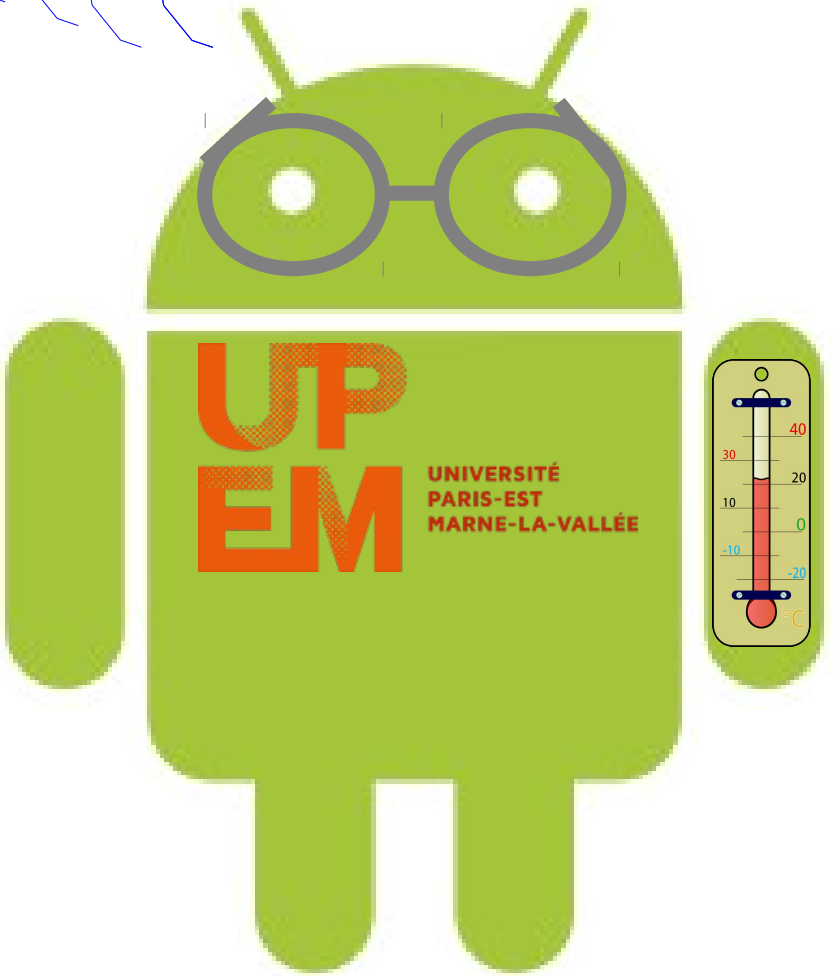
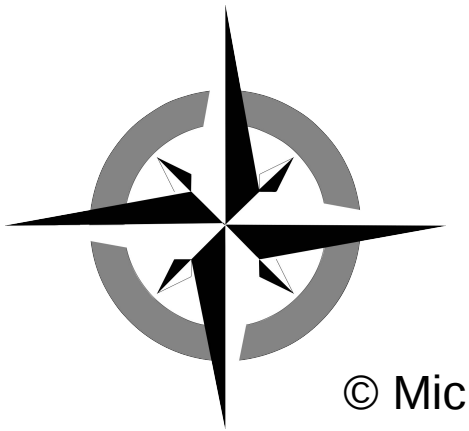
Utilisation des capteurs sous Android



Master 2 informatique
2013-2014



© Michel Chilowicz <chilowi at u-pem.fr> - Sous licence CC By-NC-SA

Vibreur

- Permission requise : *android.permission.VIBRATE*
- *Vibrator v = (Vibrator)getSystemService(Context.VIBRATOR_SERVICE)*
- Méthodes utiles :
  - *boolean hasVibrator()* : y-a-t-il un vibreur disponible ?
  - *vibrate(int durationInMillis)* : vibre pendant la durée indiquée
  - *vibrate(int[] array, int index)* : vibre selon la séquence indiquée, array[0] indique un temps de non-vibration, array[1] un temps de vibration, ... et index spécifie un indice pour commencer la répétition
  - *cancel()* : annule une vibration demandée

© chilowi at u-pem.fr (CC By-NC-SA)

Fournisseurs de localisation

- *LocationManager lm = (LocationManager)getSystemService(Context.LOCATION_SERVICE)*
- Permissions nécessaires :
  - *android.permission.ACCESS_COARSE_LOCATION* : pour une localisation basée sur les bornes cellulaires et Wi-Fi
  - *android.permission.ACCESS_FINE_LOCATION* : pour une localisation utilisant le récepteur GPS
- Localisation apportée par un *LocationProvider* plus ou moins précis, consommateur d'énergie, coûteux
  - Providers par défaut : *GPS_PROVIDER, NETWORK_PROVIDER, PASSIVE_PROVIDER* (provider "lazy")
  - Tous les providers : *List<String> getAllProviders()*
  - Meilleur provider selon une liste de critères : *String getBestProvider(Criteria criteria, boolean enabled)*
  - Un provider identifié par son nom : *LocationProvider getProvider(String name)*

© chilowi at u-pem.fr (CC By-NC-SA)

Obtenir des localisations

- Dernière localisation connue : *LocationManager.getLastKnownLocation(String provider)* (peut retourner null si le provider est indisponible)
  - Retourne un objet *Location* avec méthodes *getLatitude()*, *getLongitude()* (en degrés), *getAltitude()* (en mètres), *getTime()* (en ms depuis l'epoch)
- *LocationManager.requestLocationUpdates(String provider, long minTimeMillis, float minDistance, LocationListener listener)* pour obtenir des MAJ de localisation
  - *minTimeMillis* et *minDistance* définissent des périodes temporelles et spatiales de MAJ
  - *LocationListener* doit implanter une méthode *onLocationChanged(Location)* appelée à chaque localisation fournie
  - Désenregistrement du listener avec *removeUpdates(LocationListener)*
- *void addProximityAlert(double lat, double lon, float radius, long expirationInMillis, PendingIntent intent)* pour ajouter une alerte de proximité
  - L'intent est envoyé dès que l'on passe dans la zone spécifiée ; l'alerte expire après *expirationInMillis* (pas d'expiration si == -1)
  - Suppression de l'alerte avec *removeProximityAlert(PendingIntent intent)*

© chilowi at u-pem.fr (CC By-NC-SA)

Un service loggant les localisations

```
public class LocationLogger extends Service
{
	public static final String LOCATION_FILE = "locations.log";

	LocationManager locationManager = null; LocationListener locationListener = null; Writer writer = null;

	@Override public IBinder onBind(Intent arg0) { return null; }

	private void log(String message)
	{
		try {
			if (writer == null)
				writer = new OutputStreamWriter(openFileOutput(LOCATION_FILE, MODE_APPEND));
			writer.write(new Date() + ": " + message); writer.flush();
		} catch (IOException e) {
			Log.e(getClass().getName(), "Cannot log message " + message + " due to an exception", e);
		}
	}

	@Override public void onCreate()
	{
		locationManager = (LocationManager)getSystemService(LOCATION_SERVICE);
		locationListener = new LocationListener() {

			@Override public void onStatusChanged(String provider, int status, Bundle extras)
			{
				log(String.format("Change of status of provider %s: %d", provider, status));
			}

			@Override public void onProviderEnabled(String provider)
			{
				log(String.format("Provider %s is enabled", provider));
			}

			@Override public void onProviderDisabled(String provider)
			{
				log(String.format("Provider %s is disabled", provider));
			}

			@Override public void onLocationChanged(Location location)
			{
				log(String.format("latitude=%f, longitude=%f, altitude=%f",
					location.getLatitude(), location.getLongitude(), location.getAltitude()));
			}
		};
	}

	@Override public int onStartCommand(Intent intent, int flags, int startId)
	{
		locationManager.requestLocationUpdates(intent.getStringExtra("provider"),
			intent.getLongExtra("minTime", 10000), intent.getFloatExtra("minDistance", 100.0f),
			locationListener);
		return Service.START_REDELIVER_INTENT; // Restart the service with the intent if it is destroyed
	}

	@Override public void onDestroy()
	{
		try { if (writer != null) writer.close(); } catch (IOException e) {}
		locationManager.removeUpdates(locationListener);
	}

}
```

© chilowi at u-pem.fr (CC By-NC-SA)

Senseurs

- Obtention de *SensorManager* : *Context.getSystemService(Context.SENSOR_SERVICE)*
- Liste des senseurs : *SensorManager.getSensorList(int typeOfSensor)*
- Types de senseurs actuellement supportés :
  - Accéléromètre, gyroscope
  - Thermomètre, hygromètre, baromètre
  - Magnétomètre
  - Senseur de proximité
  - Luxmètre
- Certains senseurs sont accessibles de plusieurs manières (données brutes ou données analysées) :
  - Accéléromètre : Sensor.TYPE_ACCELEROMETER, Sensor.TYPE_GRAVITY, Sensor.TYPE_LINEAR_ACCELERATION

© chilowi at u-pem.fr (CC By-NC-SA)

Comment utiliser des senseurs ?

(1) On créé un *SensorEventListener* avec deux méthodes à implanter :

- *void onAccuracyChanged(Sensor s, int accuracy)* : est appelée lorsque la précision change (SENSOR_STATUS_ACCURACY_{UNRELIABLE, LOW, MEDIUM, HIGH})
- void onSensorChanged(SensorEvent event) : event contient les champs accuracy, sensor, timestamp et values (tableau de float décrivant les données)

(2) On enregistre ce listener (typiquement dans *Activity.onResume()*) : *sensorManager.registerListener(listener, sensor, delay) ;*

(3) Lorsque l'on ne souhaite plus recevoir des événements des senseurs (typiquement dans Activity.onPause()), on enlève le listener : *sensorManager.unregisterListener(listener, sensor) ;*

© chilowi at u-pem.fr (CC By-NC-SA)

Liste des valeurs de senseurs

```
public class SensorDisplayer extends Activity
{
        private SensorManager sensorManager;
        private List<Sensor> sensors;
        private List<float[]> latestSensorEvents = new ArrayList<float[]>();
        private SensorEventListener sensorListener;
        private ListView sensorListView;
        private ArrayAdapter<float[]> arrayAdapter;

        @Override
        protected void onCreate(Bundle savedInstanceState)
        {
                super.onCreate(savedInstanceState);
                sensorManager = (SensorManager)getSystemService(SENSOR_SERVICE);
                sensors = sensorManager.getSensorList(Sensor.TYPE_ALL);
                for (int i = 0; i < sensors.size(); i++) latestSensorEvents.add(null);
                // We create the listener for the sensor events
                this.sensorListener = new SensorEventListener() {
                        @Override public void onSensorChanged(SensorEvent event) { updateSensorValue(event); }
                        @Override public void onAccuracyChanged(Sensor sensor, int accuracy) {}
                };
                // We create a list view displaying all the sensor data
                // Data for a sensor is put as a float array into an array list
                sensorListView = new ListView(this);
                arrayAdapter = new ArrayAdapter<float[]>(this, R.layout.simpletextview, latestSensorEvents)
                                {
                                        @Override public View getView(int position, View convertView, ViewGroup parent)
                                        {
                                                TextView tv = (TextView)convertView;
                                                if (tv == null) tv = new TextView(SensorDisplayer.this);
                                                float[] values = getItem(position);
                                                String message = sensors.get(position).getName() + ":\n"
                                                        + ((values == null)?"no data available":Arrays.toString(values));
                                                tv.setText(message);
                                                return tv;
                                        }
                                };
                sensorListView.setAdapter(arrayAdapter); setContentView(sensorListView);
        }

        public void updateSensorValue(SensorEvent sensorEvent)
        {
                latestSensorEvents.set(sensors.indexOf(sensorEvent.sensor), Arrays.copyOf(sensorEvent.values, sensorEvent.values.length));
                arrayAdapter.notifyDataSetChanged();
        }

        @Override public void onResume()
        {
                super.onResume();
                // We register the listener for all the sensors
                for (Sensor s: sensors) sensorManager.registerListener(sensorListener, s, SensorManager.SENSOR_DELAY_UI);
        }

        @Override public void onPause()
        {
                super.onPause();
                // We unregister the listener
                for (Sensor s: sensors) sensorManager.unregisterListener(sensorListener, s);
        }

}
```

© chilowi at u-pem.fr (CC By-NC-SA)

MediaPlayer

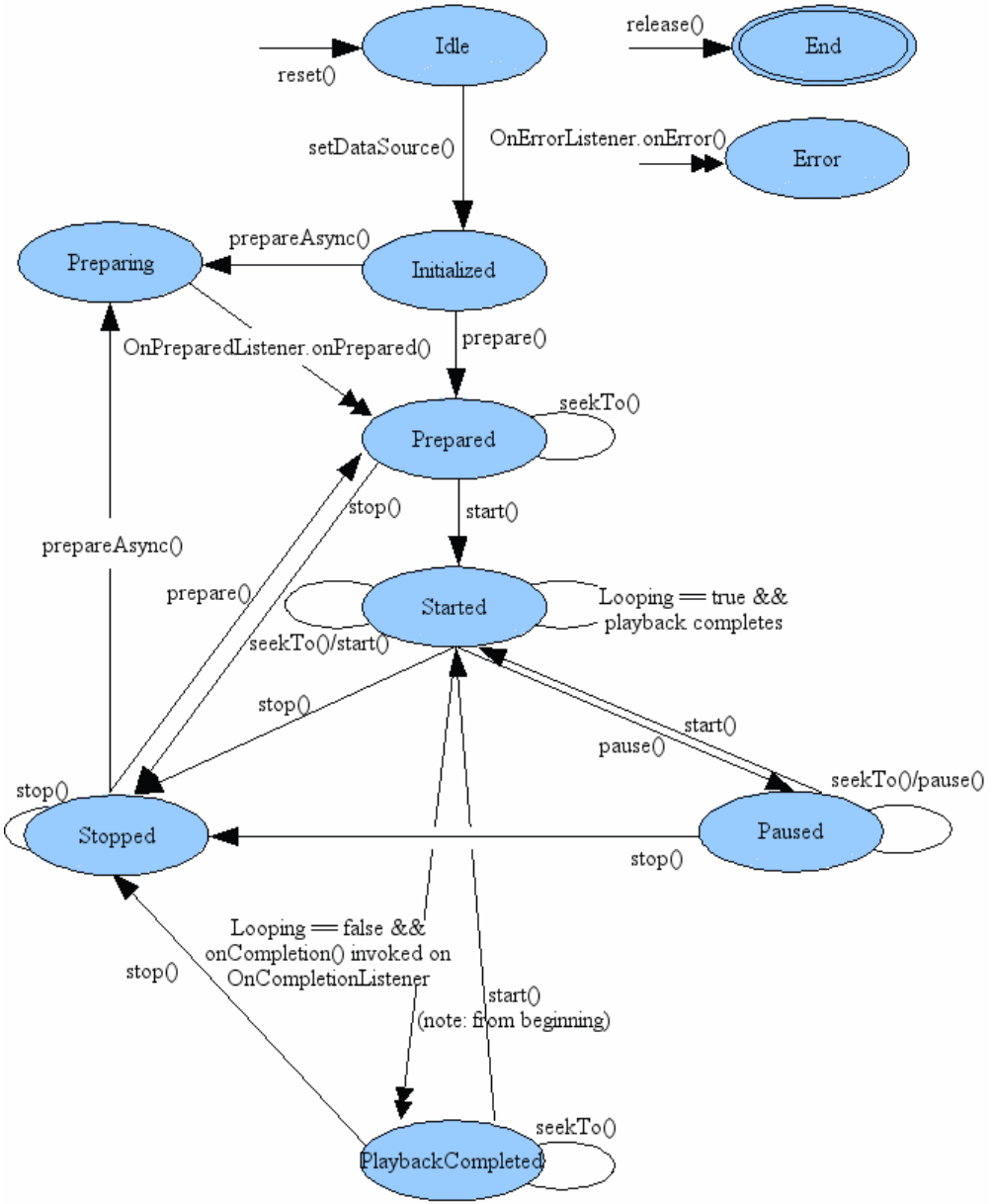
- MediaPlayer offre une API pour lire les formats usuels (vidéo : h264, audio : AAC, FLAC, MP3, midi, Vorbis...) en local ou avec les protocoles réseau RTP et HTTP
- L'utilisation de verrous peut être utile lors de la lecture : wake lock, wifi lock...
- Initialisation de MediaPlayer :

```
MediaPlayer mp = new MediaPlayer() ;
mp.setAudioStreamType(AudioManager.STREAM_MUSIC) ;
// Other types : ALARM, DTMF, NOTIFICATION, RING, SYSTEM, VOICE_CALL
mp.setDataSource(myURL) ;
mp.setOnPreparedListener(new OnPreparedListener()
    {
        public void onPrepared(MediaPlayer mp) { ... }
    }) ;
mp.setWakeMode(getApplicationContext(), PowerManager.PARTIAL_WAKE_LOCK);
mp.prepareAsync() ;
```

© chilowi at u-pem.fr (CC By-NC-SA)

Ne pas oublier de libérer les ressources avec release()

Diagramme d'état de MediaPlayer



Source : http://developer.android.com/reference/android/media/MediaPlayer.html

AudioManager

- Coopération pour la gestion du focus audio
  - Focus possédé par un seul composant
  - Focus transférable à un autre composant :
    - *FocusManager.requestAudioFocus(OnAudioFocusChange listener, int streamType, int durationHint)*
    - *OnAudioFocusChangeListener. onAudioFocusChange(int focusChanger)*
      - AUDIOFOCUS_GAIN
      - AUDIOFOCUS_LOSS
      - AUDIOFOCUS_LOSS_TRANSIENT
      - AUDIO_FOCUS_LOSS_TRANSIENT_CAN_DUCK
- Information du passage sur HP : broadcast de *android.media.AUDIO_BECOMING_NOISY*

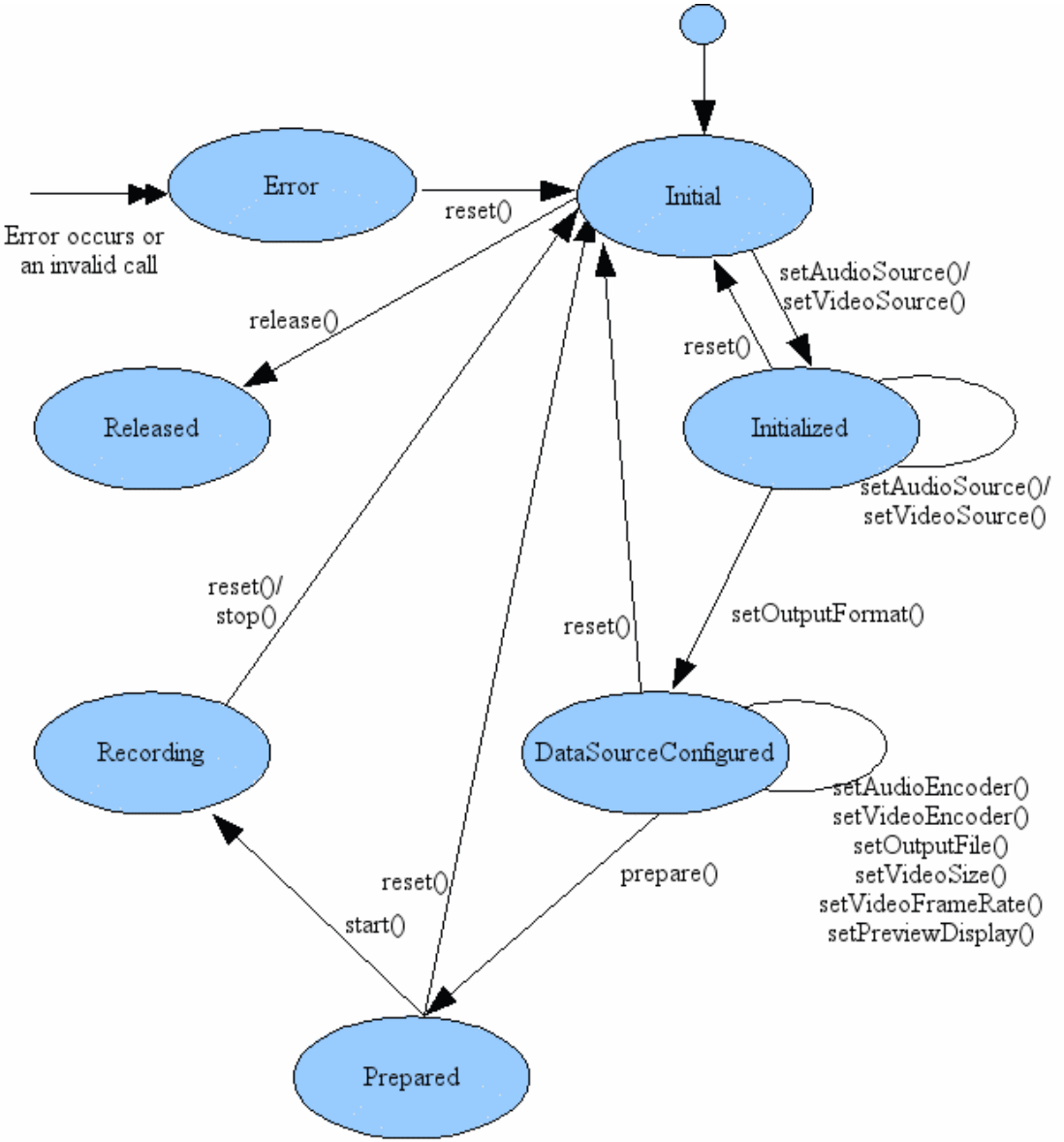
© chilowi at u-pem.fr (CC By-NC-SA)

MediaRecorder

- Requiert *android.permission.{RECORD_AUDIO, RECORD_VIDEO}*
- Pour enregistrer du son (et facultativement de la vidéo) :
  - `MediaRecorder mr = new MediaRecorder() ;`
  - `mr.setAudioSource(MediaRecorder.AudioSource.{DEFAULT, MIC, VOICE_CALL, VOICE_COMMUNICATION, VOICE_RECOGNITION, VOICE_DOWNLINK, VOICE_UPLINK})`
  - `mr.setVideoSource(MediaRecorder.VideoSource.{DEFAULT, CAMERA})`
  - `mr.setOutputFormat(MediaRecorder.OutputFormat.{DEFAULT, THREE_GP, MPEG_4, AMR_NB, AMR_WB, AAC_ADTS, ...})`
  - `mr.setAudioEncoder(MediaRecoarder.AudioEncoder.{DEFAULT, AAC, AMR_NB, AMR_WB, ...})`
  - `mr.setVideoEncoder(MediaRecorder.VideoEncoder.{DEFAULT, H263, H264})`
  - `mr.setOutputFile(path)`
  - `mr.prepare()`
  - `mr.start() ;`
  - `...`
  - `mr.stop() :`
  - `mr.release() ;`

© chilowi at u-pem.fr (CC By-NC-SA)

États de MediaRecorder



MediaRecorder state diagram

Source : http://developer.android.com/reference/android/media/MediaRecorder.html

Appel de l'application caméra

```
private final static int IMAGE_CAPTURE_REQUEST_CODE = 1;
private final static int VIDEO_CAPTURE_REQUEST_CODE = 2;

     public void onCameraStartClick(boolean video)
     {
          Intent intent = new Intent((video)?MediaStore.ACTION_VIDEO_CAPTURE:MediaStore.ACTION_IMAGE_CAPTURE);
          // We can also use MediaStore.ACTION_IMAGE_CAPTURE_SECURE to capture an image in lock mode
          // To capture a video, we use MediaStore.ACTION_VIDEO_CAPTURE
          File pictDir = new File(Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_PICTURES),
          this.getClass().getSimpleName());
          String timestamp = new SimpleDateFormat("yyyyMMdd_HHmmss").format(new Date());
          File dest = new File(pictDir, timestamp); // Destination file
          intent.putExtra(MediaStore.EXTRA_OUTPUT, Uri.fromFile(dest)); // Specify the destination file
          // Now, we start the picture capture activity
          startActivityForResult(intent, (video)?VIDEO_CAPTURE_REQUEST_CODE:IMAGE_CAPTURE_REQUEST_CODE);
     }

     @Override
     public void onActivityResult(int requestCode, int resultCode, Intent data)
     {
          switch (requestCode)
          {
          case IMAGE_CAPTURE_REQUEST_CODE:
               switch (resultCode)
               {
               case RESULT_OK:
                    Toast.makeText(this, "Image saved to " + data.getData(), Toast.LENGTH_LONG).show();
                    ((ImageView)findViewById(R.id.imageCaptureView)).setImageURI(data.getData());
                    break;
               case RESULT_CANCELED: default:
                    Toast.makeText(this, "Capture of image failed", Toast.LENGTH_SHORT).show();
               }
               break;
          case VIDEO_CAPTURE_REQUEST_CODE:
               ...
          }
     }
```

© chilowi at u-pem.fr (CC By-NC-SA)

Usage de l'API caméra

- Dans le manifeste :

```
<uses-permission android:name="android.permission.CAMERA" />
 <uses-feature android:name="android.hardware.camera" />
 <uses-feature android:name="android.hardware.camera.autofocus" />
```

- Combien de caméras ? Informations
  - *Camera.getNumberOfCameras()*
  - *Camera.getCameraInfo(int cameraid, CameraInfo i)*
- Ouvrir une caméra : *Camera.open(int cameraid)*
  - Récupération et fixation des paramètres : *getParameters()*, *setParameters(Camera.Parameters)*
  - Ne pas oublier d'appeler *release()* pour la libérer

© chilowi at u-pem.fr (CC By-NC-SA)

Prévisualisation de caméra

- Où envoyer les données de prévisualisation ?
  - Sur une texture OpenGL : *setPreviewTexture(SurfaceTexture st)*
  - Sur un *SurfaceHolder* (typiquement obtenu avec *SurfaceView.getHolder()*) : *setPreviewDisplay(SurfaceHolder h)*
  - Sur une méthode callback : *setPreviewCallback(Camera.PreviewCallback cb)*
    - *setPreviewFormat(int)* définit le format binaire des previews (par défaut NV21)
    - *PreviewCallback.onPreviewFrame(byte[] data, Camera c)* doit être implanté
- Contrôle de la prévisualisation
  - *startPreview()* pour démarrer
  - *stopPreview()* pour arrêter

© chilowi at u-pem.fr (CC By-NC-SA)

Zoom et capture

- Contrôle du zoom
  - *getMaxZoom()* : zoom maximal (grand angle=0)
  - *setZoom(int value)* : fixation du zoom
  - *startSmoothZoom(int value)*, *stopSmoothZoom()* : zoom progressif (possibilité d'utiliser un listener)
- Capture
  - *takePicture(Camera.ShutterCallback shutter, Camera.PictureCallback raw, Camera.PictureCallback postview, Camera.PictureCallback jpeg)*
    - *shutter.onShutter()* sert à signaler la prise de photo (on peut jouer par exemple un son)
    - Chaque *PictureCallback* est optionnel (peut être null) selon les données souhaitées : brutes, post-traitées ou compressées en JPEG. La méthode *onPictureTaken(byte[] data, Camera camera)* doit être implantée.

© chilowi at u-pem.fr (CC By-NC-SA)