity is destroyed, called when Activity.finished() is called.

onCreate(), onStart, and onResume() functions are more or less related in start-up phase of activity. Activity should decide how it divide its functionality into these phases.

After the Activity.onPause() the process can be killed and rest of the hook



Layouts specified in XML and used by activities of the application are located in res/layout directory.

AbsoluteLayout is used to position its children into exact x, y location.

FrameLayout stacks ups its children into one position.

LinearLayout positions its children horizontally or vertically.

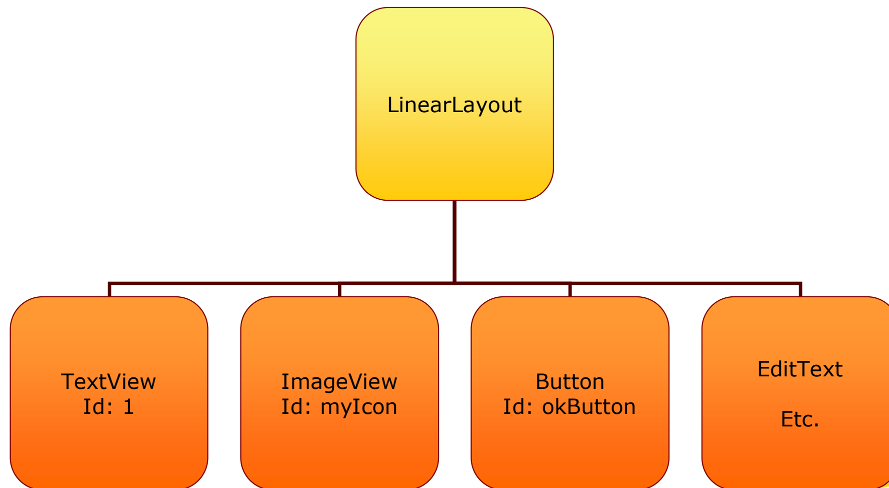
RelativeLayout aligns its children based on the positions of other views (parent or other children).

Other view groups: Gallery, GridView, ListView, ScrollView, Spinner, SurfaceView, TabHost, TableLayout, ViewFlipper, and ViewSwitcher.

References:

-http://developer.android.com/reference/android/view/ViewGroup.html

-http://developer.android.com/reference/android/view/View.html



The visual appearance of an activity is consisting of view groups and views inside a view group.

Note that the hierarchy of elements can be nested and contain nested view groups.

Commonly, elements are built by the platform based on specifications in XML (res/layout).

Views are associated with an id. Activity can access any view by calling findViewById() function. Activity needs to access views, for example, for setting listeners to views.

Event Listeners

- View.OnClickListener.onClick()
- View.onLongClickListener.onLongClick()
- View.onFocusChangeListener.onFocusChange()
- View.onKeyListener.onKey()
- View.onTouchListener.onTouch()

© 2009 - 2010 Digia Plc

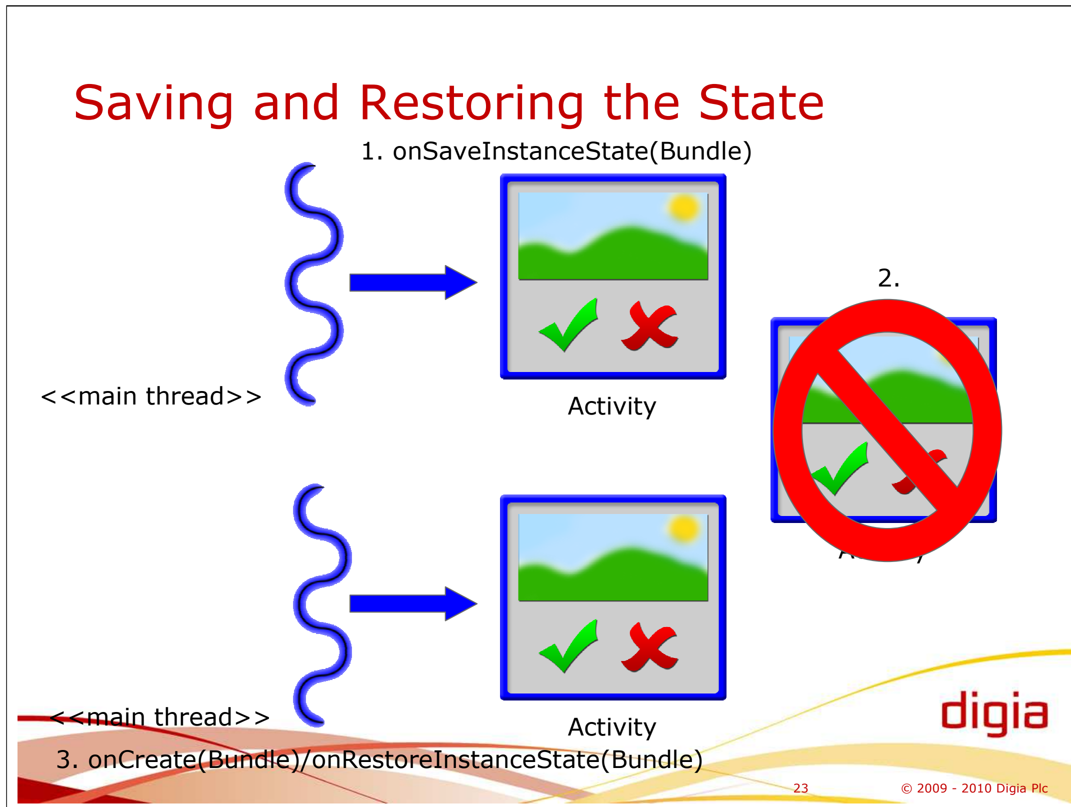
An activity, for example, can process events by implementing various event listener interfaces and subscribing the observer view.

Code Insert: OnClickListener

- Using a Button
- Defining OnClickListener
- Subscribing listener to a button

digia

© 2009 - 2010 Digia Plc



onSaveInstanceState()/onRestoreInstanceState() can be used to save the transient state of activity.

onRestoreInstanceState() is called just after onStart().

Bundle can be used to store name/value pairs that describe the state of activity.

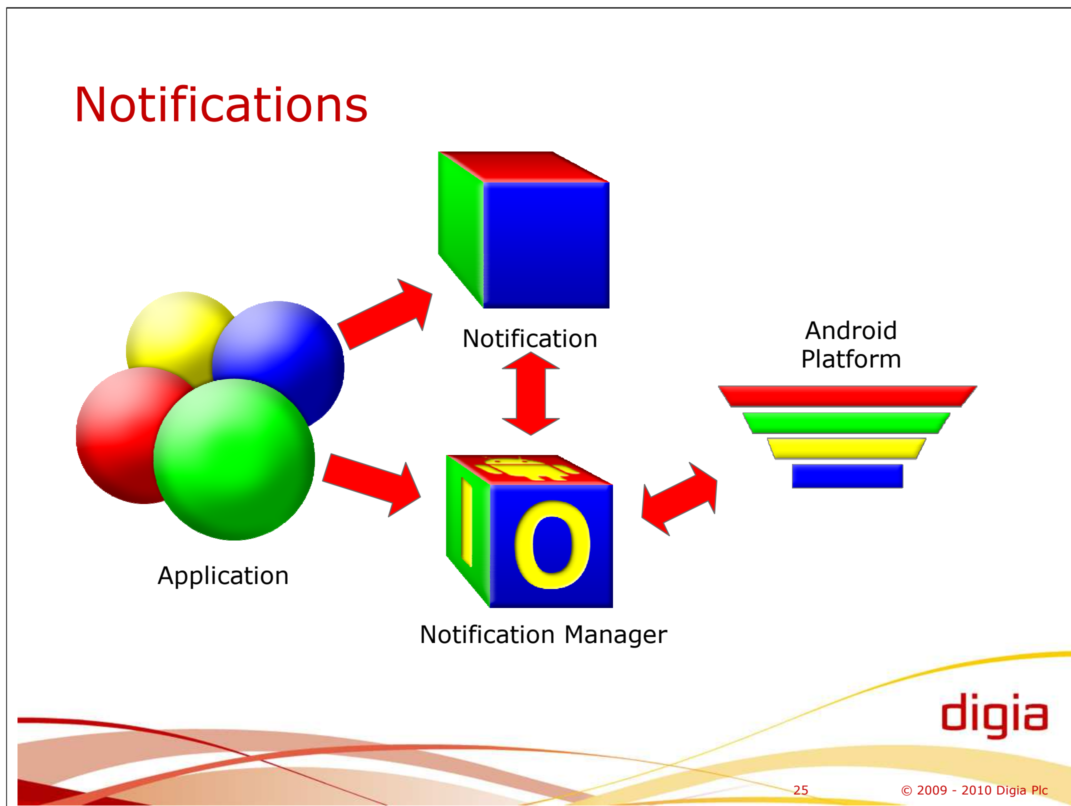
Note that onSaveInstanceState() is not called if the user is navigating with BACK button, which does not save the current state of activity. In other words, if the user navigates back, there is no way to restore the activity (other than just start it again, have a fresh copy).

Code Insert: Saving the State

- Using a Bundle
- Introducing onSaveInstanceState()
- Introducing onRestoreInstanceState()
- Saving and restoring the state of an activity

digia

© 2009 - 2010 Digia Plc



Android provides various notifications. These include:

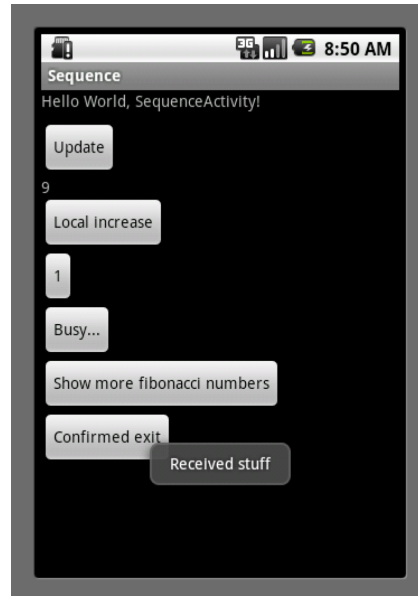
1. Messages (texts and icons)
2. Turning LEDs on and off
3. Vibrating
4. Playing a sound

NotificationManager is the interface for the application to display and cancel notifications represented by Notification class.

References:

- http://developer.android.com/reference/android/app/Notification.html
- http://developer.android.com/reference/android/app/NotificationManager.html

Toast



digia

© 2009 - 2010 Digia Plc

Android provides an easy way to show text notifications on top of the screen. They can be triggered by background applications and services.

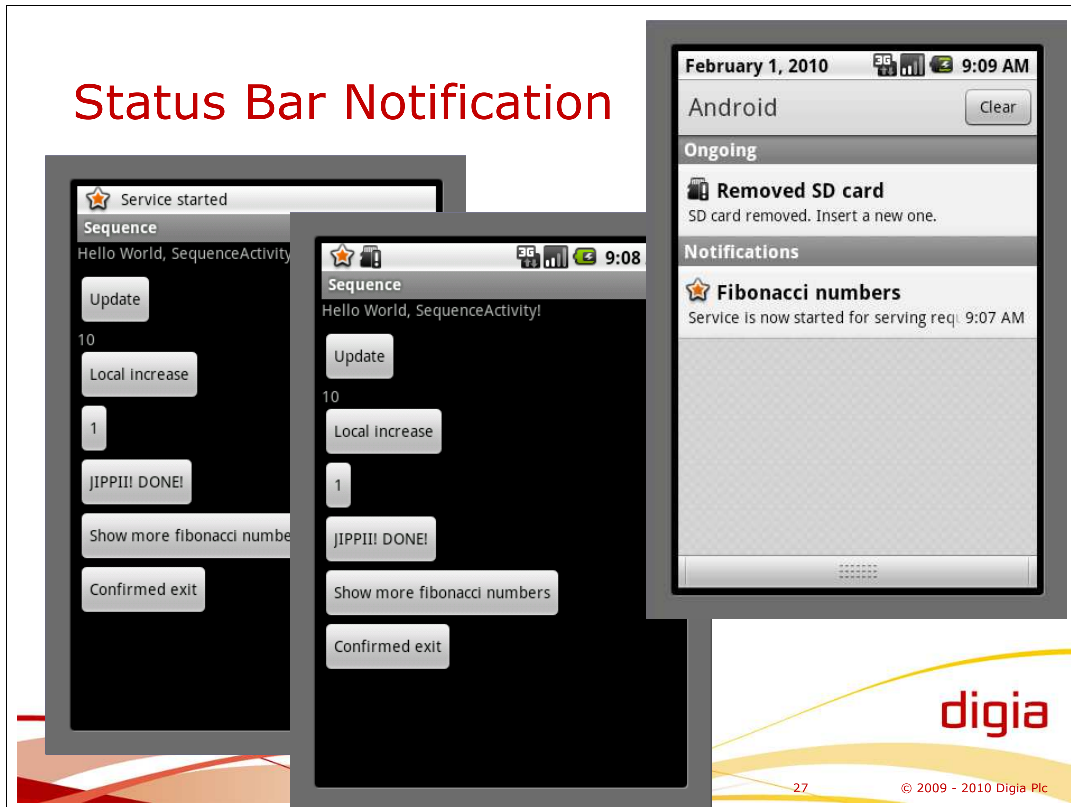
Toast.*makeText*(context, "Received stuff", *TIME_TO_DISPLAY*).show();

The Toast is a quick way to make simple notifications for the user without disturbing the user too much. The notification is not blocking the active application and it is dismissed automatically.

The Toast notification can use an application specific layout to create a custom notification message.

References:

-http://developer.android.com/guide/topics/ui/notifiers/toasts.html



A status bar notification provides a persistent notification. The dismissal policy of the notification can be specified by the application.

Notification is associated with an intent that is used to start an activity if the user taps on the notification. That can be used to activate the application (if in the background) that published the notification, for example.

Publishing a notification:

```
private void showServiceStartedNotification() {
        Notification notification = new Notification(android.R.drawable.star_big_on,
        getString(R.string.noteServiceStarted_tickerText), System.currentTimeMillis());
        notification.flags |= Notification.FLAG_AUTO_CANCEL;

        String contentTitle = getString(R.string.noteServiceStarted_contentTitle);
        String contentText = getString(R.string.noteServiceStarted_contentText);
        Intent notificationIntent = new Intent(SequenceActivity.this, SequenceActivity.class);

        Context applicationContext = getApplicationContext();
        PendingIntent contentIntent = PendingIntent.getActivity(applicationContext, 0, notificationIntent, 0);

        notification.setLatestEventInfo(applicationContext, contentTitle, contentText, contentIntent);

        NotificationManager manager =
        (NotificationManager)getSystemService(NOTIFICATION_SERVICE);
        manager.notify(NOTIFICATION_SERVICE_STARTED, notification);
}
```

Cancelling a notification:

```
private void dismissServiceStartedNotification() {
        NotificationManager manager =
        (NotificationManager)getSystemService(NOTIFICATION_SERVICE);
        manager.cancel(NOTIFICATION_SERVICE_STARTED);
}
```

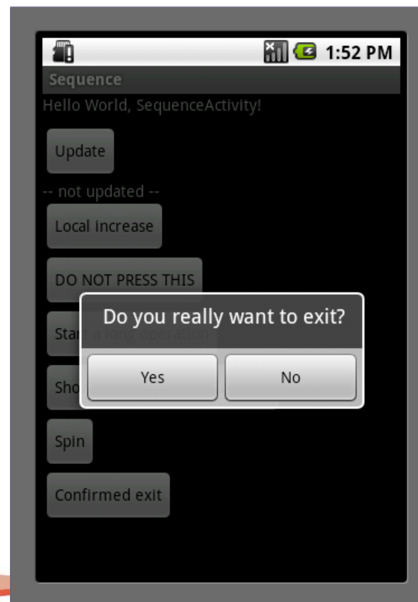
Code Insert: Status Bar Notification

- Creating a notification
- Use of Intent and PendingIntent
- Setting notification information
- Showing and cancelling the notification with Notification Manager

digia

© 2009 - 2010 Digia Plc

Dialog



digia

© 2009 - 2010 Digia Plc

Dialog can be built, or constructed, by using a dialog builder:

```
protected Dialog onCreateDialog(int id) {
        Dialog result = null;

        if (MY_EXIT_DIALOG == id) {
                AlertDialog.Builder builder = new AlertDialog.Builder(this);
                builder.setMessage(getString(R.string.exitDialogMessage))
                    .setCancelable(false)
                    .setPositiveButton(getString(R.string.exitDialogYes), new
                DialogInterface.OnClickListener() {
                        public void onClick(DialogInterface dialog, int id) {
                            SequenceActivity.this.finish();
                        }
                    })
                    .setNegativeButton(getString(R.string.exitDialogNo), new
                DialogInterface.OnClickListener() {
                        public void onClick(DialogInterface dialog, int id) {
                            dialog.cancel();
                        }
                    });
        result = builder.create();
}

Return result;
}
```

Managing Dialogs

- Dialogs are managed with the following functions:
  - Activity.showDialog(int id)
  - Dialog Activity.onCreateDialog(int id)
  - Activity.onPrepareDialog(int id, Dialog dialog)
  - Activity.dissmisDialog(int id)
  - Activity.removeDialog(int id)
  - Dialog.cancel()

digia

© 2009 - 2010 Digia Plc

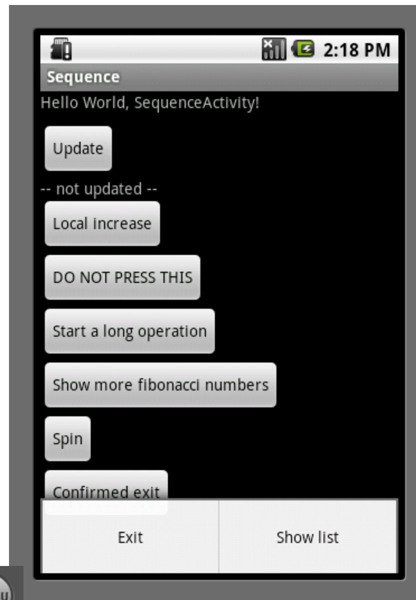
Code Insert: AlertDialog

- Creating a dialog
- Building of the dialog with AlertDialog.Builder
- Specifying the factory method for the dialog

digia

© 2009 - 2010 Digia Plc

Options Menu



digia

© 2009 - 2010 Digia Plc

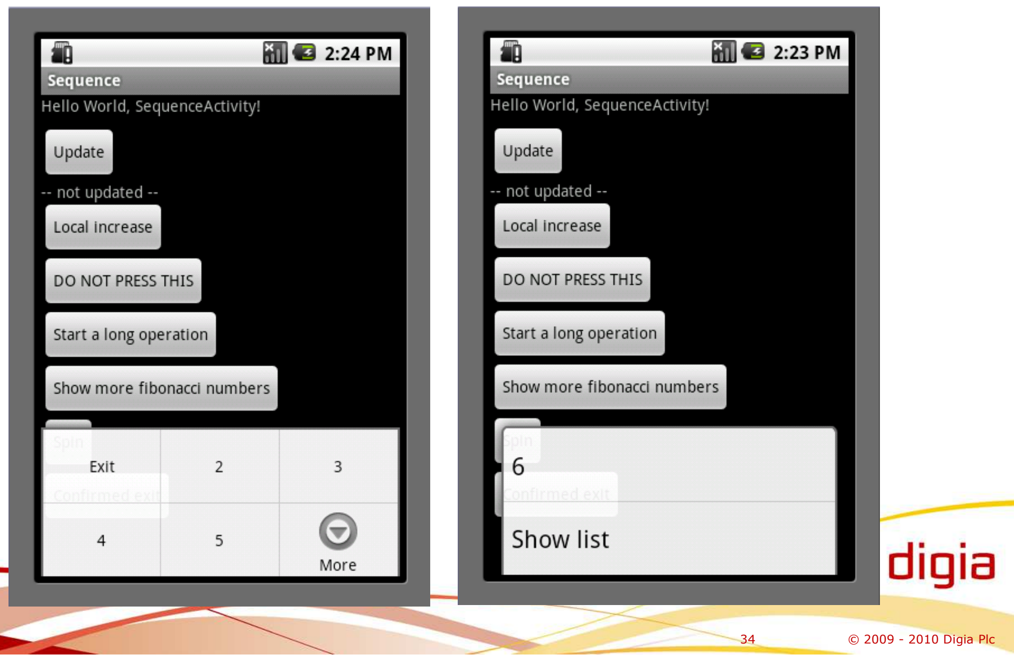
Managing Options Menus

- Options menus are managed with the following functions:
  - Boolean onCreateOptionsMenu(Menu menu)
  - Boolean onOptionsItemSelected(MenuItem item)
  - onOptionsMenuClosed(Menu menu)

digia

© 2009 - 2010 Digia Plc

Menu Items



There are six slots for menu items in the options menu.

Code Insert: Options Menu

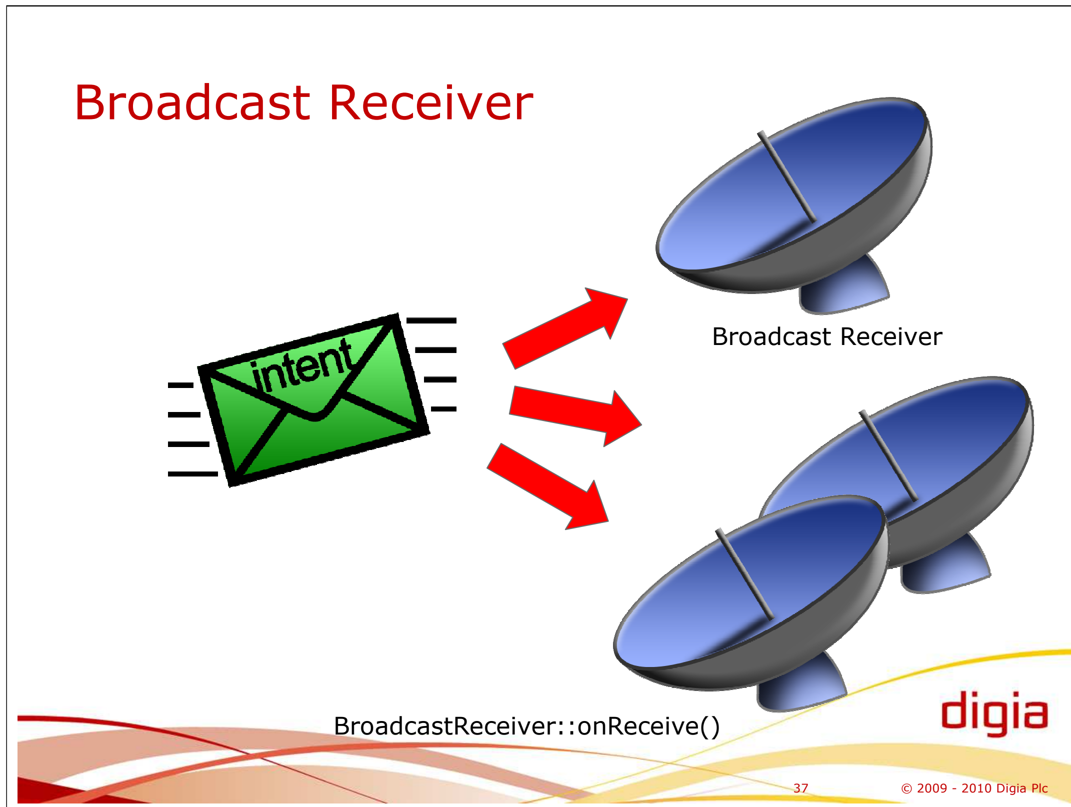
- Creating a options menu
- Populating a menu
- Reacting on menu item selection

digia

© 2009 - 2010 Digia Plc

- Architecture
- Activity & Intent
- Broadcast Receiver
- Service
- Content Provider

© 2009 - 2010 Digia Plc



For example, an activity can send broadcast events to so called broadcast receivers, or classes deriving from BroadcastReceiver.

There are system initiated broadcasts for various events:

- ACTION_TIME_TICK
- ACTION_TIME_CHANGED
- ACTION_TIMEZONE_CHANGED
- ACTION_BOOT_COMPLETED
- ACTION_PACKAGE_ADDED
- ACTION_PACKAGE_CHANGED
- ACTION_PACKAGE_REMOVED
- ACTION_PACKAGE_RESTARTED
- ACTION_PACKAGE_DATA_CLEARED
- ACTION_UID_REMOVED
- ACTION_BATTERY_CHANGED
- ACTION_POWER_CONNECTED
- ACTION_POWER_DISCONNECTED
- ACTION_SHUTDOWN

Intent is matched against intent filters of broadcast receivers by the platform.

Intent is broadcast by calling Context.sendBroadcast() function.

Intent Filter of Broadcast Receiver

```
- <receiver android:name=".MyReceiver" android:exported="true" android:enabled="true">
  - <intent-filter>
      <action android:name="jep" />
    </intent-filter>
  </receiver>
```


© 2009 - 2010 Digia Plc

Broadcast receivers can be created dynamically in run-time and be registered to receive intents.

Also, a broadcast receiver can be static meaning its intent filter is specified in AndroidManifest.xml and it is registered automatically by the platform to receive intents.

Code Insert: Broadcast Receiver

- Implementing a broadcast receiver
- Defining the intent filter of a broadcast receiver
- Sending broadcast intents

digia

© 2009 - 2010 Digia Plc

- Architecture
- Activity & Intent
- Broadcast Receiver
- Service
- Content Provider

digia

© 2009 - 2010 Digia Plc

Example AIDL Specification

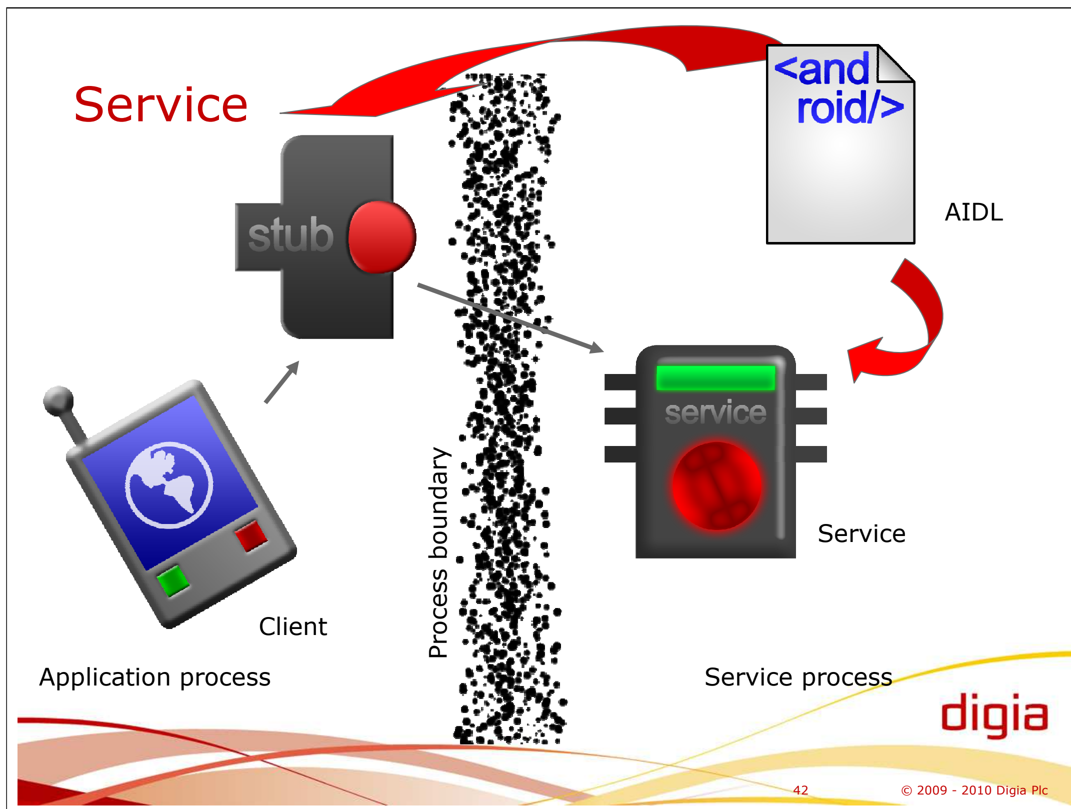
```
package servicestub;

interface ISequence {
    int getSequence();
}
```

© 2009 - 2010 Digia Plc

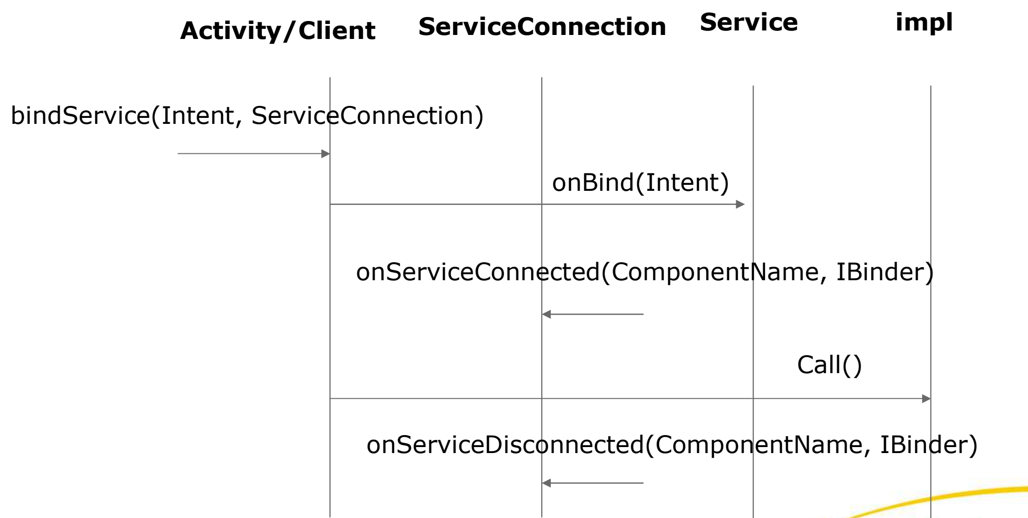
Reference:

-http://developer.android.com/guide/developing/tools/aidl.html



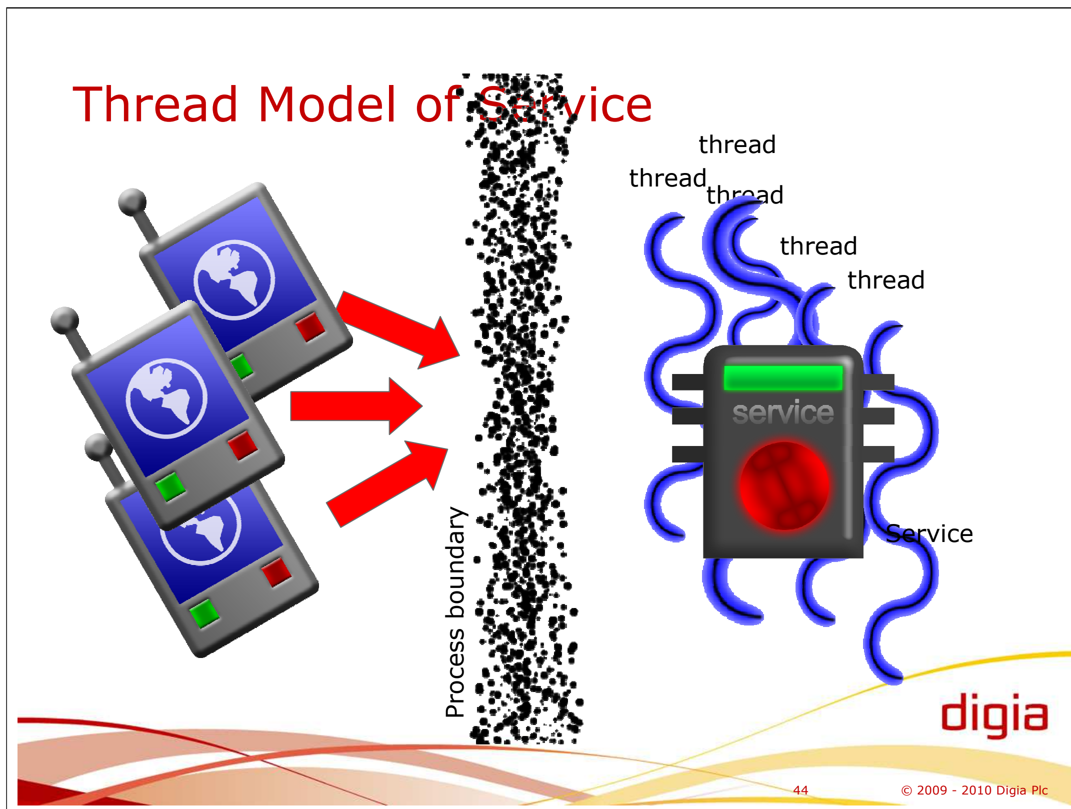
Client uses a service thru a stub that is generated from an interface specification written in AIDL language. The AIDL specification describes the interface of service.

Android provides the IPC mechanism used by the stub.



When the onServiceConnected() is called, the client gets the reference to the service implementation and the client can call the service implementation.

The onServiceDisconnected() gets called if the service implementation crashes.



The use of the interface of a service is synchronous for the client.

If there are multiple simultaneous clients to a service, the service is accessed by multiple threads. The platform creates the threads automatically in the service process to handle the requests of clients.

The service can be explicitly started (Context.startService()) by an application, which implies that the service is explicitly shut down (Context.stopService()) at some point.

Normally, service is started when the first client tries to bind to that service. Also, when the last client unbinds the service is closed automatically.

Service in XML

```
- <service android:name="com.example.android.sequenceservice.SequenceService" android:process=":service">
  - <intent-filter>
    - <!--
          These are the interfaces supported by the service, which
                        you can bind to.
      -->
      <action android:name="servicestub.ISequence" />
    </intent-filter>
  </service>
```


© 2009 - 2010 Digia Plc

Service is specified inside the application tags in AndroidManifest.xml file.

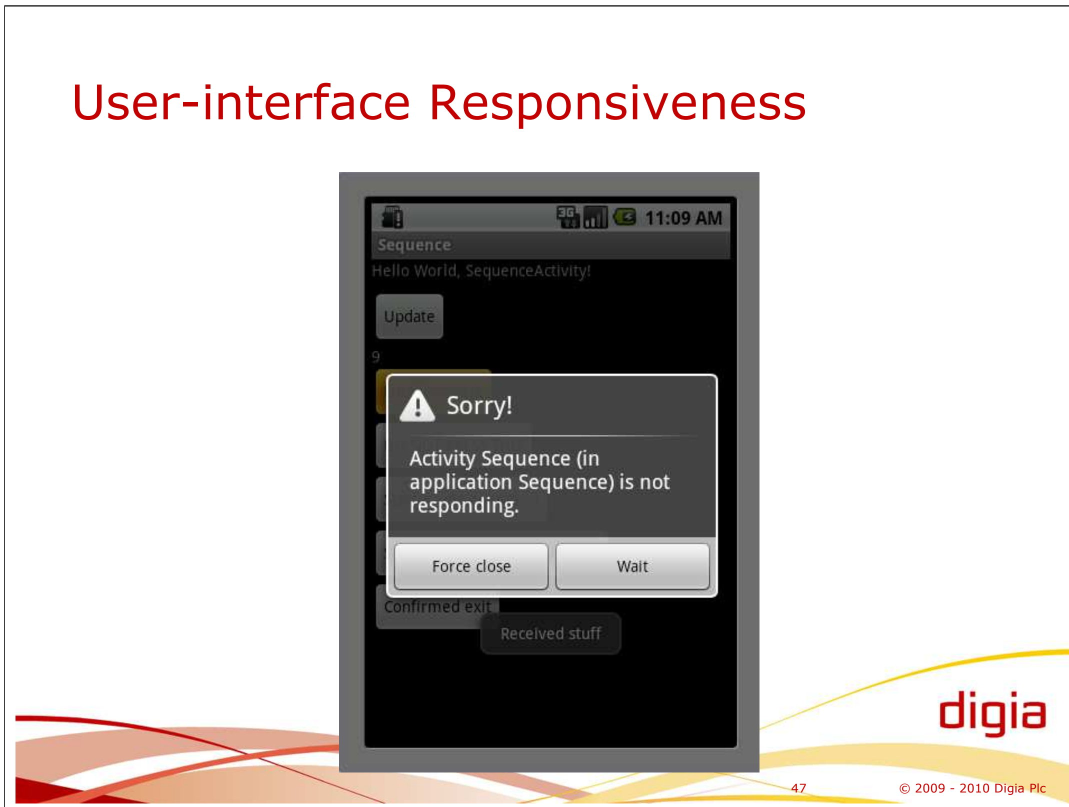
By default a service resides in the same process as the application itself. The service can be “extracted” into another process by specifying the android:process attribute of the service tag, which gives a name for the process. For clients it is transparent if the service is located in other process. Also, the platform instantiates the service automatically.

Intent filters of a service specifies the supported interfaces of that service. The action and its name attribute of intent filter is used to match the bind request of client (the intent used in binding) to the matching service.

Code Insert: Service

- AIDL specification for a service
- Implementing a service
- Defining AndroidManifest.xml for a service
- Using a service in the client side

© 2009 - 2010 Digia Plc

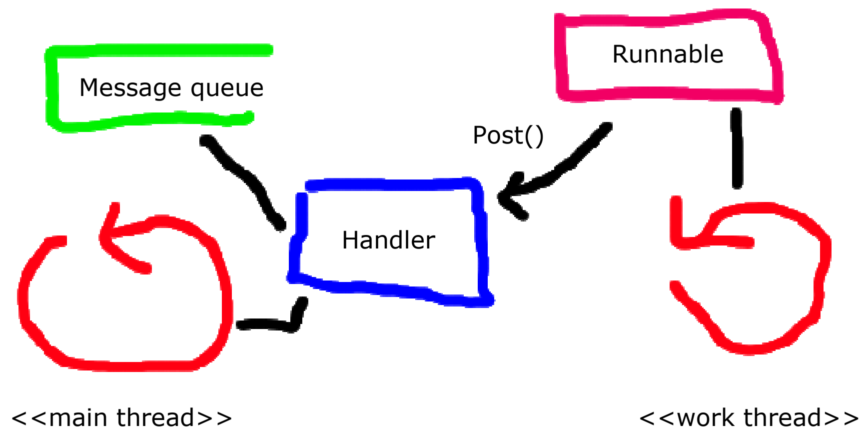


All the “onSomething” hook methods should return as soon as possible.

If the platform notices that activity is spending too much time inside the hook method, the platform shows the ANR dialog (Application is Not Responding).

All event functions (for example, onTouchEvent()) should return in 5 secs. A broadcast receiver should finish in 10 secs.

Handler “Bicycle” Diagram



digia

© 2009 - 2010 Digia Plc

The Android UI framework is not thread-safe and must always be manipulated on the main thread.

Handler, its message queue provide an easy way to implement data hand-over between threads.

A handler is associated with the thread that creates the instance of the handler. That thread processes all the messages and runnables posted into the message queue. Posting can be done from any thread. Posted runnable is then executed by the thread owning the handler (and message queue).

Messages (a message is practically integer number and Object reference) and Runnables can be posted into the message queue.

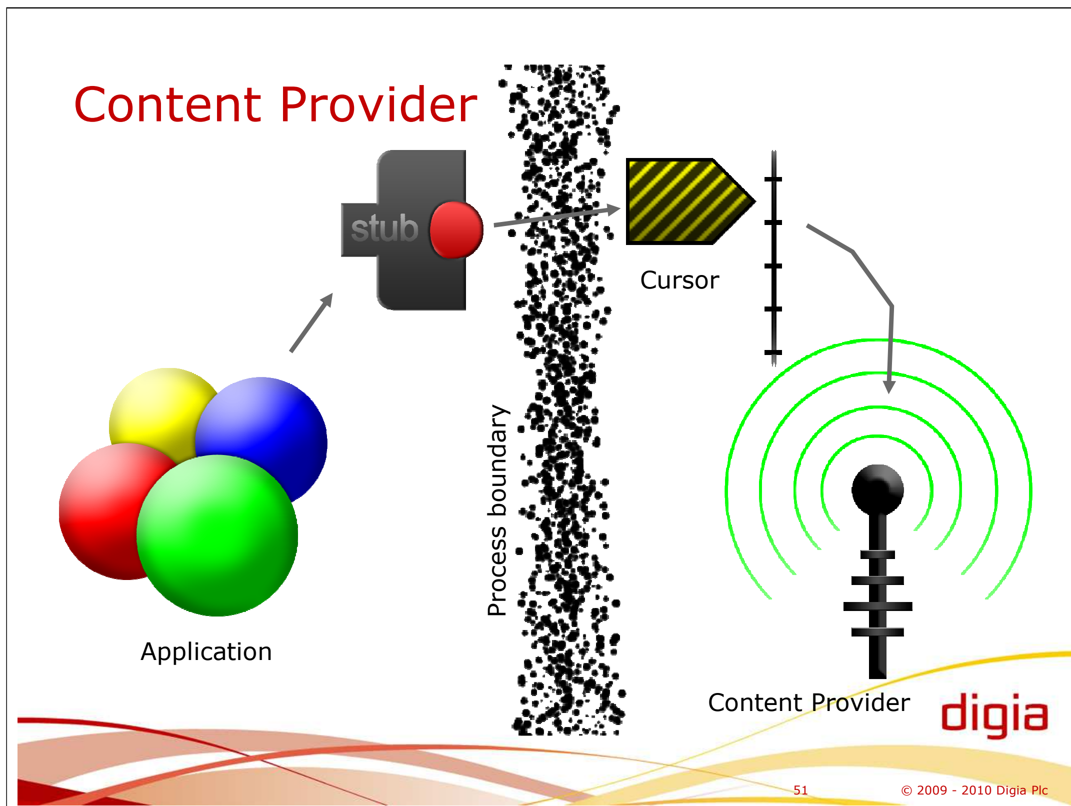
Code Insert: Worker Thread

- Creating a worker thread
- Using a handler
- Interacting between the work thread and main thread

© 2009 - 2010 Digia Plc

- Architecture
- Activity & Intent
- Broadcast Receiver
- Service
- Content Provider

© 2009 - 2010 Digia Plc



Content Provider provides a generic way to share information between the applications in the mobile device. For example, contact information is shared through a content provider to any application.

A content provider is often accessed from another process. The platform provides the means of IPC (Binder).

An application using a content provider access data through the cursor without any direct reference to the content provider. The cursor represents the results of a query from the content provider. The cursor also keeps track of position on the result, that is, it is an iterator of the result.

The cursor can also be used to modify data in the content provider.

The application gets an cursor through any activity by content resolver. The “correct” content provider is resolved based on the URI.

Content Providers in XML

```
<provider android:name="com.example.android.fibonacci.FibonacciContentProvider"
  android:authorities="com.example.android.fibonacci" />
```

© 2009 - 2010 Digia Plc

The provider tag is used inside the application tag.

The name attribute specifies the class name implementing the content provider.

The authority identifies the content provider and is the part of its URI. The client of the content provider refers to the content provider by URI, for example: content://com.example.android.fibonacci/2/10. The authority is used in intent resolution. The client

Ways to Store Data Persistently

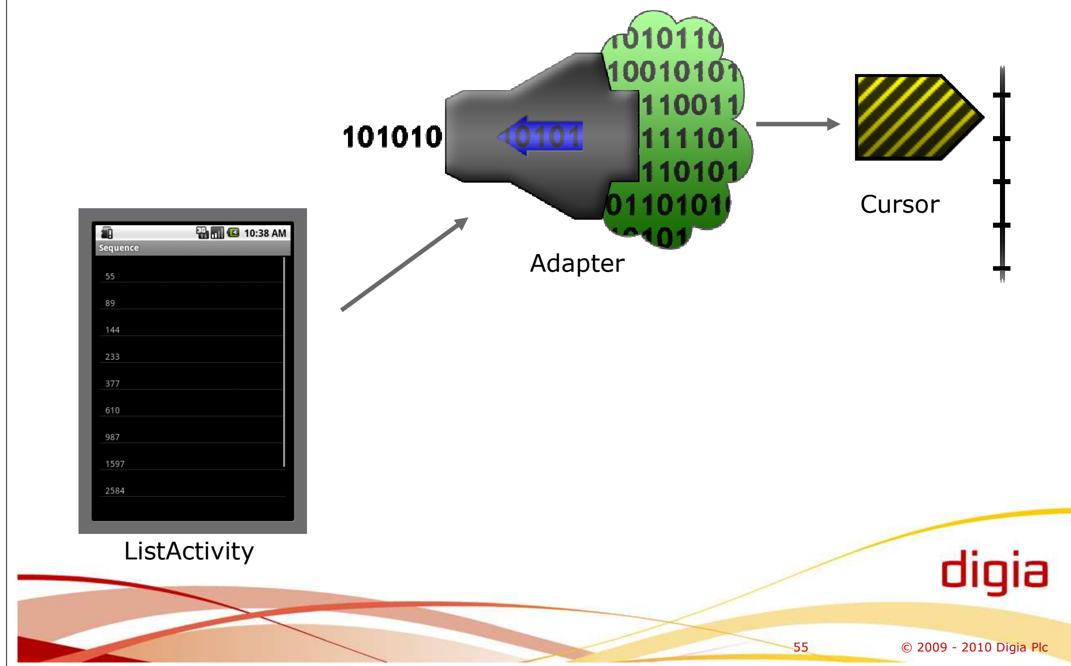
1. Flat file
2. Preferences
3. SQLite
4. Content Provider

© 2009 - 2010 Digia Plc

Code Insert: Content Provider

- Implementing a content provider
- Implementing a cursor for the content provider
- Using a cursor in the client side

© 2009 - 2010 Digia Plc



Cursor is not used directly by activities. Cursors and content providers are generic classes. Adapter provides means for displaying that generic data in a list activity.

Adapter uses a cursor that is a result from the query of particular data (set of columns of content provider data) and maps that into views of one item of the activity.

Adapter is built-in functionality of the list activity. Interface for an adapter is android.widget.Adapter. One implementation of the interface is SimpleCursorAdapter. SimpleCursorAdapter can be used to show text and icons in a list item. The user of the SimpleCursorAdapter provides the mapping from data (column name) into a view (TextView or ImageView).

Adapter can implement re-cycling of views, or graphical items, for efficiency.

Example:

```
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.alist);

        ContentResolver resolver = getContentResolver();
        final Uri uri = Uri.parse(FibonacciContentProvider.CONTENT_URI.toString() + "/10/20");
        cursor = resolver.query(uri, null, null, null, null);
        startManagingCursor(cursor);

        ListAdapter adapter = new SimpleCursorAdapter(
                this,
                R.layout.alist,
                cursor,
                new String[] {"value"},
                new int[] {R.id.AnotherTextView});

        setListAdapter(adapter);
}
```

Code Insert: ListActivity & Cursor

- Showing data from a content provider in ListActivity
- Using SimpleCursorAdapter

digia

© 2009 - 2010 Digia Plc