Introduction

Android API's provide a way to implement custom input methods, which can be created using different forms, including inputs based on drawing or speaking. This article takes on the subject of implementing a sample input method that allows a user to write with stroke gestures, the letters used in the Graffiti alphabet from the Palm OS on PDA devices. You can familiarize yourself with the basics of this handwriting recognition system at the following Wikipedia page - http://en.wikipedia.org/wiki/Graffiti_(Palm_OS).

Input method service

First look at the InputMethodService class, which is a basic framework for creating your own input method. It provides many useful tools for creating your own input method, especially key-based methods. To begin, you will need to create a class that extends InputMethodService and register it in your Manifest file, with a proper permission and intent filter. The service class should return the view of the input area from the overridden onCreateInputView method. Manifest declaration should also contain a pointer to the input method configuration file, which you can leave empty for now. Please note that after installing your own input method on the device, you have to set it to default in the system settings to start using it.

```
<service
  android:name="com.samsung.penboard.Penboard"
  android:permission="android.permission.BIND_INPUT_METHOD" >
  <intent-filter>
    <action android:name="android.view.InputMethod" />
  </intent-filter>

  <meta-data
  android:name="android.view.im"
  android:resource="@xml/method" />
</service>
```

[Code 1] Service declaration in AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<input-method xmlns:android="http://schemas.android.com/apk/res/android" >

</input-method>
```

[Code 2] XML configuration file for input method (xml/method.xml)

```
public class Penboard extends InputMethodService {
  private PenboardView mPenboardView;

  @Override
  public View onCreateInputView() {
    PenboardView penboardView = (PenboardView)
getLayoutInflater().inflate(R.layout.penboard, null);
    mPenboardView = penboardView;
```

```
    return penboardView;
  }
}
```

[Code 3] Main service class skeleton with the method returning the input area view

As you can see, the input area view is being inflated from XML during creating the input view.

View of the input area

Let’s look at the input area view layout first.

```
<com.samsung.penboard.PenboardView
xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent" >

  <LinearLayout
    android:id="@+id/left_panel"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:orientation="vertical" >

    <Button
      android:id="@+id/symbols_btn"
      android:layout_width="85dp"
      android:layout_height="85dp"
      android:layout_marginBottom="5dp"
      android:layout_marginTop="10dp"
      android:background="@drawable/symbols" />

    <Button
      android:id="@+id/shift_btn"
      android:layout_width="85dp"
      android:layout_height="85dp"
      android:layout_marginBottom="10dp"
      android:layout_marginTop="5dp"
      android:background="@drawable/shift_off" />
  </LinearLayout>

  <com.samsung.penboard.DrawingSpaceView
    android:id="@+id/drawing_space"
    android:layout_width="wrap_content"
    android:layout_height="200dp"
```

```
    android:layout_centerInParent="true"
    android:layout_toLeftOf="@+id/right_panel"
    android:layout_toRightOf="@+id/left_panel"
    android:background="@drawable/background"
    android:fadeDuration="300"
    android:fadeEnabled="true"
    android:fadeOffset="200"
    android:gestureColor="#000"
    android:gestureStrokeAngleThreshold="0.0"
    android:gestureStrokeLengthThreshold="0.0"
    android:gestureStrokeSquarenessThreshold="0.0"
    android:gestureStrokeType="multiple" >
  </com.samsung.penboard.DrawingSpaceView>

  <LinearLayout
    android:id="@+id/right_panel"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentRight="true"
    android:orientation="vertical" >

    <Button
      android:id="@+id/backspace_btn"
      android:layout_width="85dp"
      android:layout_height="85dp"
      android:layout_marginBottom="5dp"
      android:layout_marginTop="10dp"
      android:background="@drawable/backspace" />

    <Button
      android:id="@+id/space_btn"
      android:layout_width="85dp"
      android:layout_height="85dp"
      android:layout_marginBottom="10dp"
      android:layout_marginTop="5dp"
      android:background="@drawable/space" />
  </LinearLayout>

</com.samsung.penboard.PenboardView>
```

[Code 4] Layout of the input area view

As you may have noticed, the layout contains three columns. There are two side panels: the left one contains symbols and shift buttons and the right one contains the backspace and space buttons.

Between them is the drawing space, where gestures are performed. PenboardView is actually a RelativeLayout with a few additional functionalities to manage its contents.

```
public class PenboardView extends RelativeLayout {
  private final Context mContext;
  private DrawingSpaceView mDrawingSpaceView;
  private Button mSymbolsButton;
  private Button mShiftButton;
  private Button mBackspaceButton;
  private Button mSpaceButton;

  public PenboardView(Context context, AttributeSet attrs) {
    super(context, attrs);
    mContext = context;
  }

  @Override
  protected void onFinishInflate() {
    mDrawingSpaceView = (DrawingSpaceView) findViewById(R.id.drawing_space);
    mDrawingSpaceView.setOnGestureRecognizedListener(new OnGestureRecognizedListener() {
      @Override
      public void gestureRecognized(String character) {
        enterCharacter(character);
      }
    });
    mSymbolsButton = (Button) findViewById(R.id.symbols_btn);
    mSymbolsButton.setOnClickListener(mButtonClickListener);
    mSymbolsButton.setOnLongClickListener(mButtonLongClickListener);
    mShiftButton = (Button) findViewById(R.id.shift_btn);
    mShiftButton.setOnClickListener(mButtonClickListener);
    mShiftButton.setOnLongClickListener(mButtonLongClickListener);
    mBackspaceButton = (Button) findViewById(R.id.backspace_btn);
    mBackspaceButton.setOnClickListener(mButtonClickListener);
    mBackspaceButton.setOnLongClickListener(mButtonLongClickListener);
    mSpaceButton = (Button) findViewById(R.id.space_btn);
    mSpaceButton.setOnClickListener(mButtonClickListener);
    mSpaceButton.setOnLongClickListener(mButtonLongClickListener);
  }
}
```

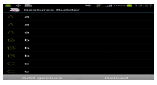
[Code 5] Input area class skeleton with its views initialization

[Image 1] Input method look

Defining gestures

Since the created app is based on a set of gestures included in the Graffiti alphabet, you'll need to define them in an Android way. Android provides a gestures API, but to handle them, you'll need to create a gesture library file. A little help comes from the GestureBuilder app, which is included in the SDK samples. You can simply define gestures and save them to a file using this app. Create at least three gestures for every character (for the recognition engine to be more precise), and name a gesture the same as the character it represents. Where you are finished, put the created file into res/raw/gestures in your project.

[Image 2] Defining a gesture



[Image 3] List of all defined gestures

For more information regarding working with gestures in Android, check out the following article on the Samsung Developers
website : http://developer.samsung.com/android/technical-docs/Gestures-in-Android

Gesture overlay view

As you saw above, the input area layout contains the DrawingSpaceView class with a few attributes specific for the GestureOverlayView class, which is the Android component responsible for handling gestures. Those attributes are related to background image, colors, fade effect and some thresholds when a gesture is recognized. The only thing left to do is load the created gesture file, handle gesture recognition and pass a character to the upper class.

```
public class DrawingSpaceView extends GestureOverlayView implements
OnGesturePerformedListener {
  private final static double SCORE_TRESHOLD = 3.0;
  private final GestureLibrary mGestureLibrary;
  private OnGestureRecognizedListener mOnGestureRecognizedListener;

  public DrawingSpaceView(Context context, AttributeSet attrs) {
    super(context, attrs);
    mGestureLibrary = GestureLibraries.fromRawResource(context, R.raw.gestures);
    mGestureLibrary.load();
    addOnGesturePerformedListener(this);
  }

  public void setOnGestureRecognizedListener(OnGestureRecognizedListener
```

```
onGestureRecognizedListener) {
    mOnGestureRecognizedListener = onGestureRecognizedListener;
  }

  @Override
  public void onGesturePerformed(GestureOverlayView overlay, Gesture gesture) {
    List<Prediction> predictions = mGestureLibrary.recognize(gesture);
    Prediction bestPrediction = null;
    if (predictions.size() > 0) {
      bestPrediction = predictions.get(0);
    }
    if (mOnGestureRecognizedListener != null && bestPrediction != null) {
      if (bestPrediction.score > SCORE_TRESHOLD) {
        mOnGestureRecognizedListener.gestureRecognized(bestPrediction.name);
      } else {
        clear(false);
      }
    }
  }
}
```

[Code 6] Gesture overlay view class (DrawingSpaceView.java)

The gesture library based on the created file is initialized during object creation. When a gesture is recognized, the OnGesturePerformed listener is invoked. If the prediction score of the gesture is higher than the given threshold, the recognized character is passed to the PenboardView class through the OnGestureRecognizedListener. Otherwise, the gesture is ignored.

Passing letters

The moment a character is recognized, is the moment when input connection should be notified. But first, there is a need to check if shift button was pressed, because it may affect the entered letter. All of the logic below is done in the PenboardView class. This class mediates between the input method service and the gesture overlay view.

```
private ShiftState mShiftState = ShiftState.OFF;

private enum ShiftState {
  OFF, ON, CAPS_LOCK
};

private void shift() {
  switch (mShiftState) {
  case OFF:
    switchShiftTo(ShiftState.ON);
    break;
```

```
  case ON:
    switchShiftTo(ShiftState.CAPS_LOCK);
    break;
  case CAPS_LOCK:
    switchShiftTo(ShiftState.OFF);
    break;
  default:
    break;
  }
}

private void switchShiftTo(ShiftState state) {
  switch (state) {
  case OFF:
    mShiftState = ShiftState.OFF;
    mShiftButton.setBackgroundResource(R.drawable.shift_off);
    break;
  case ON:
    mShiftState = ShiftState.ON;
    mShiftButton.setBackgroundResource(R.drawable.shift_on);
    break;
  case CAPS_LOCK:
    mShiftState = ShiftState.CAPS_LOCK;
    mShiftButton.setBackgroundResource(R.drawable.shift_caps_lock);
    break;
  default:
    break;
  }
}

private void enterCharacter(String character) {
  switch (mShiftState) {
  case OFF:
    mOnCharacterEnteredListener.characterEntered(character);
    break;
  case ON:
    mOnCharacterEnteredListener.characterEntered(character.toUpperCase(Locale.ENGLISH));
    switchShiftTo(ShiftState.OFF);
    break;
  case CAPS_LOCK:
    mOnCharacterEnteredListener.characterEntered(character.toUpperCase(Locale.ENGLISH));
    break;
  default:
```

```
    break;
  }
}
```

[Code 7] Character entering with shift handling (PenboardView.java)

Invoking OnCharacterEnteredListener causes communication with input connection in the input service.

```
penboardView.setOnCharacterEnteredListener(new OnCharacterEnteredListener() {
  @Override
  public void characterEntered(String character) {
    getCurrentInputConnection().commitText(character, 1);
  }
});
```

[Code 8] Committing a character (PenboardView.java)

After committing, the character is displayed on the screen.

[Image 4] Actual writing

Using space and backspace

Providing space and backspace functionalities is quite simple. Inserting a space is nothing more than inserting a character, which is represented by one space.

```
private final OnClickListener mButtonClickListener = new OnClickListener() {
  @Override
  public void onClick(View v) {
    switch (v.getId()) {
    […]
    case R.id.space_btn:
      mOnCharacterEnteredListener.characterEntered(" ");
      break;
    […]
    }
  }
};
```

[Code 9] Entering a space (PenboardView.java)

You may want to implement deleting a single character and a whole word, so you need to distinguish the two backspace modes, which can be represented by short and long presses of the backspace button .

```
private final OnClickListener mButtonClickListener = new OnClickListener() {
@Override
  public void onClick(View v) {
    switch (v.getId()) {
    […]
    case R.id.backspace_btn:
      mOnBackspacePressedListener.backspacePressed(false);
    break;
    […]
    }
  }
};

private final OnLongClickListener mButtonLongClickListener = new OnLongClickListener() {
  @Override
  public boolean onLongClick(View v) {
    switch (v.getId()) {
    […]
      case R.id.backspace_btn:
      mOnBackspacePressedListener.backspacePressed(true);
      return true;
    […]
    }
    return false;
  }
};
```

[Code 10] Behavior after short and long-press of the backspace button (PenboardView.java)

The boolean value in the listener interface informs if the click was short or long. Now the input connection can handle backspace functionality.

```
penboardView.setOnBackspacePressedListener(new OnBackspacePressedListener() {
  @Override
  public void backspacePressed(boolean isLongClick) {
    if (isLongClick) {
      deleteLastWord();
    } else {
      getCurrentInputConnection().deleteSurroundingText(1, 0);
    }
  }
});

private void deleteLastWord() {
```

```
  final int charactersToGet = 20;
  final String splitRegexp = " ";

  // delete trailing spaces
  while (getCurrentInputConnection().getTextBeforeCursor(1,
0).toString().equals(splitRegexp)) {
    getCurrentInputConnection().deleteSurroundingText(1, 0);
  }

  // delete last word letters
  String[] words = getCurrentInputConnection().getTextBeforeCursor(charactersToGet,
0).toString().split(splitRegexp);
  getCurrentInputConnection().deleteSurroundingText(words[words.length - 1].length(), 0);
}
```

[Code 11] Backspace functionality managed by the input service (PenboardView.java)

Passing symbols

Since the user may want to enter not only the letters, you should provide a set of additional symbols to use in your input method. One of the methods to achieve this is creating another activity, which will handle symbol selection and pass it to the service. It is a little more complicated, because the service will not handle input when it is obscured by another window. Therefore, you may create a symbols queue, which will be empty and all of its contents will be entered when the service is ready.

```
public class SymbolsActivity extends Activity {
  public static final String INTENT_ACTION = "com.samsung.penboard.SYMBOL_ENTERED";
  public static final String INTENT_EXTRA_NAME = "symbol";
  private static final String[] SYMBOLS = new String[] { "1", "2", "3", "4", "5", "6", "7",
"8", "9", "0", "!", "@",
      "#", "$", "%", "^", "&", "*", "(", ")", "`", "-", "=", "~", "_", "+", "[", "]", "\\",
"{", "}", "|", ";",
      "'", ":", "\"", ",", ".", "/", "<", ">", "?" };
  private GridView mGridView;

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    requestWindowFeature(Window.FEATURE_NO_TITLE);
    setContentView(R.layout.symbols);
    mGridView = (GridView) findViewById(R.id.symbols_gridview);
    ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
android.R.layout.simple_list_item_1, SYMBOLS);
    mGridView.setAdapter(adapter);
    mGridView.setOnItemClickListener(new OnItemClickListener() {
```

```
      @Override
      public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        Intent intent = new Intent(INTENT_ACTION);
        intent.putExtra(INTENT_EXTRA_NAME, SYMBOLS[position]);
        LocalBroadcastManager.getInstance(SymbolsActivity.this).sendBroadcast(intent);
      }
    });
  }
}
```

[Code 12] Symbols activity (SymbolsActivity.java)

As you can see, it contains a defined array of symbols. They are shown in a grid view, and after choosing one of them, a broadcast message is sent using the LocalBroadcastManager. You can set this activity style to dialog and define some grid view attributes in the layout file.

```
<activity
  android:name=".SymbolsActivity"
  android:configChanges="orientation|keyboardHidden"
  android:screenOrientation="portrait"
  android:theme="@android:style/Theme.Dialog" >
</activity>
```

[Code 13] Manifest declaration of activity (AndroidManifest.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="400dp"
    android:background="@android:color/transparent"
    android:orientation="horizontal" >

    <GridView
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/symbols_gridview"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:columnWidth="40dp"
        android:gravity="center"
        android:numColumns="auto_fit"
        android:stretchMode="columnWidth" />

</LinearLayout>
```

[Code 14] Activity layout (symbols.xml)

What is left to do is to receive those broadcast in the input area view and pass the symbols to the

input service.

```
private final Queue<String> mSymbolsQueue = new LinkedList<String>();

public PenboardView(Context context, AttributeSet attrs) {
  […]
  LocalBroadcastManager.getInstance(mContext).registerReceiver(mBroadcastReceiver,  new
IntentFilter(SymbolsActivity.INTENT_ACTION));
}

private final BroadcastReceiver mBroadcastReceiver = new BroadcastReceiver() {
  @Override
  public void onReceive(Context context, Intent intent) {
    if (SymbolsActivity.INTENT_ACTION.equals(intent.getAction())) {
      mSymbolsQueue.add(intent.getStringExtra(SymbolsActivity.INTENT_EXTRA_NAME));
    }
  }
};
```

[Code 15] Receiving symbols message and adding them to the queue (PenboardView.java)

```
public class Penboard extends InputMethodService {
  private PenboardView mPenboardView;
  […]

  @Override
  public void onStartInput(EditorInfo attribute, boolean restarting) {
    if (mPenboardView != null) {
      Queue<String> symbolsQueue = mPenboardView.getSymbolsQueue();
      while (!symbolsQueue.isEmpty()) {
        String character = symbolsQueue.poll();
        getCurrentInputConnection().commitText(character, 1);
      }
    }
  }
  […]
}
```

[Code 16] Handling symbols input (Penboard.java)

As you can see, the OnStartInputListener is used to check if there are any symbols in the queue, to be sure that the input service is ready and they can be entered.

[Image 5] Symbols activity

Summary

After going through the above steps, you can implement a simple gesture-writing input method. If you would like to use more of input service features, please review further documentation on the Android Developers website:
http://developer.android.com/guide/topics/text/creating-input-method.html

ref:http://developer.samsung.com/android/technical-docs/Implementing-a-custom-input-method