Intro to Android Development 2

Accessibility Capstone

Nov 22, 2011

Outline for Today

- Application components
  - Activities
  - Intents
  - Manifest file
- Visual user interface
  - Creating a user interface
  - Resources
- TextToSpeech

Helpful Links

- Developer Guide / Getting Started
  - http://developer.android.com/guide/index.html
- API Reference
  - http://developer.android.com/reference/packages.html

Application Processes

- An app runs in its own linux process
- Process is shut down when app is not needed or system resources are needed
- A process runs in its own VM

Application Components

- An application can use pieces of other applications
- *Application component* - piece of application that can run independently.
- System starts app process when an app component is run
- No single entry point – no “main()”

A kind of Component: an Activity

- An application component
- A visual UI (a screen) for one task

A kind of Component: an Activity

- An application component
- A visual UI (a screen) for one task
- Examples
  - List of menu items
  - Display of photographs with captions
  - Select contact from list
  - Compose text message to contact
  - Change settings
  - Review old messages

text messaging app

Activating Components: Intents

- Holds content of async message used to activate an app component
- Holds action to be done and data to act on
- To start an activity from your app, use Context.startActivity(Intent intent)
- Intent intent = new Intent(this, ActivityToLaunch.class);
  startActivity(intent);

Declaring components: the Manifest file

- Every app has a AndroidManifest.xml file

    To learn about XML files, see the W3Schools XML tutorial.

- Where app components are declared
- Set capabilities and permissions
- Include libraries
- Name Java package

    Unique identifier for an app

    “edu.washington.cs.MyApp”

Manifest file

```
<?xml version="1.0" encoding="utf-8"?>
<manifest . . . >
    <application . . . >
        <activity android:name="com.example.project.FreneticActivity"
                  android:icon="@drawable/small_pic.png"
                  android:label="@string/freneticLabel"
                  . . .  >
        </activity>
        . . .
    </application>
</manifest>
```

Manifest file

```
<?xml version="1.0" encoding="utf-8"?>
<manifest . . . >
    <application . . . >
        <activity android:name="com.example.project.FreneticAct
                  android:icon="@drawable/small_pic.png"
                  android:label="@string/freneticLabel"
                  . . .  >
        </activity>
        . . .
    </application>
</manifest>
```

component declaration

Manifest file

```
<?xml version="1.0" enc
<manifest . . . >
    <application . . . >
        <activity android:name="com.example.project.FreneticActivity"
                  android:icon="@drawable/small_pic.png"
                  android:label="@string/freneticLabel"
                  . . .  >
        </activity>
        . . .
    </application>
</manifest>
```

Your Activity class's name after the app package name

Visual User Interfaces

- An activity has a visual UI
- Activity comes with built in window, can contain hierarchy of views.
- View – a rectangular area on the screen with contents
- Parent view contain and organize layout of their children
- Examples: TextView, Button, etc.

Creating a User Interface

- *Procedural* – using code, dynamic
- *Declarative* – no code, using XML

```
package edu.washington.cs.capstone;
import android.app.Activity;
import android.os.Bundle;public

class DesigningUIAct extends Activity {
	/** Called when the activity is first created. */
	@Override
	public void onCreate(Bundle savedInstanceState) {
		TextView tv = new TextView(this);
	tv.setText("Hello, Android");
	setContentView(tv);
	}
}
```

Procedural or declarative?

Creating a User Interface

- *Procedural* – using code, dynamic
- *Declarative* – no code, using XML

```
package edu.washington.cs.capstone;
import android.app.Activity;
import android.os.Bundle;public

class DesigningUIAct extends Activity {
        /** Called when the activity is first created. */
        @Override
        public void onCreate(Bundle savedInstanceState) {
                TextView tv = new TextView(this);
         tv.setText("Hello, Android");
         setContentView(tv);
        }
}
```

Procedural!

Creating a User Interface

- *Procedural* – using code, dynamic
- *Declarative* – no code, using XML

```
package edu.washington.cs.capstone;
import android.app.Activity;
import android.os.Bundle;public

class DesigningUIAct extends Activity {
        /** Called when the activity is first created. */
        @Override
        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.main);
    }
}
```

Declarative!

Layouts as Resources

- Resources – non-code information a program needs. Localized string, bitmap, etc.
- gen/R.java – automatically generated resource ID's
- res/layouts/main.xml - default, specifies activity UI.
- Layouts: LinearLayout, FrameLayout, RelativeLayout, TableLayout, etc.

main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    />
</LinearLayout>
```

main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/re
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    />
</LinearLayout>
```

Layout objects: containers for child objects and a behavior of how to position them

main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/andr
oid"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    />
</LinearLayout>
```

XML namespace for android.
Define once on first elemenet.

main.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    />
</LinearLayout>
```

refers to string in res/values/strings.xml

Text To Speech

- Check for language resources
- Create an instance of the TTS engine
- Speak!
  - Set the language
  - Add *utterances* to TTS queue

Check for Language Resource

```
Intent checkIntent = new Intent();
checkIntent.setAction
        (TextToSpeech.Engine.ACTION_CHECK_TTS_DATA);
startActivityForResult(checkIntent,
                                        MY_DATA_CHECK_CODE);
```

Check for Language Resource

```
Intent checkIntent = new Intent();
checkIntent.setAction
        (TextToSpeech.Engine.ACTION_CHECK_TTS_DATA);
startActivityForResult(checkIntent,
                                        MY_DATA_CHECK_CODE);
```

Create an intent object to start the activity that checks for TTS resources. This activity is a component of another application.

Check for Language Resource

```
Intent checkIntent = new Intent();
checkIntent.setAction
        (TextToSpeech.Engine.ACTION_CHECK_TTS_DATA);
startActivityForResult(checkIntent,
                                        MY_DATA_CHECK_CODE);
```

Set the intent action. The Intent is an object carrying data for an asynchronous message. The action tells the system what component to launch.

Check for Language Resource

```
Intent checkIntent = new Intent();
checkIntent.setAction
        (TextToSpeech.Engine.ACTION_CHECK_TTS_DATA);
startActivityForResult(checkIntent,
                                    MY_DATA_CHECK_CODE);
```

A member variable we create, used to identify this particular call to *startActivityForResult()* in the handler.

Create an instance of the TTS Engine

```
private TextToSpeech mTts;
protected void onActivityResult(int requestCode,
                                        int resultCode, Intent data) {
  if (requestCode == MY_DATA_CHECK_CODE) {
    if (resultCode ==
                TextToSpeech.Engine.CHECK_VOICE_DATA_PASS) {
      // success, create the TTS instance
      mTts = new TextToSpeech(this, this);
    } else {
      // missing data, install it
      Intent installIntent = new Intent();
      installIntent.setAction(
                  TextToSpeech.Engine.ACTION_INSTALL_TTS_DATA);
      startActivity(installIntent);
    }
  }
}
```

Create an instance of the TTS Engine

An activity has finished

```
private TextToSpe
protected void onActivityResult(int requestCode,
                                int resultCode, Intent data) {
    if (requestCode == MY_DATA_CHECK_CODE) {
        if (resultCode ==
                    TextToSpeech.Engine.CHECK_VOICE_DATA_PASS) {
            // success, create the TTS instance
            mTts = new TextToSpeech(this, this);
        } else {
            // missing data, install it
            Intent installIntent = new Intent();
            installIntent.setAction(
                        TextToSpeech.Engine.ACTION_INSTALL_TTS_DATA);
            startActivity(installIntent);
        }
    }
}
```

Create an instance of the TTS Engine

```
private TextToSpeech mTts;
protected void onActivityResult(int requestCode,
                                int resultCode, Intent data) {
  if (requestCode == MY_DATA_CHECK_CODE) {
    if (resultCode ==
            TextToSpeech.Engi
      // success, create the TTS
      mTts = new TextToSpeech(this, this);
    } else {
      // missing data, install it
      Intent installIntent = new Intent();
      installIntent.setAction(
                TextToSpeech.Engine.ACTION_INSTALL_TTS_DATA);
      startActivity(installIntent);
    }
  }
}
```

Checks if the TTS data check activity is the one that finished

Create an instance of the TTS Engine

```
private TextToSpeech mTts;
protected void onActivityResult(int requestCode,
                                 int resultCode, Intent data) {
    if (requestCode == MY_DATA_CHECK_C
        if (resultCode ==
                TextToSpeech.Engine.CHECK_VOICE_DATA_PASS) {
            // success, create the TTS instance
            mTts = new TextToSpeech(this, this);
        } else {
            // missing data, install it
            Intent installIntent = new Intent();
            installIntent.setAction(
                    TextToSpeech.Engine.ACTION_INSTALL_TTS_DATA);
            startActivity(installIntent);
        }
    }
}
```

yes! TTS is ready to go

Speak!

```
public class TTSExampleAct extends Activity
                                implements OnInitListener {
…
public void onInit(int arg0) {
      mTts.setLanguage(Locale.US);
      String myText1 = "Did you sleep well?";
      String myText2 = “It's time to wake up!";
      mTts.speak(myText1, TextToSpeech.QUEUE_FLUSH, null);
      mTts.speak(myText2, TextToSpeech.QUEUE_ADD, null);
}
….

}
```

Speak!

```
public class TTSExampleAct extends Activity
                                        implements OnInitListener {
…
@Overridepublic void onInit(int arg0) {
      mTts.setLanguage(Locale.US);
      String myText1 = "Did you sleep well?";
      String myText2 = “It's time to wake up!";
      mTts.speak(myText1, TextToSpeech.QUEUE_FLUSH, null);
      mTts.speak(myText2, TextToSpeech.QUEUE_ADD, null);
}
….

}
```

Speak!

```
public class TTSExampleAct extends Activity
                                        implements OnInitListener {
…
@Overridepublic void onInit(int arg0) {
      mTts.setLanguage(Locale.US);
      String myText1 = "Did you sleep well?";
      String myText2 = “It's time to wake up!";
      mTts.speak(myText1, TextToSpeech.QUEUE_FLUSH, null);
      mTts.speak(myText2, TextToSpeech.QUEUE_ADD, null);
}
….

}
```

what’s the difference between QUEUE_ADD and QUEUE_FLUSH?