Complex User Input

CS 2046
Mobile Application Development
Fall 2010
Announcements

• HW1 due Monday, 11/1, at 11:59pm
  – See newsgroup for more clarifications.

• If you’re stuck:

• http://developer.android.com/resources/tutorials/notepad/index.html
Intro of the Day – App Widgets

• Miniature views that can be embedded in other applications
  – Usually, home screen

• Consist of:
  – AppWidgetProviderInfo
    • XML metadata
  – AppWidgetProvider
    • Program logic
  – View layout

AppWidgetProvider Info

- Defines properties of the App Widget
  - Minimum size, layout, update frequency

```xml
<appwidget-provider
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:minWidth="294dip"
  android:minHeight="72dip"
  android:updatePeriodMillis="86400000"
  android:initialLayout="@layout/example_appwidget"
  android:configure="com.example.android.ExampleAppWidgetConfigure">
</appwidget-provider>
```
AppWidgetProvider

• Misnomer; really a BroadcastReceiver
  – Details of receiver are abstracted away

• onUpdate: Called according to updatePeriodMillis.
  – This is where most program logic goes.
  – For non-updating widgets (buttons), called once on creation.

• onDeleted: When an instance is deleted.

• onEnabled: When the first instance is created.

• onDisabled: When the last instance is deleted.
App Widget Layout

• Like other layouts, goes in res/layout
  – UI guidelines:

• Difference – layouts are based on RemoteViews, which only support:
  – FrameLayout, LinearLayout, RelativeLayout
  – AnalogClock, Button, Chronometer, ImageButton, ImageView, ProgressBar, TextView
• Declare the App Widget in AndroidManifest.xml:

```xml
<receiver android:name="ExampleAppWidgetProvider">

    <intent-filter>
        <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
    </intent-filter>

    <meta-data android:name="android.appwidget.provider"
        android:resource="@xml/example_appwidget_info" />

</receiver>
```
More on App Widgets

• For more, see:
    – How to create configuration Activities
    – More example code and guidelines

• Some examples (with code):
  – App Widget API Demo
  – Wiktionary Sample
Recap

• Covered many basic UI elements
  – Widgets, layouts, menus, dialogs
  – Event handling
    • How do we respond to clicks, or other touch events?

• Next step – what other kinds of input are available to the user, and how do we process them?
Keyboard Input

• For standard widgets, keyboard input is automatic
  – i.e. EditText: Launches on-screen keyboard if needed

• What if we want to take input from keys?
  – Two cases:
    • Optional for phones with hardware keyboards
      – e.g. shortcuts
    • Required for program
      – e.g. a crossword puzzle program
Optional Keyboard Input

• Activity/view-wide: onKeyDown

```java
private boolean onKeyDown(int keyCode, KeyEvent event) {
    switch (keyCode) {
        case KeyEvent.KEYCODE_X:
            // Handle X key
            return true;
    }
    return false;
}
```

• KeyEvent allows for more complex input
  – i.e. multiple keys, held keys
Required Keyboard Input

• Easiest way:
  – Extend EditText
  – Override onKeyDown to capture key events
  – Override onDraw to make it look however you want

• Lets EditText handle hiding/showing the software keyboard when necessary
  – This is a notoriously tricky task on Android
Customized IME

• Have ability to control the type of on-screen keyboard that appears.

• For EditText – use android:inputType flags
  – e.g. textEmailAddress will include @ key without having to press Alt.

• Many more customizations – see:
  http://developer.android.com/resources/articles/on-screen-inputs.html
Touch Screen for Custom Views

• Override onTouchEvent(MotionEvent e)
  – For gestures, we just passed this event elsewhere

• Call e.getX() and e.getY() to get coordinates
  – Top left is (0, 0)
  – Bottom right is (getHeight(), getWidth())

• Common task – drawing a custom grid
  – Need to map coordinates to location in grid
  – If grid width = w and height = h, just do e.getX()/w and e.getY()/h
Speech Recognition

• Another method of working around difficulty of keyboard input.

• Requires an app installed on the phone which responds to a Recognizer Intent.
  – Many phones come with Google Voice Search
  – Emulator, unfortunately, does not.

• Code from API demos:
Accepting Speech Input

• Step 1: Check if recognition is possible

```java
PackageManager pm = getPackageManager();
List<ResolveInfo> activities = pm.queryIntentActivities(
    new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH), 0);
if (activities.size() != 0) {
    // Speech recognition enabled
} else {
    // Speech recognition disabled
}
```
Accepting Speech Input

• Step 2: Request recognition

```java
Intent intent =
    new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL,RecognizerIntent.LANGUAGE_MODEL_FREE_FORM);
intent.putExtra(RecognizerIntent.EXTRA_PROMPT, "Speak");
startActivityForResult(intent, UNIQUE_CODE);
```
Accepting Speech Input

• Step 3: Process results

```java
if (requestCode == UNIQUE_CODE && resultCode == RESULT_OK)
{
    ArrayList<String> matches = data.getStringArrayListExtra(
        RecognizerIntent.EXTRA_RESULTS);
    // Process matches...
}
```
Accelerometer

• Detect orientation and motion of device.

• CAN be tested on Android emulator!
  – (not by shaking the window)
  – See SensorSimulator for a program which lets you simulate sensor data.
    • Unfortunately, requires use of a deprecated API, but the differences are small.
Basic Approach

• Access the SensorManager service through getSystemService(Context.SENSOR_SERVICE).

• Register a listener for the sensor you wish to use with SensorManager’s registerListener() method.

• Handle events on the onSensorChanged method of the listener.
Other Sensors

• Same approach works for other sensors – the values array changes based on the type.

• From Sensor class:
  – Sensor.TYPE_MAGNETIC_FIELD
  – Sensor.TYPE_LIGHT
  – Sensor.TYPE_PROXIMITY
  – Sensor.TYPE_TEMPERATURE
  – and more…
Device Orientation

• Can get orientation from sensors, but this is deprecated.

• Instead, use SensorManager.getOrientation()

• 3D coordinates = Linear Algebra
  – Out of the scope of the course, but see http://android-developers.blogspot.com/2010/09/one-screen-turn-deserves-another.html if you’re interested.
Information Storage

CS 2046
Mobile Application Development
Fall 2010
Storing Information

• Many methods available for storing information
• Appropriate method depends on:
  – Type of information
  – Who needs to access it
Application Preferences

- Preferences is a bit of a misnomer
  - Can store preferences
  - Really, stores arbitrary, persistent key-value pairs

- Uses `SharedPreferences` class

- Thread-safe access model
Accessing Shared Preferences

• Obtain an instance with `Context.getSharedPreferences(String name, int mode)`
  – Name is unique per application
  – Mode is one of:
    • MODE_PRIVATE
    • MODE_WORLD_READABLE
    • MODE_WORLD_WRITEABLE

• Alternative: If just one preferences file for Activity, can call `Activity.getPreferences(int mode)`.
Reading Preferences

• Reading preferences is simple, given a SharedPreferences object (prefs).
  – prefs.get<TYPE>(String key, <TYPE> default)
    • If key doesn’t exist, returns default
    • If key exists but is of wrong type, throws ClassCastException
Writing Preferences

• Writing is more complex – uses transactions.

```java
SharedPreferences.Editor editor = prefs.edit();
editor.put*(key, value);
editor.remove(key);
editor.commit();
```

• All changes are committed atomically when `commit()` is called – enables thread safety.
Modifying Preferences

• What about actual preferences that we want the user to modify?

• Solution: XML and PreferenceActivity

• Pros:
  – Define preferences in a resource instead of in code
  – Consistent behavior with rest of platform
XML Preference File

• Place in res/xml/
• Root tag: <PreferenceScreen>
• Categories go in <PreferenceCategory>
• Preferences contain:
  – key – name of preference (unique ID)
  – title – plaintext name of preference
  – summary – more details
• Types of preferences:
  – CheckboxPreference, EditTextPreference, ListPreference
    • Each has additional attributes
Preference Activity

• Modifying preferences is easy:

```java
public class PreferencesFromXml extends PreferenceActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        addPreferencesFromResource(R.xml.preferences);
    }
}
```
Storing files

• Suppose preferences aren’t enough, and we need to store some kind of file on the system.

• Three choices
  – Simplest – need a read-only file bundled with app.
  – Need a (small) file private to your application
  – Need a (possibly large) file which may be useful for other applications.
Raw Resources

• Case 1: Bundled file with application

• Place in res/raw/<filename>.ext

• Access with:

getResources().openRawResource(R.raw.<filename>)

• Returns an InputStream of the file.
Internal Storage

• By default, files stored internally are private to your application.
  – Other applications (and non-rooted phone users) cannot tamper with them.
  – Will be removed when app is uninstalled.

• Call Context.openFileOutput() for writing.
• Call Context.openFileInput() for reading.
Next time...

• How do we store potentially large files that we want to share with other applications (and the user)?

• How do we store tabular data?