Departamento de Informática | Universidad de Valladolid

Tecnología de Programación

Widgets y otros elementos folclóricos

Félix Prieto

Curso 2011/12

App Widgets

- Se trata de vistas que pueden ser integradas en la pantalla principal
- Reciben periódicamente mensajes de actualización
- Pueden estar asociados a una aplicación
- Pueden tener una actividad especializada en configurarlos
- Pero pueden desarrollarse de modo totalmente independiente

Para construir un widget

- `AppWidgetProviderInfo` Información sobre el widget, habitualmente en XML
- `AppWidgetProvider` Clase mediante la cual recibimos los eventos asociados con el widget y los procesamos
- Un layout que describe el widget (descrito en XML)
- Opcionalmente, una actividad que facilita su configuración

AppWidgetProviderInfo

- En `AndroidManifest.xml`
  - Un bloque `receiver`
    - Información sobre el `AppWidgetProvider`
    - Un `intent-filter` para aceptar actualizaciones
    - Información sobre el resto de los recursos en XML
- En `/res/xml`
  - Un bloque `appwidget-provider`
    - `android:minWidth`: Ancho mínimo
    - `android:minHeight`: Alto mínimo
    - `android:updatePeriodMillis`: Intervalo entre actualizaciones, no puede ser inferior a 30 minutos
    - `initialLayout`
- El layout se define como cualquier otro...
- Sería razonable respetar las recomendaciones de Google:
  `developer.android.com/guide/practices/ui_guidelines/widget_design.html`

AppWidgetProvider

- Construimos un descenciente de `AppWidgetProvider`
- Redefinimos los métodos relacionados con su ciclo de vida:
  - `onEnabled` Se crea la primera instancia del Widget
  - `onDeleted` Se borra alguna instancia del Widget
  - `onDisabled` Se borra la última instancia del Widget
  - `onUpdate` Se actualiza el widget
  - `onReceive` Se recibe «cualquier» mensaje, incluidos los anteriores

Implementación de `onUpdate`

```
@Override
public void onUpdate(Context context,
    AppWidgetManager appWM,int[] appWIds) {
  Log.d(TAG, TAG + ":␣Llega␣actualización");
  RemoteViews vista = new RemoteViews(context.
      getPackageName(),R.layout.main);
  vista.setTextViewText(R.id.text, "Otro␣Valor");
  appWM.updateAppWidget(appWIds, vista);
  super.onUpdate(context, appWM, appWIds);
}
```


Curiosidades de `onUpdate`

- Posiblemente estamos actualizando varios Widgets
- Tenemos que acceder a la vista mediante `RemoteViews` porque no pertenece a «nuestra aplicación»
- Necesitamos `AppWidgetManager` cambiar la vista en todos los Widgets activos

Comentarios sobre las actualizaciones

- Mediante este mecanismo el intervalo mínimo de actualización es de 30 minutos
- Incluso Google recomienda actualizaciones cada 60 minutos para conservar la batería del dispositivo
- Podemos aumentar la frecuencia de actualización utilizando un objeto `AlarmManager`
  - Activamos un reloj en el método `onEnabled`
  - Desactivamos el reloj en el método `onDisabled`
  - El sistema activará un intent a intervalos regulares mientras el reloj esté activo

El método *onEnabled*

```
@Override
public void onEnabled(Context context) {
  // Instalada la primera instancia del widget
  Log.d(TAG, TAG + ": Ejecutado onEnabled()");
  Intent intent =
    new Intent(ProveedorWidget.ACTUALIZACION_DEL_WIDGET);
  if (pendingIntent == null)
    pendingIntent = PendingIntent.getBroadcast(context, 0,
      intent, 0);
  if (alarmManager == null)
    alarmManager = (AlarmManager) context
        .getSystemService(Context.ALARM_SERVICE);
  Calendar calendar = Calendar.getInstance();
  calendar.setTimeInMillis(System.currentTimeMillis());
  calendar.add(Calendar.SECOND, 45);
  alarmManager.setRepeating(AlarmManager.RTC,
      calendar.getTimeInMillis(),45 * 1000, pendingIntent);
  super.onEnabled(context);
}
```


El método *onDisabled*

```
@Override
public void onDisabled(Context context) {
  // Eliminando la última instancia del widget
  Log.d(TAG, TAG + ": Ejecutado onDisabled()");
  alarmManager.cancel(pendingIntent);
  super.onDisabled(context);
}
```


El sistema de telefonía

- Acceso a la API pública mediante `TelephonyManager`
- Obtenemos una instancia de la clase mediante `getService(context.TELEPHONY_SERVICE)`
- Generar llamadas mediante un intent con dos argumentos
  - `Intent.CALL_ACTION` o `Intent.DIAL_ACTION` en función de si deseas ver el marcador
  - `Uri.parse("tel:"+elNumero)`
- `PhoneNumberUtils` proporciona utilidades para gestionar números de teléfono
- También se puede acceder a los datos de la agenda del dispositivo
- Todo esto requiere permisos especiales en la instalación

Permisos necesarios

- `android.permission.READ_PHONE_STATE`
- `android.permission.MODIFY_PHONE_STATE`
- `android.permission.CALL_PHONE`
- `android.permission.CALL_PRIVILEGED`
- `android.permission.PROCESS_OUTGOING_CALLS`

La idea básica

- El sistema de telefonía no es diferente del resto de los sistemas Android
- Si el usuario nos da permiso, podemos utilizar el sistema de telefonía
- Podemos «escuchar» los eventos de telefonía y reaccionar a ellos
- Quizás el usuario no espera que nuestro programa lo haga,...
  - ¿Cuántos permisos requiere el programa que se tira pedos?
- Existe un sistema en Android Market para denunciar estos casos

Los pasos necesarios

- Elaborar una aplicación que haga otra cosa.
- Añadir un `BroadcastReceiver` (por ejemplo desde XML para los eventos `PHONE_STATE` y `NEW_OUTGOING_CALL`
- Declarar los permisos adicionales en `AndroidManifest.xml`
- Escribir el código que tratará las llamadas.

Las llamadas salientes

- El «broadcast» que se emite es «ordenado»
- Podemos dar una prioridad alta a nuestro receptor
- Podemos interrumpir el «broadcast» o alterar sus datos
- Luego podemos:
  - Impedir llamadas salientes
  - Modificar llamadas salientes
  - Monitorizar o realizar acciones no relacionadas con la llamada

Las llamadas entrantes

- Detectamos un cambio de estado en el teléfono
- Hay que seleccionar sólo las llamadas entrantes
- El «broadcast» que se emite no es «ordenado», no podemos cancelarlo
- Pero podemos «colgar» el teléfono aunque esta acción no esté disponible en la API oficial
  - Al menos, de momento, y en algunos dispositivos

Para usar la cámara necesitamos...

- Añadir un requisito de hardware a la aplicación
  `<uses-feature android:name="android.hardware.camera"/>`
- Añadir permiso para usar la cámara
  `<uses-permission android:name="android.permission.CAMERA"/>`
- Elegir un tipo de implementación
  - Utilizar una aplicación de cámara instalada
  - Realizar nuestra propia aplicación
- Las imágenes o videos suelen guardarse en la tarjeta SD
  - Necesitamos permisos adicionales si vamos a escribir sobre la tarjeta SD
- El programa de cámara suele utilizar también permisos de acceso al audio y al sistema de localización.

Elementos básicos

- `Camera` Permite acceder a la cámara directamente, sin utilización del programa de cámara instalado
- `SurfaceView` Permite presentar la imagen en vivo de la cámara al usuario
- `MediaStore.ACTION_IMAGE_CAPTURE` o `MediaStore.ACTION_VIDEO_CAPTURE` permiten mediante un intent capturar imágenes y vídeos respectivamente
- Las imágenes se guardan en el fichero indicado por nuestra actividad
  - Almacenar fuera de la SD puede provocar problemas con los permisos
- Nuestra actividad puede «procesar» la imágen tras recuperarla del fichero

Interpretar códigos de barras

- Podemos tomar una foto, localizar en ella un código de barras e interpretarlo
- Existen bibliotecas que realizan todo el trabajo por nosotros
  - `http://code.google.com/p/zxing/`
- Usamos un par de clases en nuestra aplicación:
  - `IntentIntegrator` e `IntentResult`
- Lanzamos el intent mediante
  ```
  IntentIntegrator i = new IntentIntegrator(this);
  i.initiateScan()
  ```
- Recuperamos el resultado mediante en `onActivityResult` usando un `IntentResult`
- La biblioteca gestiona la instalación de una aplicación si es necesario

Cámara vs. Emulador

- No todas las versiones del emulador tienen implementada la cámara
  - El emulador de «froyo» muestra un patrón de imágen para hacer las pruebas
  - El emulador de «ics» puede localizar la webcam del ordenador
- No todas las cámaras emuladas son compatibles con zxing
  - Sólo he conseguido que funcione en dispositivos físicos
- No todo emulador dispone de «Android Marquet»
  - En ese caso hay que instalar la aplicación `BarcodeScanner` de forma manual
    - Descargar de la web del desarrollador
    - Instalar mediante
      `adb install Descargas/BarcodeScanner3.72.apk`