Cas EDF : Développement Android - Concepts avancés – Partie 3

Cette publication comporte cinq parties dont l'ordre est dicté par la logique du développement.
Les parties 2 et 3 sont facultatives.

Partie 1 : Gestion des clients
Partie 2 : Géolocalisation de l'agent et géocodage du client sélectionné
- **Partie 3 : Signature Client**

Partie 4 : Communication avec le serveur
Partie 5 : Identification, import et export des données.

Description du thème

Propriétés	Description
Intitulé long	**Cas EDF : Développement Androïd - Concepts avancés - Partie 3 : Signature Client**
Formation concernée	BTS Services Informatiques aux Organisations
Matière	SLAM 4
Présentation	Développement permettant d'aborder des concepts de la programmation Android d'une application embarquée, communiquant avec un serveur. Il aborde les notions : ➢ d'affichage de liste / d'adapter, ➢ de GEOLOCALISATION / GEOCODER, ➢ de graphisme (canvas) et d'encodage JPG, ➢ d'échange avec un serveur WEB (THREAD / JSON / GSON), ➢ d'utilisation d'un SGBDO DB4o.
Notions	Savoirs • D4.1 - Conception et réalisation d'une solution applicative • D4.2 - Maintenance d'une solution applicative Savoir-faire • Programmer un composant logiciel • Exploiter une bibliothèque de composants • Adapter un composant logiciel • Valider et documenter un composant logiciel • Programmer au sein d'un framework
Transversalité	SLAM5
Pré-requis	Développement d'une application Android sous un environnement Eclipse. (Exemple : Cas AMAP Jean-Philippe PUJOL)
Outils	Eclipse, DB4o, OME, Gson, Google play services, Apache, Mysql
Mots-clés	Application mobile, Android, SGBDO, DB4o, Géolocalisation, Géocodage, Thread, json, Gson, MVC, canvas, encodage JPG
Durée	24 heures (8,4,4,4,4) (Temps divisé par 2 si utilisation du squelette application)
Auteur	Pierre François ROMEUF avec la relecture et les judicieux conseils de l'équipe CERTA
Version	v 1.0
Date de publication	Juin 2014

Contexte

Application embarquée sur une solution technique d'accès (STA) sous Android, permettant à un agent EDF d'effectuer sa tournée journalière de relevés des compteurs EDF.

Les principales fonctionnalités sont :

- Identification de l'agent sur le device avec contrôle sur un serveur web,
- Import des clients depuis un serveur web,
- Affichage des clients,
- Saisie des informations clients,
- Aide au déplacement via géolocalisation de la position de l'agent EDF et géocodage de l'adresse client,
- Enregistrement de la signature client validant les informations saisies,
- Export des données sur le serveur web.

Le SGBD embarqué est un SGBDO DB4o. Le serveur distant est un serveur de type LAMP installé sur la ferme de serveurs ou via un hébergement gratuit (ex : http://www.hostinger.fr/)

Cette application peut être dérivée pour de multiples besoins : ceux des livreurs, des commerciaux, des visiteurs, des contrôleurs ...

Le code fourni en annexe nécessite de votre part une compréhension.
Il représente normalement votre travail de programmeur, de fouille sur internet, avec tests, compréhension et modifications du code.

Il vous est fourni afin que ce développement ne représente qu'un travail raisonnable et pour vous présenter les différentes facettes du développement sur Android.

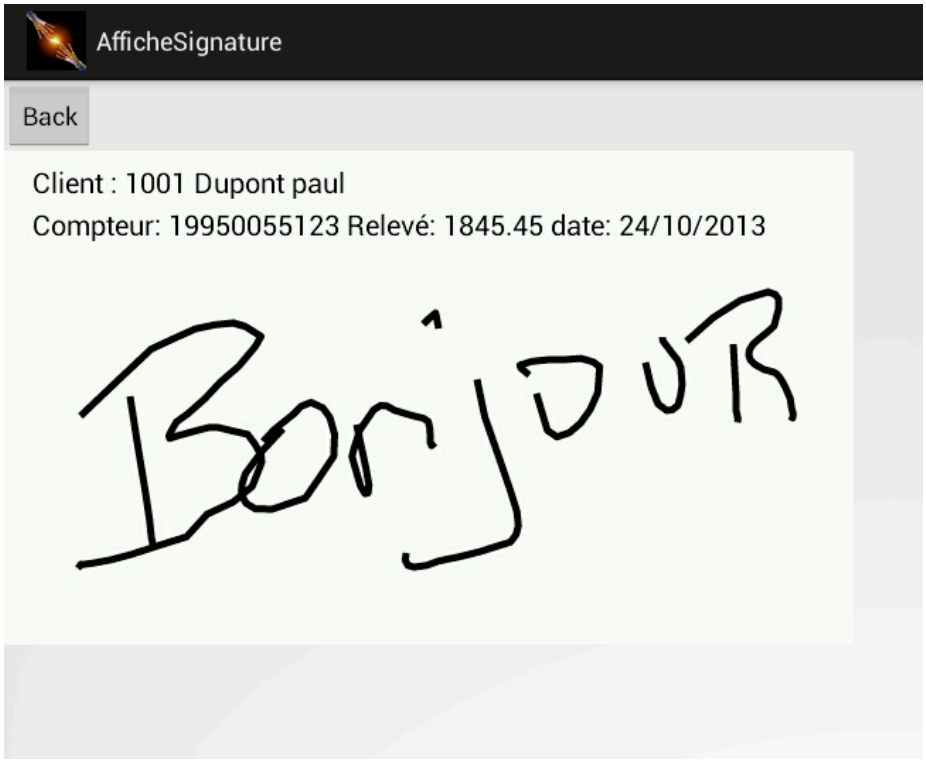
Conseils : consultez notamment developer.android.com**,** le tutoriel du zéro (http://uploads.siteduzero.com/pdf/554364-creez-des-applications-pour-android.pdf), etc.

Signature du client

Activity CaptureSignature & AfficheSignature

Exemple d'écran CaptureSignature :

Exemple d'écran AfficheSignature :



- Créez deux nouvelles Activity *CaptureSignature* et *AfficheSignature*.

- Associez l'activity *CaptureSignature* au clic du bouton 'Signature' de l'Activity *ModificationClient*

    - Appel de la méthode de vérification des données saisies dans l'activity *ModificationClient*
    - Si le résultat de la vérification est valide, sauvegarde des données du client
    - Si le résultat de la vérification est valide, appel de l'Activity *CaptureSignature* en lui passant l'identifiant du client.

- Associez l'activity *AfficheSignature* au clic du bouton 'Vue Signature' de l'Activity *ModificationClient*

    - Vérifiez que le client possède une signature enregistrée à partir de la longueur de signature_Base64 qui doit être supérieure à 0.
    - Si le résultat de la vérification est valide, appel de l'Activity *AfficheSignature* en lui passant l'identifiant du client.

Vue d'ensemble de l'API Canvas

L'API *Canvas* permet de créer des effets graphiques complexes.

Vous dessinez sur une surface *bitmap* (image matricielle définie par un tableau de point). Le bitmap contient les données réelles de pixels de l'image, alors que la classe *Canvas* fournit une interface de haut niveau pour dessiner des formes, les lignes, le texte, etc. La classe *Bitmap* fournit une interface de plus bas niveau, c'est-à-dire de l'image, vous permettant de manipuler directement les pixels.

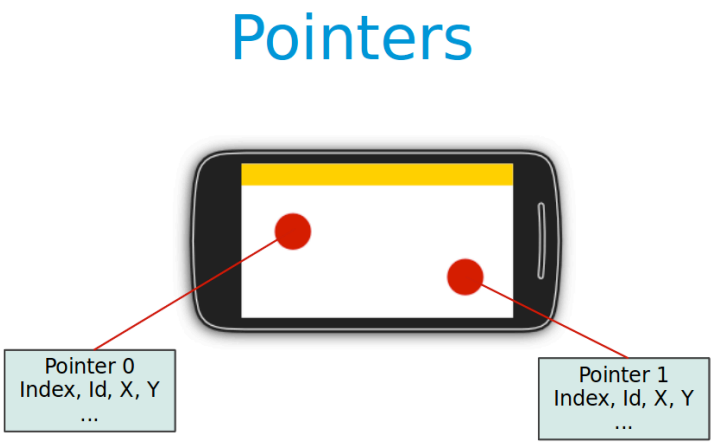
La classe *Canvas* fournit les méthodes de dessin pour dessiner sur un bitmap et la classe *Paint* spécifie comment vous dessinez sur le *bitmap*.

Un objet de type *Canvas* contient le *bitmap* sur lequel vous dessinez. Il fournit également des méthodes pour les opérations de dessin, par exemple, *drawARGB()* pour dessiner en couleur, *drawBitmap( )* pour dessiner une image bitmap, *drawText()* pour dessiner un texte, *drawRoundRect( )* pour dessiner un rectangle avec des coins arrondis et bien plus encore (redimensionner, faire pivoter, transformer l'image, sauvegarde et restauration des états de dessin). Imaginez-le comme un morceau de papier sur lequel pouvez dessiner.

Pour dessiner sur l'objet *Canvas* vous utilisez un objet de type *Paint*.
La classe *Paint* permet de préciser les effets de couleur, la police et certains effets pour l'opération de dessin. Imaginez-le comme le pinceau physique que vous tenez dans votre main lors de la peinture sur une toile. Vous pouvez avoir plusieurs brosses différentes qui ont toutes des propriétés différentes, et que vous utilisez chacun dans un but différent.

La classe *Path* permet de décrire un conteneur virtuel d'actions graphiques à l'intérieur de votre *Canvas*.

La classe centrale pour réaliser une interface graphique sous Android est la classe "*View*".
La classe *View* réagit aux événements tactiles et nous permettra de récupérer les points du dessin.



Nous allons donc réagir aux *onTouchEvent()* et récupérer la position du doigt via *get(x)* et *get(y)*.
Les événements :

MotionEvent.ACTION_DOWN	Le doigt touche l'écran (démarre le dessin)
MotionEvent.ACTION_MOVE	Le doigt bouge
MotionEvent.ACTION_UP	Le doigt ne touche plus l'écran
MotionEvent.ACTION_CANCEL	L'événement est supprimé, une autre application a pris le contrôle.

Si l'application ne nécessite pas une quantité importante de transformations ou de vitesse de défilement d'écran (*frame-rate*) spécifique, il est plus pratique de créer une *View* personnalisée qui va directement nous proposer un *Canvas* pour dessiner avec l'appel *de View.onDraw( )* .

De plus, nous utiliserons les méthodes permettant de sauvegarder un *bitmap* en chaine de caractères, le *bitmap* étant transformé ici selon le protocole JPEG en un tableau de bits, encodé par la suite dans une chaine de caractères en base 64 qui nous permettra de sauvegarder la signature et de la réafficher.

Activity CaptureSignature

Layout

Nous utiliserons un *Layout* réduit pour diminuer la taille de la sauvegarde du JPEG, exemple :

Un *LinearLayout* largeur 600 hauteur 500 contenant

- un *LinearLayout* pour les boutons (largeur `match_parent)` contenant 3 boutons (height 50dp width `match_parent)`
- un *LinearLayout* (*height* et *width* `match_parent)` que l'on remplira avec la *View* permettant de signer à la main.

Code

- Déclarez une variable linearLayout de type *LinearLayout* et l'associer au dernier *LinearLayout* de notre *Layout*.
- Déclarez une variable *client* de type Client et la charger via appel des méthodes de Modele.
- Dans le *onCreate*
  - Récupérez via l'objet `Bundle` l'identifiant du client
  - Générez 2 chaines de caractères, lig1 et lig2, telles qu'elles sont décrites sur la copie d'écran ("Client :...","Compteur :....")
  - Déclarer les 3 *Buttons*, les affecter aux boutons du *LinearLayout*.
  - Rendre le bouton *save* inaccessible
  - Générer les `setOnClickListener` des 3 boutons

La classe *View* représente le bloc de construction de base pour les composants de l'interface utilisateur et donc nécessaire pour gérer une zone de dessin. Une vue occupe une zone rectangulaire à l'écran et propose les méthodes pour le dessin et la gestion des événements.

Nous allons donc ici, à l'intérieur de notre *Activity,* déclarer une classe héritant de *View* surchargeant les méthodes nous permettant de réagir aux événements du doigt et nous permettant de dessiner (cf. code annexe **`class`** `Signature`).

- Déclarez une variable *signature* de type Signature
- Modifiez le constructeur de Signature
  - Ajoutez 2 paramètres chaines de caractères permettant de passer lig1,lig2 et de les initialiser
  - Affectez la couleur du *background.* Exemple this.setBackgroundColor(Color.WHITE);
  - Définissez les propriétés de notre pinceau *paint.* Exemple :

```
paint.setAntiAlias(true); // empêche le scintillement gourmand en cpu et
mémoire
paint.setColor(Color.BLACK);
paint.setStrokeWidth(5f); //taille de la grosseur du trait en pixel
paint.setTextSize(20);// taille du texte pur afficher les lignes
```

- Ajoutez à la classe Signature une méthode permettant de sauver le dessin (cf. annexe)
- Ajoutez à la classe Signature une méthode permettant de réinitialiser le dessin

```
path.reset();invalidate();
```

- Modifiez la méthode onDraw de Signature en ajoutant le code permettant de dessiner les lignes et l'ensemble des points du *path*

```
paint.setStyle(Paint.Style.FILL);
canvas.drawText(lig1, 20, 30, paint);
canvas.drawText(lig2, 20, 60, paint);
paint.setStyle(Paint.Style.STROKE);
canvas.drawPath(path, paint);
```

- Modifiez la méthode *onTouchEvent* de Signature en ajoutant le fait que le bouton *save* devient accessible

- Dans le *onCreate* de CaptureSignature

  - Instanciez signature(variable de type Signature) en appelant le constructeur de Signature signature = new Signature(this, null,lig1,lig2);

  - Ajouter à linearLayout (le linearlayout de notre xml) signature

```
linearLayout.addView(signature, LayoutParams.MATCH_PARENT,
            LayoutParams.MATCH_PARENT);
```

  - Sur le clic du bouton *clear*, appelez la méthode de Signature permettant de réinitialiser le dessin et rendez le bouton *save* inaccessible.

  - Sur le clic du bouton *save,* appelez la méthode de Signature permettant de sauvegarder le dessin en *String*, puis modifiez la variable signature_Base64 de l'objet client, sauvegardez l'objet client et appelez finish()

  - Sur le clic de *cancel* appelez finish().

Activity AfficheSignature

Layout

Idem que CaptureSignature sauf pour les boutons.

Code

La grande différence consiste dans le code de la classe Signature.
En effet, cette dernière ne sert qu'à afficher la signature sauvegardée dans la variable signature_Base64 du client dont l'identifiant est passé en paramètre.

La méthode *onTouchEvent* de dessinsignature ne sert donc à rien.
Seules les variables `bitmap et canvas` sont utiles.

Le constructeur de Signature ne fait rien et n'a pas besoin des chaines de caractères.
La méthode *onDraw* de Signature ne sert qu'a dessiner le *Canvas* via

```
canvas.drawBitmap(bitmap, 0, 0, null);
```

Dans *l'Activity AfficheSignature* il suffit donc d'appeler, à partir d'une variable de type Signature, la méthode dessine (cf. annexe) de Signature qui, à partir de la chaine de caractères signature_Base64 de l'objet client, va régénérer l'image via un bitmap qui sera redessiné.

ANNEXE : Exemple de code

Gestion graphique de la signature

Classe héritant de View

```
public class Signature extends View {
	// variables nécessaire au dessin

	private Paint paint = new Paint();
	private Path path = new Path();();// collection de l'ensemble des points
sauvegardés lors des mouvements du doigt
	private Canvas canvas;
	private Bitmap bitmap;

	public Signature(Context context, AttributeSet attrs) {
		super(context, attrs);
	}

	//gestion du dessin
	@Override
	protected void onDraw(Canvas canvas) {
		super.onDraw(canvas);
	}
	// gestion des événements du doigt
	@Override
	public boolean onTouchEvent(MotionEvent event) {
		float eventX = event.getX();
		float eventY = event.getY();
		switch (event.getAction()) {
		case MotionEvent.ACTION_DOWN:
			path.moveTo(eventX, eventY);
			return true;
		case MotionEvent.ACTION_MOVE:
			path.lineTo(eventX, eventY);
			break;
		case MotionEvent.ACTION_UP:
			// nothing to do
			break;
		default:
			return false;
		}
		invalidate();
		return true;
		}
	}
```

Sauvegarde d'un *Canvas* en un *String*, image encodée en jpeg base64

```
	public String save() {
		String vretour;
		if (bitmap == null) {
			bitmap = Bitmap.createBitmap(this.getWidth(), this.getHeight(),
					Bitmap.Config.RGB_565);
		}
		try {
			canvas = new Canvas(bitmap);
			this.draw(canvas);
			ByteArrayOutputStream ByteStream = new ByteArrayOutputStream();
			bitmap.compress(Bitmap.CompressFormat.JPEG, 100, ByteStream);
			byte[] b = ByteStream.toByteArray();
			vretour = Base64.encodeToString(b, Base64.DEFAULT);

		} catch (Exception e) {
```

```
                Toast.makeText(getApplicationContext(), "Erreur Sauvegarde",
                                Toast.LENGTH_LONG).show();
                vretour = null;
        }
                return vretour;
    }
```

Transformation d'un *String* en image jpeg et dessin dans un *Canvas*

```
    public void dessine(String encodedString) {
        try {
                byte[] encodeByte = Base64
                                .decode(encodedString, Base64.DEFAULT);
                bitmap = BitmapFactory.decodeByteArray(encodeByte, 0,
                            encodeByte.length);
                bitmap = bitmap.copy(bitmap.getConfig(), true);
        } catch (Exception e) {
                Toast.makeText(getApplicationContext(), "error dessine",
                                Toast.LENGTH_LONG).show();
        }
        canvas = new Canvas(bitmap);
        this.draw(canvas);
    }
```