Android: an introduction to development

(from a mainframe development viewpoint)

Robert Harris

The independent Transaction Processing expert

dfhexpert

事务 处理 软件
资深 独立 专家

dfhexpert@gmail.com

shìwù chùlǐ ruǎnjiàn
zīshēn dúlī zhuānjiā

British Computer Society: Advanced Programming Specialist Group, November 2011

DFHEXPERT Trademarks

The DFHEXPERT logo and dfhexpert are trademarks of dfhexpert and Robert Harris in the United Kingdom.

Liability for this document

This document is for guidance only. Please confirm contents with official publications.

No liability or warranty is given as to the accuracy or consequences of using this document.

dfhexpert and/or Robert Harris make/makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose.

While every care has been taken in the compilation of this information, dfhexpert and/or Robert Harris will not be held responsible for any loss, damage or inconvenience caused as a result of inaccuracy, error or omission within this document.

English law applies to this document.

Copyright of this document

Other Trademarks

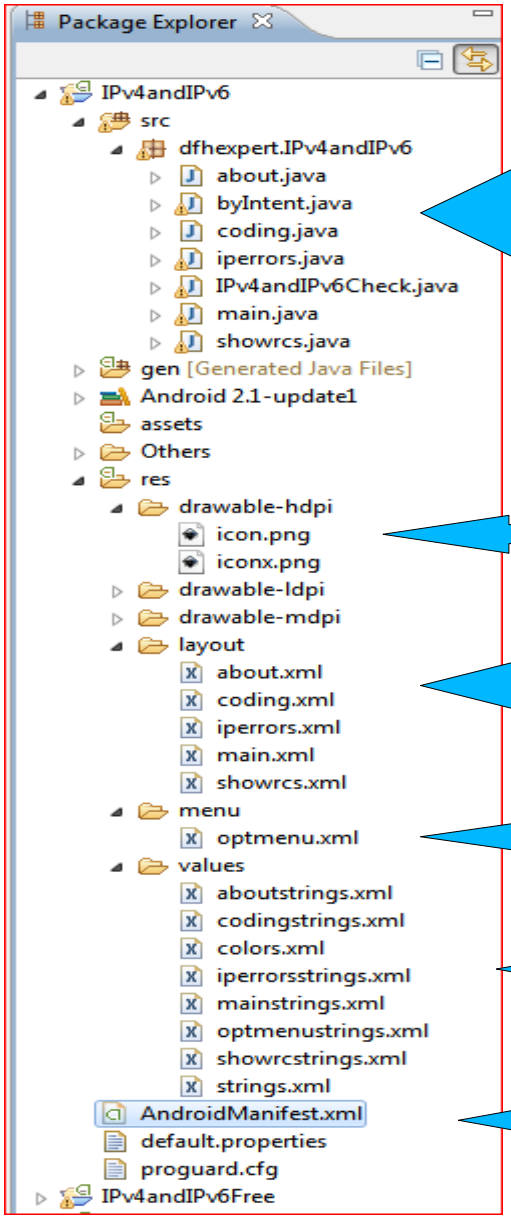
- The following are trademarks of the International Business Machines Corporation in the United States, other jurisdictions, or both: IBM, CICS Transaction Server, CICS TS, CICS, DB2, MQSeries, WebSphere, z/OS, zSeries, Parallel Sysplex. See http://www.ibm.com/legal/copytrade.shtml
- Java, JavaBeans, Enterprise Java Beans, JDBC, EJB, and all Java-based trademarks are trademarks of Sun Microsystems Inc. in the United States, other jurisdictions, or both
- Microsoft, Windows, Windows Vista, and the Windows logo are trademarks of the Microsoft Corporation in the United States, other jurisdictions, or both. See http://www.microsoft.com/about/legal/trademarks/usage/default.mspx
- Android is a trademark of Google Inc. in the United States, other jurisdictions, or both
- Other company, product and service names, and logos may be trademarks and service marks of other corporations

Android usage

- Android is an Stateless Open Source Operating System
  - Mobile telephony
  - Mobile data/web access
  - Platform independent computing
- Android runs on (in screen size order)
  - Smart phones (the Android default)
  - Padds (intermediate size)
  - Tablets (biggest size)
- Now hardware linkup with Motorola mobility
  - >50% of the market before this (August 2011)

Development Environment

- Based on the standard Eclipse IDE
  - Standard Java development environment
  - Debugging via an emulator or on a real device
- Android is OO Java
  - Override methods to do user code
  - Java methods all available for functional use
  - Android APIs to build screens, links etc.
- Android Applications
  - Use XML to define things (screen layout, literals)
  - Operate via the AndroidManifest.xml file



Code in Java Classes (a screen)

Launcher Icon

Screen layouts

Application Menu

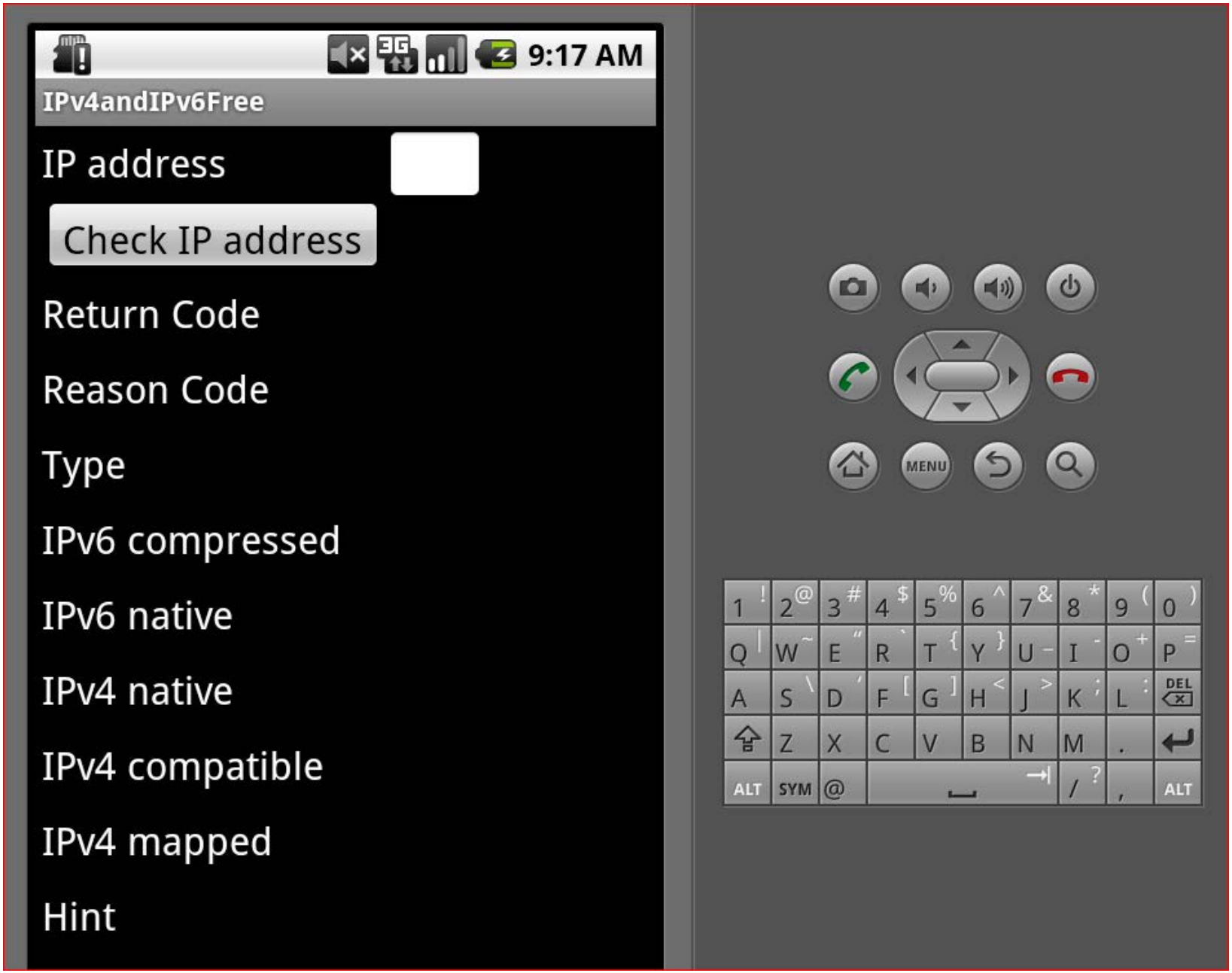
Constants

Manifest file

Android supplies build tools to generate an `.APK`

The Emulator

- Running the Package starts the Android Emulator
  - Normal
  - With debug support
- Can configure the screen size and properties
- Runs the app via a mouse and keyboard
  - Change screen orientation
  - Long and Short presses
  - Menu and Back buttons (external to device)
- Vital in development
  - catches runtime exceptions



Display (Layout) elements

- Common GUI elements
  - Literal and Input text
  - Buttons (Press, Radio, Check)
  - Icons
- Layout options
  - Scrolling (Vertical - sometimes default - Horizontal)
  - Linear
  - Table
  - List
- All specified as a Layout in XML
  - name of layout = name of the file

```
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/in_vertScroll"
    android:layout_width="wrap_content"
    android:layout_height="fill_parent"
    android:scrollbarSize="12dip">

<HorizontalScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/in_horizScroll"
    android:layout_width="wrap_content"
    android:layout_height="fill_parent"
    >
```

```
<TableLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/main"
  android:layout_width="wrap_content"
  android:layout_height="fill_parent"
  android:stretchColumns="*"
  >
```

```
 <TableRow>
   <TextView
        android:id="@+id/in_t00"
        android:text="@string/ins_ip"
        android:gravity="left|fill_vertical"
        android:padding="3dip"
        android:textColor="@color/White"
        android:layout_width="wrap_content"
        android:layout_height="@dimen/td_tv_c1h"
        android:textSize="@dimen/td_tv_c1t"
        />

   <EditText android:id="@+id/in_tip"
             android:layout_width="wrap_content"
             android:layout_height="@dimen/td_tv_c2h"
             android:background="@android:drawable/editbox_background"
             android:gravity="fill_vertical"
             android:textSize="@dimen/td_tv_c2t"
             />
   </TableRow>
```

```
<TableRow>
  <Button    android:id="@+id/inb_go"
             android:layout_width="wrap_content"
             android:layout_height="@dimen/td_tv_c1h"
             android:layout_below="@id/in_t00"
             android:layout_alignParentLeft="true"
             android:layout_marginLeft="5px"
             android:text="@string/inb_go"
             android:textSize="@dimen/td_tv_c1t"
             />
  </TableRow>
```

All items have an id:
`@+id/xx` generates a Java constant used to access it in code

`@string/thing` refers to a literal defined in another XML file

```
<string name="ab_t3">
  "Robert Harris and/or dfhexpert hopes this app is useful to you.\n \n"
  "Robert Harris希望这个程序对你有用。\n \n"
  "Email dfhexpertapps@gmail.com to contact me (especially if you find a bug!). "
  </string>
```

Positioning on the screen

- Cannot use screen co-ordinates
  - Screen size unknown
  - Orientation unknown
    - Android recommends both portrait and landscape
  - EditText sizes are dynamic
- Linear Layout
  - Align elements in a fixed direction (Vertical or Horizontal)
  - Can expand into blank areas (`weight`)
- Relative Layout
  - Align elements in a position relative to each other
  - Below, toLeftOf an earlier element (`gravity`)

Table Layout

- Forces elements into cells to ensure correct alignment in device independent fashion
- Columns can expand or contract as needed
- Put inside Vertical and Horizontal scrollers
  - So get bigger display area than the viewport
- Table structure should be maintained over device orientation change
  - Can manipulate cells, rows or columns via API
  - Dynamically insert rows containing cells with other views (Text, EditText)
  - Some APIs don't work until the screen is built

Altering Display Properties

Control Corner Shapes, Fills, Colours

Create <shape> in XML file in the res/drawable folder

```
<!--  res/drawable/rounded_edittext.xml -->

<shape xmlns:android="http://schemas.android.com/apk/res/android"
      android:shape="rectangle"
      android:padding="10dp"
      >
     <solid android:color="#FFFFFF"/>
     <corners
          android:bottomRightRadius="15dp"
          android:bottomLeftRadius="15dp"
          android:topLeftRadius="15dp"
          android:topRightRadius="15dp"
          />
     </shape>
```

Use via @drawable/file_name in the layout

```
<EditText
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:padding="5dip"
    android:background="@drawable/rounded_edittext" />
```

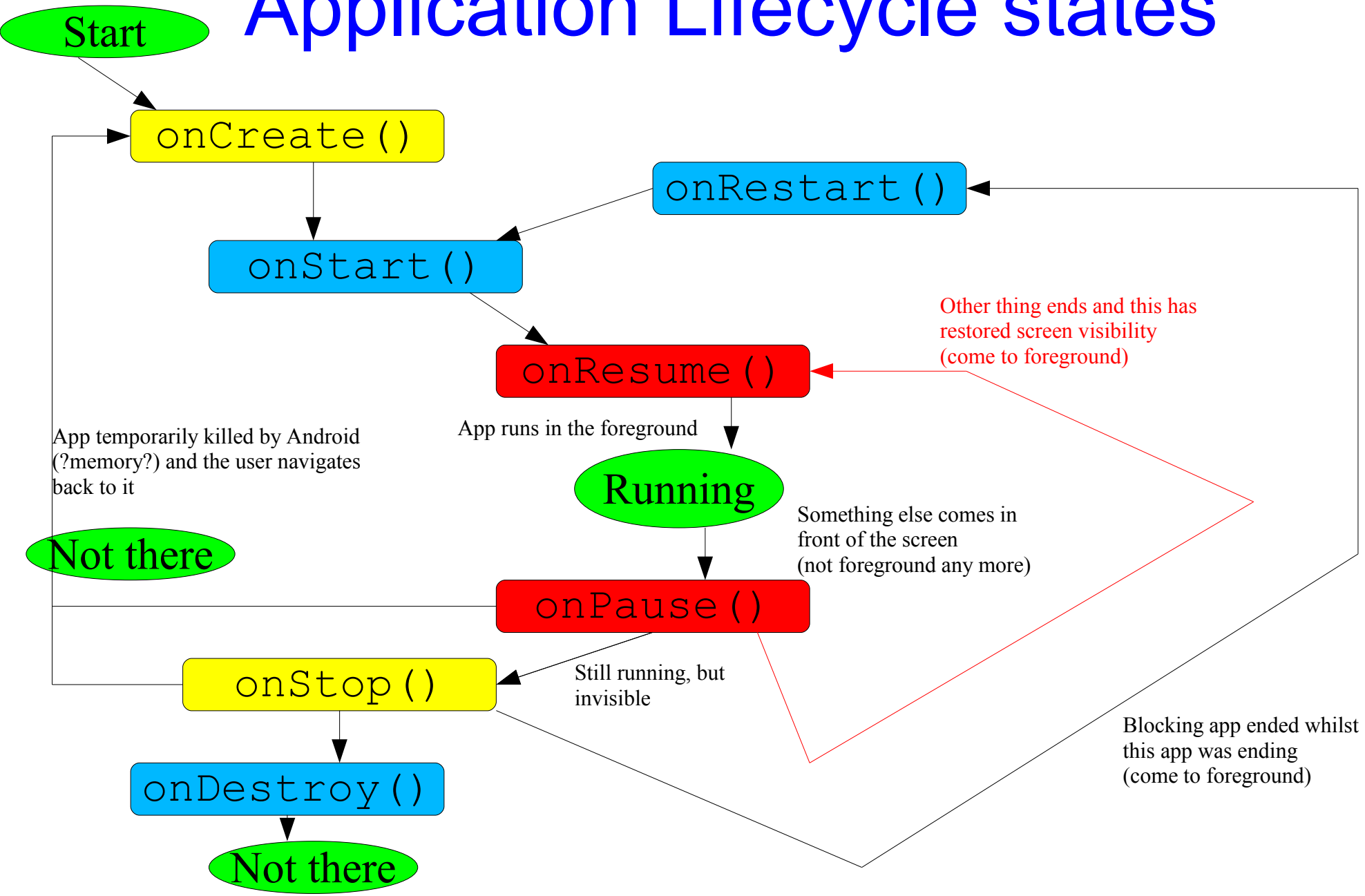
Create your own contents

- Create Java Class with the required methods
  - `onCreate` for initialisation
  - `onDraw` to display it
  - `onMeasure` to calculate the size
  - `onLayout` to compute rendering bounds
  - `onSaveInstanceState` for orientation change
  - `onPause`, `onResume` for visibility change
- Use in a layout XML file
  - <dfhexpert.myclass …… />

Apps are not programs

- Android is in control
  - Apps are rather like exits in mainframe environment
    - Apps implement methods which get called
  - You do not get to control things
    - Install and Uninstall
    - Lifecycle
    - When the screen is updated
- Multi-thread support
  - Locking and race conditions
  - Runaway task
- Only the Main thread can update the screen
  - Some APIs don't work until the screen is updated

Application Lifecycle states



The main methods to implement

- `onCreate(Bundle)` – the first method called
  - Set the screen layout
  - Initialise variables
  - Recover state (Android is stateless) from Bundle
    - Android does not save method state
- `onSaveInstanceState` – save current state
  - Screen orientation change
  - Save state in a Bundle {K=V}
    - On exit Android has cleared state and calls onCreate(Bundle)
- `onClick`, `onLongClick` – Button or something clicked
  - Do action
- `selfDestruct` – end of instance

The contents of an App Class

- App extends Android classes
  - Method callbacks to run the app
  - usually `extend Activity`
  - `implement onClickListener, onLongClickListener`
- Generally no enforcement of
  - @Override
  - private, public, protected, final
- Have your own methods in there for usage in `this`
- Package name has to be unique in the Android Marketplace
- All exported in a signed .APK file

```
package dfhexpert.IPv4andIPv6;

import java.util.* ;
import android.app.Activity;

final public class main extends    Activity
                        implements OnClickListener, OnLongClickListener
 {

    @Override
    public void onCreate(Bundle inBundle)

    public void onClick(View inview)

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent inintent)

    final public boolean onLongClick(View v)

    @Override
    final public boolean onCreateOptionsMenu(Menu menu)

    @Override
    final public boolean onPrepareOptionsMenu(Menu menu)

    @Override
    final public boolean onOptionsItemSelected(MenuItem item) {

    public boolean willIntentRun(Context context, Intent intent )

} // end of main class
```

main.java

```
87      @Override
88      public void onCreate(Bundle inBundle)
89          {
90          super.onCreate(inBundle)           ;
91          setContentView(R.layout.byobject) ;
92
93          /* Grab hold of the XML Resources */
94
95          Resources res = getResources() ;
96
97          /* Set the base global text size from the initial object layout */
98
99          if ( gTextSize == 0 )
100             {
101              TextView v = (TextView) findViewById(R.id.byObjectC1) ;
102              gTextSize  = v.getTextSize()                          ;
103             }
104
105         /* Reload the pre-rotation state if around */
106
107         if ( inBundle != null )
108             {
109               gColourObject =   inBundle.getInt(   "gColourObject" ) ;
110               gColourCN     =   inBundle.getInt(   "gColourCN"     ) ;
111               gColourPy     =   inBundle.getInt(   "gColourPy"     ) ;
112               gOrderBy      =   inBundle.getInt(   "gOrderBy"      ) ;
113               gTextSize     =   inBundle.getFloat( "gTextSize"     ) ;
114
115             }
116
117         /* Load the Object and Collective Nouns */
118
119         boolean rc = doBuildItems(res) ;
120
121         /* Build the columns */
122
123         if ( rc == true ) doBuildTable(res) ;
124
125         } /* End of oncreate method */
```

Set the Layout (Screen)

Access part of the Layout

Recover state

Build the screen

On exit, control goes back to Android and the screen is built & displayed. The class resumes in another method/callback when something happens on the screen

```
final public boolean onLongClick(View v)
 {

  // Get handle to the clipboard service

  ClipboardManager clipboard =
         (ClipboardManager) getSystemService(Context.CLIPBOARD_SERVICE) ;

  /* Grab the item id clicked and its text */

  Context       context = getApplicationContext()   ;
  TextView      vv      = (TextView) v              ;
  String        vt      = vv.getText().toString()   ;
  int           vi      = vv.getId()                ;

  switch ( vi )
  {
    case R.id.in_tac :

         /* Long click on the ac runs the examples screen  */
         /*      with those IP showing the ac backlit      */

         Intent iniperi = new Intent(this,iperrors.class) ;
         iniperi.putExtra("ac", vt)                       ;
         startActivity(iniperi)                           ;
         break                                            ;

    case R.id.in_t6c :  case R.id.in_t6n :
    case R.id.in_t4n :  case R.id.in_t4c :  case R.id.in_t4m :

         /* Long click on the IPv4/5 returns copies       */
         /*      that data value to the clipboard         */

         clipboard.setText(vt)                                    ;
         String   tt = "\"" +  vt + "\" copied to clipboard"      ;
         Toast.makeText(context,tt,Toast.LENGTH_SHORT).show()     ;
         break                                                    ;

    default : break ;
  } // End of select

  return true ;

  } /* End of OnlongClick callback method */
```

Callback method when something was Long Clicked (v is the id of the item clicked)

Decide who was clicked

Click to show another screen

Which View was clicked

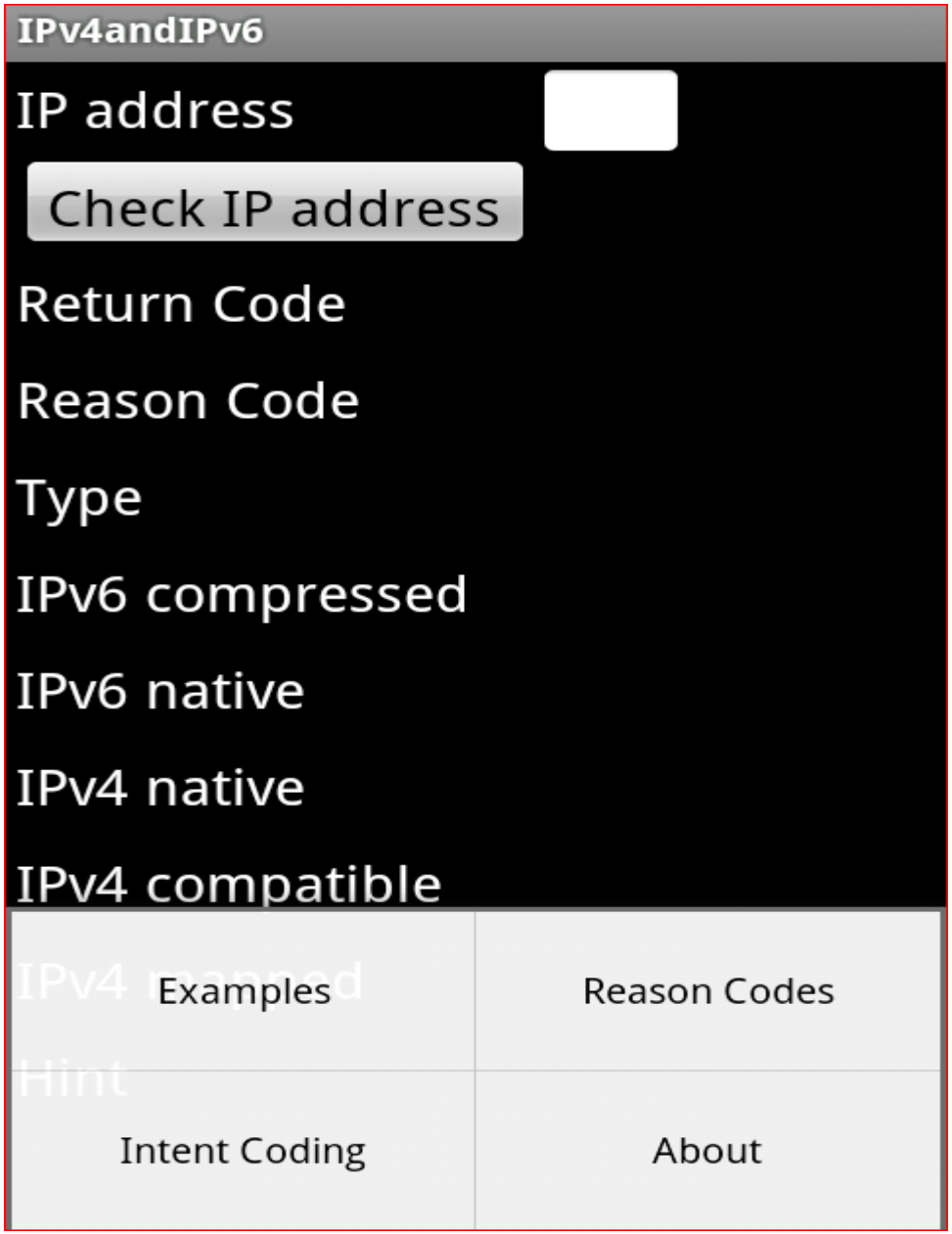
Use API to access services

Control returns back to Android. A class method is not called again until something happens on the screen

Menus

- Menus are built in – special layouts for them
  - Options (Application) Menu
    - Access via icon on device
    - Used for settings (email address, font size)
    - Data should be preserved over application instance
    - Overlays app screen
  - Context Menu
    - Under your control by press of something on the screen
    - Do something specific to the item being clicked
    - Not really a Button substitute
  - Processed by observing the id of the menu item clicked

```
<!--  Application Menu layout -->

<menu  xmlns:android="http://schemas.android.com/apk/res/android">

  <!--  Show Samples -->

  <item android:id="@+id/mSamples"
        android:title="@string/mia_Samples"
        android:enabled="true"
        android:visible="true"
        />

  <!--  Show the reason code samples  -->

  <item android:id="@+id/mShowRCs"
        android:enabled="true"
        android:title="@string/mia_ShowRCs"
        android:visible="true"
        />

   <!-- Show the Coding tabs -->

  <item android:id="@+id/mCoding"
        android:enabled="true"
        android:title="@string/mia_Coding"
        android:visible="true"
        />

  <!-- general About tabs -->

  <item android:id="@+id/mAbout"
        android:enabled="true"
        android:title="@string/mia_About"
        android:visible="true"
        />

  </menu>
```



Showing a different screen

- You start the App in a class which manages a screen/layout
- When a Button is pressed, you want to show another screen
    - `onPause` method gets invoked to show being obscured
- New Screen
    - has it's own class
    - overlays (higher Z order) current screen
    - exits using the standard device back button
        - returns to previous screen/layout
        - control returns to the class which owns the previous screen
        - usually previous screen is active (with data)
- Initial screen now visible again
    - `onResume` method run before device's screen updated

Running another class

- Run another class in the same App (one of yours)
  - Screen and methods local to it
  - Stacked on top of current class
    - which is still running but obscured
  - Go back (down the Z order) by the back key
- Run someone else's App
- Run an Android built-in service
  - Telephony
  - Database (mySQL builtin)
  - Mobile data access (web pages)

Intents

- Intents run something else
  - Another class (=screen) in your app
  - Android built-in service
    - telephony
    - read web page
- Intents contain the indirected name of something to run
  - Maps to a package.class in AndroidManifest.xml
    - Scan of all installed apps to find it if not yours
- Used both to invoke (with data) and return (+data)
  - Contains user data as input
- Start via API, get returns in `onActivityResult` callback
- A running Intent is an Activity

Manifest Android Control

- AndroidManifest.xml is key to running an App
  - Versioning indication (Android Marketplace)
  - Launcher information
  - What is the minimum Android release required
  - What device it will run on
    - Physical keyboard? Camera?
  - Definition of the App and it's first class
  - Other classes (screens) that run
  - Service provided by your app to others
  - What parts of Android are used
    - IP access? Camera?
- Vital to get right

Manifest control

```
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
          package="dfhexpert.IPv4andIPv6"
          android:versionCode="100"
          android:versionName="1.0.0">

    <uses-sdk android:minSdkVersion="7"
              android:targetSdkVersion="7"
              />

    <uses-configuration android:reqKeyboardType="undefined"
                        android:reqHardKeyboard="false"
                        android:reqNavigation="undefined"
                        android:reqFiveWayNav="true"
                        />

     <uses-configuration android:reqKeyboardType="undefined"
                         android:reqHardKeyboard="false"
                         android:reqNavigation="nonav"
                         android:reqFiveWayNav="false"
                         />
```

Version Control

Min Android level

What hardware is needed to run

Google's Android Marketplace shows the devices on which the app will run

Manifest defines the App

```
<!-- The Main application -->

<application android:icon="@drawable/icon"
             android:label="@string/app_name"
             android:allowTaskReparenting="false"
             android:description="@string/app_description"
             android:persistent="false"
             >

    <!-- The main screen -->

    <activity android:name="dfhexpert.IPv4andIPv6.main"
              android:label="@string/app_name"
              android:alwaysRetainTaskState="false"
              android:enabled="true"
              android:launchMode="standard"
              android:screenOrientation="unspecified"
              android:stateNotNeeded="true"
              android:windowSoftInputMode="stateAlwaysHidden"
              >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>

        </activity>
```

Class to run

First class for the App

Can be user invoked from the Launcher

Manifest for another screen in the App

```
<!--  Activity for the About screen -->

<activity android:name="dfhexpert.IPv4andIPv6.about"
          android:label="@string/ab_about"
          android:alwaysRetainTaskState="false"
          android:enabled="true"
          android:launchMode="standard"
          android:screenOrientation="unspecified"
          android:stateNotNeeded="true"
          android:windowSoftInputMode="stateAlwaysHidden"
          >

   <intent-filter>
       <action android:name="android.intent.action.VIEW" />
       <category android:name="android.intent.category.LAUNCHER" />
       </intent-filter>

   </activity>
```

Normal screen

Manifest for running function by request from someone else

```
<!-- Intent receiving processing, no screen, background -->

<activity android:name="dfhexpert.IPv4andIPv6.byIntent"
          android:label="@string/app_name"
          android:alwaysRetainTaskState="false"
          android:enabled="true"
          android:launchMode="standard"
          android:screenOrientation="unspecified"
          android:stateNotNeeded="true"
          android:windowSoftInputMode="stateAlwaysHidden"
          >
   <intent-filter>
       <action   android:name="dfhexpert.IPv4andIPv6.action.CHECKIP" />
       <category android:name="android.intent.category.DEFAULT"      />
       <data     android:scheme="ip"  android:host="dfhexpert"       />
       </intent-filter>

   </activity>
```

Class to run

Name of Intent

Data for Intent is in URI format of `scheme://host/data`

Creating and running an Intent

For local class

```
Intent insam = new Intent(this,iperrors.class) ;
       insam.putExtra("max", "4") ;
       startActivity(insam) ;
```

For remote class

```
Intent ipintent = new Intent() ;
    ipintent.setAction("dfhexpert.IPv4andIPv6.action.CHECKIP") ;
    ipintent.addCategory("android.intent.category.DEFAULT")    ;

    Uri    ipuri = Uri.parse("ip://dfhexpert/"+ip+"/")   ;
    ipintent.setData(ipuri)                              ;

    // Schedule the Intent to do the checking

    int iprc = 100                                       ;
    startActivityForResult(ipintent,iprc)                ;

    // Processing continues in the onActivityResult method
```

Conclusion

- Android has a significant learning curve
- Good integration with Eclipse
- Development running in emulator and real device
- Layout is complex and sometimes unintuitive
- AndroidManifest.xml is critical and difficult to get right
- Strangely both alien and familiar to mainframe transaction processing developers
- Lots of good advice and examples on the web
- Mainframe 3270/LU2 applications can be refaced for Android mobile usage quite directly

Thank You

dfhexpert thanks you for coming to this talk

Contact Robert Harris on dfhexpert@gmail.com

Connect via www.linkedin.com/in/dfhexpert

Admire his android apps by searching in the Google app market for 'pub:dfhexpert'

谢谢　再见