Android : les éléments avancés

Kotlin

* Création et utilisation d'un content provider
* Présentation

Un content provider (fournisseur de contenu) permet de partager des données d'une application entre différentes applications. L'intérêt d'utiliser un content provider pour partager des données est multiple :

* Sécurité : permet aux applications d'accéder à des données en lecture et écriture via des fonctions spécifiques. Ces fonctions peuvent empêcher des actions dangereuses lors de modifications et peuvent contrôler les informations à fournir.
* Productivité : toutes les fonctions de partage de données ont un fonctionnement commun et s'utilisent de la même manière, permettant une meilleure gestion du code.

Généralement, un content provider manipule les données d'une base de données.

* Structure

Un content provider est une classe héritant de la classe ContentProvider. Cette classe contient au moins les fonctions suivantes :

|  |  |
| --- | --- |
| Fonction | Description |
| getType() | Cette fonction permet de fournir un type MIME pour décrire les données traitées par le content provider. |
| onCreate() | Cette fonction s'exécute au moment de la création du content provider. |
| query() | Cette fonction permet de renvoyer à l'utilisateur du content provider des données dans un objet de type Cursor. |
| insert() | Cette fonction permet d'insérer des données via le content provider. |
| update() | Cette fonction permet de mettre à jour des données via le content provider. |
| delete() | Cette fonction permet de supprimer des données via le content provider. |

* Identification

Pour utiliser un content provider, il est nécessaire de connaître son identifiant. Cet identifiant est nommé URI.

L'URI contient généralement le nom de la classe contenant le content provider :

Exemple d'URI

fr.acos.contentproviderwithkotlin.MonContentProvider

Si un content provider offre la possibilité d'exploiter différents types de données, son URI peut ressembler à ceci :

Exemple d'URI vers des données de type Personne

fr.acos.contentproviderwithkotlin.MonContentProvider/Personne

Cette URI indique que le content provider fr.acos.contentproviderwithkotlin.MonContentProvider sera utilisé pour manipuler des données de type Personne.

Exemple d'URI vers des données de type Animal

fr.acos.contentproviderwithkotlin.MonContentProvider/Animal

Cette URI indique que le content provider fr.acos.contentproviderwithkotlin.MonContentProvider sera utilisé pour manipuler des données de type Animal.

* Créer un content provider

Pour créer un content provider, il suffit de faire un clic droit sur un package. Puis, choisir : New > Other > Content provider, puis définir un nom de classe.

* Création : définir les fonctionnalités d'un content provider

Lors de la création d'un content provider, il est nécessaire de redéfinir les fonctions query(), insert(), update(), delete().

* Lire des données via un content provider

La fonction query() permet de faire des recherches sur les données manipulées par le content provider.

Syntaxe de la fonction query

query(<uri>, <projection>, <selection>, <selectionArgs>,   
<sortOrder>):Cursor

|  |  |
| --- | --- |
| Paramètre | Description |
| uri | L'URI du content provider. Il faut utiliser le format content://URI. |
| projection | La liste de toutes les colonnes à retourner. Null si toutes les colonnes sont retournées. |
| selection | Permet de définir un filtre sur les colonnes passées en paramètres. Null si aucun filtre. |
| selectionArgs | Permet de définir les valeurs de restriction pour le filtre défini. |
| sortOrder | Permet de définir les colonnes sur lesquelles le résultat sera trié. |
| Valeur de retour | Un curseur contenant les résultats de la recherche. |

Exemple d'utilisation de la fonction query

/\*\*   
 \* Fonction permettant de récupérer des personnes.   
 \*   
 \* @param uri uri du content provider   
 \* @param projection inutile   
 \* @param selection inutile   
 \* @param selectionArgs inutile   
 \* @param sortOrder inutile   
 \*/   
override fun query(uri: Uri, projection: Array<String>?, selection: String?,   
                  selectionArgs: Array<String>?, sortOrder: String?): Cursor?   
{   
    Log.i("XXX","Fonction query. uri = $uri")   
    Log.i("XXX","Fonction query. Matcher.match(uri) = ${MATCHER.match(uri)}")   
    var cursor:Cursor? = null   
    //Création d'un objet représentant la base de données.   
    val db = MaBaseDeDonnees.getInstance(context)   
    //Création d'un objet représentant la DAO pour les personnes.   
    val dao = db.personneDao()   
   
    when (MATCHER.match(uri))   
    {   
        //Si l'URI est égal à content://fr.acos.MonContentProvider/personnes/\*   
        CODE\_PERSONNE\_ITEM ->   
        {   
            //Récupération de l'identifiant à la fin de l'URI   
            val id = uri.lastPathSegment   
            //Retourne un objet de type Cursor contenant une personne.   
            cursor = dao.getInCursor(id.toLong())   
        }   
        //Si l'URI est égal à content://fr.acos.MonContentProvider/personnes   
        CODE\_PERSONNE\_DIR ->   
        {   
            //Retourne un objet de type Cursor contenant plusieurs personnes.   
            cursor = dao.getInCursor()   
        }   
        else -> throw IllegalArgumentException("Informations erronées")   
    }   
    return cursor;   
}

Dans cet exemple, la fonction query utilise l'ORM Room (cf. chapitre Android : quelques librairies incontournables - Room) pour faire ses recherches dans la base de données. Les paramètres projection, selection, selectionArgs et sortOrder ne sont pas utilisés.

* Insérer des données via un content provider

La fonction query() permet de faire des insertions sur les données manipulées par le content provider.

Syntaxe de la fonction insert

insert(<uri>, <values>)

|  |  |
| --- | --- |
| Paramètre | Description |
| uri | L'URI du content provider. Il faut utiliser le format content://. |
| values | Objet de type ContentValues, la clé représente la colonne et la valeur contient la valeur à insérer. |

Exemple d'utilisation de la fonction insert

/\*\*   
 \* Fonction permettant d'ajouter une personne   
 \*   
 \* @param uri URI du content provider   
 \* @param values informations relatives à la personne à inserer.   
 \*/   
override fun insert(uri: Uri, values: ContentValues?): Uri?   
{   
    Log.i("XXX","Fonction insert. uri = $uri")   
    when (MATCHER.match(uri))   
    {   
        //Si l'URI est égal à content://fr.acos.MonContentProvider/personnes   
        CODE\_PERSONNE\_DIR ->   
        {   
            //Création d'un objet représentant la base de données.   
            val db = MaBaseDeDonnees.getInstance(context)   
            //Création d'un objet représentant la DAO pour les personnes.   
            val dao = db.personneDao()   
            //Transformation du ContentValue en objet de type Personne.   
            val personneAInserer = itemBuilder(values)   
            //Insertion en base. id est égal à l'identifiant du   
            //nouvel enregistrement.   
            val id = dao.insert(personneAInserer)   
            //Retourne l'URI avec le nouvel identifiant à la fin.   
            return ContentUris.withAppendedId(uri,id)   
        }   
        //Si l'URI est égal à content://fr.acos.MonContentProvider/personnes/\*   
        CODE\_PERSONNE\_ITEM -> throw IllegalArgumentException("Pas   
d'ID nécessaire pour l'insertion")   
        else -> throw IllegalArgumentException("Informations   
erronées")   
    }   
}

Dans cet exemple, la fonction query() utilise l'ORM Room (cf. chapitre Android : quelques librairies incontournables - Room) pour faire ses insertions dans la base de données.

* Mettre à jour des données via un content provider

La fonction query() permet de faire des mises à jour sur les données manipulées par le content provider.

Syntaxe de la fonction update

update(<uri>, <values>)

|  |  |
| --- | --- |
| Paramètre | Description |
| uri | L'URI du content provider. Il faut utiliser le format content://. |
| values | Objet de type ContentValues, la clé représente la colonne et la valeur contient la valeur à insérer. |

Exemple d'utilisation de la fonction update

/\*\*   
 \* Fonction permettant de mettre à jour des personnes   
 \*   
 \* @param uri URI du content provider   
 \* @param values  informations relatives à la personne à modifier.   
 \* @param selection inutile   
 \* @param selectionArgs inutile   
 \*/   
override fun update(uri: Uri, values: ContentValues?, selection: String?,   
                    selectionArgs: Array<String>?): Int {   
    Log.i("XXX","Fonction update. uri = $uri")   
    Log.i("XXX","Fonction update. Matcher.match(uri) = ${MATCHER.match(uri)}")   
   
    when (MATCHER.match(uri))   
    {   
        //Si l'URI est égal à content://fr.acos.MonContentProvider/personnes/\*   
        CODE\_PERSONNE\_ITEM ->   
        {   
            //Création d'un objet représentant la base de données.   
            val db = MaBaseDeDonnees.getInstance(context)   
            //Création d'un objet représentant la DAO pour les personnes.   
            val dao = db.personneDao()   
            //Transformation du ContentValue en objet de type Personne.   
            val personneAMettreAJour = itemBuilder(values);   
            //Mise à jour en base de données.   
            val nombreDeMiseAjour = dao.updatePersonne(personneAMettreAJour)   
            //Retourne le nombre d'éléments mis à jour.   
            return nombreDeMiseAjour   
        }   
        //Si l'URI est égal à content://fr.acos.MonContentProvider/personnes   
        CODE\_PERSONNE\_DIR -> throw IllegalArgumentException("ID   
nécessaire pour l'insertion")   
        else -> throw IllegalArgumentException("Informations erronées")   
    }   
}

Dans cet exemple, la fonction update() utilise l'ORM Room (cf. chapitre Android : quelques librairies incontournables - Room) pour faire ses mises à jour dans la base de données.

* Supprimer des données via un content provider

La fonction query() permet de faire des suppressions sur les données manipulées par le content provider.

Syntaxe de la fonction delete

myCR.delete(<uri>, <selection>, <selectionArgs>)

|  |  |
| --- | --- |
| Paramètre | Description |
| uri | L'URI du content provider. Il faut utiliser le format content://. |
| selection | Permet de définir un filtre sur les colonnes passées en paramètres. Null si aucun filtre. |
| selectionArgs | Permet de définir les valeurs de restriction pour le filtre défini. |

Exemple d'utilisation de la fonction delete

/\*\*   
 \* Fonction permettant de supprimer une personne.   
 \*   
 \* @param uri URI du content provider   
 \* @param selection inutile   
 \* @param selectionArgs inutile   
 \*/   
override fun delete(uri: Uri, selection: String?, selectionArgs:   
Array<String>?): Int   
{   
    Log.i("XXX","Fonction delete. uri = $uri")   
   
    when (MATCHER.match(uri))   
    {   
        //Si l'URI est égal à content://fr.acos.MonContentProvider/personnes/\*   
        CODE\_PERSONNE\_ITEM ->   
        {   
            //Récupération de l'identifiant à la fin de l'URI   
            val id = uri.lastPathSegment   
            //Création d'un objet représentant la base de données.   
            val db =  MaBaseDeDonnees.getInstance(context)   
            //Création d'un objet représentant la DAO pour les personnes.   
            val dao = db.personneDao()   
            //Création d'un objet de type Personne avec seulement un identifiant.   
            val personneASupprimer = Personne(id.toLong())   
            //Suppression de la base de données.   
            val nombreDeSuppression = dao.deletePersonne(personneASupprimer)   
            //retourne le nombre de lignes supprimées.   
            return nombreDeSuppression   
        }   
        //Si l'URI est égal à content://fr.acos.MonContentProvider/personnes   
        CODE\_PERSONNE\_DIR ->throw IllegalArgumentException("ID   
nécessaire pour la suppression")   
        else -> throw IllegalArgumentException("Informations erronées")   
    }   
}

Dans cet exemple, la fonction delete() utilise l'ORM Room (cf. chapitre Android : quelques librairies incontournables - Room) pour faire ses suppressions dans la base de données.

* Création : définir les permissions d'un content provider
* Présentation

Si un ContentProvider ne spécifie aucune autorisation, aucune application ne pourra l'utiliser. En revanche, l'application contenant le ContentProvider peut l'utiliser en lecture et en écriture, quelles que soient les autorisations spécifiées.

Un ContentProvider peut avoir une utilisation limitée à l'application le contenant ou être utilisé par d'autres applications. Pour limiter l'utilisation à l'application le contenant, il faut définir l'attribut du ContentProvider android: exported à false dans le manifeste de l'application. Sinon, définissez l'attribut android: exported à true pour permettre aux autres applications d'accéder aux ContentProvider.

Si un ContentProvider est utilisé entre plusieurs applications d'un même développeur, il faut définir l'attribut du ContentProvider android: protectionLevel à signature dans le manifeste de l'application. Ainsi, aucune permission ne sera nécessaire.

Pour spécifier une autorisation pour utiliser le ContentProvider dans le but de modifier des données, l'attribut android:writePermission doit être spécifié avec une valeur correspondant à l'autorisation nécessaire.

android:writePermission="fr.acos.contentpriverwithkotlin.WRITE"

Pour spécifier une autorisation pour utiliser le ContentProvider dans le but de lire des données, l'attribut android:readPermission doit être spécifié avec une valeur correspondant à l'autorisation nécessaire.

android:readPermission="fr.acos.contentpriverwithkotlin.READ"

Le droit d'écriture donne implicitement le droit de lecture.

* Usage

<provider   
    android:name=".provider.MonContentProvider"   
    android:authorities="fr.acos.MonContentProvider"   
    android:enabled="true"   
    android:exported="true"   
    android:readPermission="fr.acos.contentpriverwithkotlin.READ"   
    android:writePermission="fr.acos.contentpriverwithkotlin.WRITE"   
    ></provider>

* Utiliser un content provider

Points nécessaires pour utiliser un content provider :

* Définir les permissions requises par le content provider.
* Connaître l'URI du content provider.
* Récupérer un objet de type ContentResolver fourni par le Context.
* Utiliser l'objet de type ContentResolver avec l'URI.
* Connaître l'URI du content provider

Pour utiliser un content provider, il est nécessaire de connaître son URI. Android fournit de nombreux content providers de base, leur URI est disponible sur le site internet : https://developer.android.com

Voici une liste de content providers fournis par Android :

|  |  |
| --- | --- |
| Content Provider | Description |
| AlarmClock provider | Permet de manipuler les alarmes du téléphone. |
| Blockednumber provider | Permet de récupérer les numéros bloqués. |
| Calendar provider | Permet de gérer les événements du calendrier. |
| CallLog provider | Permet de récupérer les appels récents. |
| Contacts provider | Permet de gérer les contacts du téléphone. |
| Documents provider | Permet de manipuler des fichiers locaux et des fichiers sur le cloud. |
| MediaStore provider | Permet de manipuler des fichiers média (Image, vidéo, mp3...). |
| Settings provider | Permet de manipuler les données de configuration du téléphone. |
| Telephony provider | Permet de manipuler des données relatives à la téléphonie de l'appareil. |
| UserDictionary provider | Permet de manipuler des données relatives au dictionnaire de mots utilisateur. |

* Récupérer un objet de type ContentResolver fourni par le Context

Présentation

C'est le Context qui fournit le ContentResolver. Depuis n'importe quelle activité ou classe héritant du Context, il est possible d'accéder au ContentResolver :

val leContentResolver = contentResolver

L'objet contentResolver fournit les fonctions query(), insert(), update() et delete() nécessaires pour exploiter les fonctions de base d'un content provider. Les fonctions query(), insert(), update() et delete() attendent en premier paramètre l'URI du content provider à utiliser.

* Définir les permissions requises par le content provider

Pour connaître les permissions nécessaires pour utiliser un content provider : soit se référer à la documentation du content provider, soit étudier les informations fournies par l’erreur qui sera levée lors de la première utilisation du content provider.

* Usage

Insertion

Exemple utilisant un content provider pour insérer une personne en base de données. L'utilisation d'un nouveau thread est obligatoire afin de ne pas ralentir l'interface graphique.

Exemple d'insertion avec un content provider

/\*\*   
 \* BOUCHON   
 \*/   
val personneAInserer = ContentValues()   
personneAInserer.put("id", 0)   
personneAInserer.put("nom", "Momo")   
personneAInserer.put("prenom", "Touchtouch")   
/\*\*   
 \* INSERT   
 \*   
 \* Insertion de trois personnes   
 \*/   
var URL\_INSERT = "content://fr.acos.MonContentProvider/personnes"   
Thread({   
    leContentResolver.insert(Uri.parse(URL\_INSERT), personneAInserer)   
}).start()

Lecture

Exemple utilisant un content provider pour lire des enregistrements de personnes en base de données. L'utilisation d'un nouveau thread est obligatoire afin de ne pas ralentir l'interface graphique.

Exemple de sélection avec un content provider

/\*\*   
 \* QUERY ONE   
 \*   
 \* Récupération de la personne ayant l'id 1   
 \*/   
var URL\_RECUP\_ONE = "content://fr.acos.MonContentProvider/personnes/1"   
Thread(   
        {   
            val cursor = leContentResolver.query(Uri.parse(URL\_RECUP\_ONE),   
                    null,    
                    null,    
                    null,    
                    null)   
   
            while (cursor.moveToNext()) {   
                Log.i(TAG,    
                        "${cursor.getString(0)} - " +   
                                "${cursor.getString(1)} -" +   
                                " ${cursor.getString(2)}");   
            }   
        }).start()

Dans cet exemple, la boucle while utilisée avec le cursor permet de traiter les enregistrements les uns après les autres.

* Utilisation : accès aux contacts de votre téléphone
* Présentation

Android fournit un content provider nommé "contacts provider". Ce content provider gère les données des contacts du téléphone. Ce content provider gère de nombreuses données sur plusieurs sources de données. L'organisation de ce content provider est complexe, mais plusieurs classes et interfaces permettent de simplifier son utilisation.

* Fonctionnement

Le content provider offre l'accès à trois tables :

* La table ContactsContract.Contacts : contient des enregistrements représentant différentes personnes avec des informations d'agrégations de la table ContactsContract.RawContacts (cette table contient des contacts).
* La table ContactsContract.RawContacts : contient des enregistrements représentant des résumés de la table ContactsContract.Data (cette table contient des contacts bruts).
* La table ContactsContract.Data : contient des enregistrements contenant un détail d'un contact : adresses email, numéro de téléphone, etc.

Dans les explications qui suivent, il faut bien différencier un contact brut d'un contact.

* La table ContactsContract.RawContacts

Un enregistrement de cette table représente les données d'un contact provenant d'un seul type de compte et d'un nom de compte. (Google, Outlook par exemple). Une personne peut donc être liée à plusieurs contacts (on appelle ces contacts des contacts bruts). Pour résumer : cette table contient des enregistrements de contacts bruts.

La majorité des données liées à un contact brut se trouve dans la table ContactsContract.Data.

Principales colonnes de la table ContactsContract.RawContacts :

|  |  |
| --- | --- |
| Colonne | Description |
| \_ID | Identifiant unique du contact brut. Cet identifiant permet de faire le lien entre la table ContactsContract.RawContacts et la table ContactsContract.Data. |
| CONTACT\_ID | Identifiant unique du contact auquel est lié le contact brut. |
| ACCOUNT\_NAME | Le nom du compte pour le type de compte. Par exemple l'adresse antho.cosson@gmail.com pour un compte de type Google. |
| ACCOUNT\_TYPE | Le type de compte (Google, Outlook…). |

Le nom d'un contact brut est stocké dans un enregistrement de type ContactsContract.CommonDataKinds.StructuredName de la table ContactsContract.Data.

* La table ContactsContract.Data

La table ContactsContract.Data contient des enregistrements contenant les détails d'un contact. Chaque détail est un enregistrement de la table ContactsContract.Data. Si un contact brut a cinq adresses email, alors il y aura cinq enregistrements dans la table ContactsContract.Data pour enregistrer ces informations.

Le nom d'affichage, le numéro de téléphone, l'adresse électronique, l'adresse postale, la photo et les lignes de détails du site web, etc. se trouvent tous dans la table ContactsContract.Data.

La table ContactsContract.Data contient des colonnes avec des noms descriptifs et d'autres avec des noms génériques. Le contenu d'une colonne "nom descriptif" a la même signification, quel que soit le type de données de l'enregistrement, tandis que le contenu d'une colonne nom générique a des significations différentes selon le type de données.

Chaque enregistrement est lié à un type pour indiquer s'il est question d'une adresse email ou d'un numéro de téléphone, etc.

Principales colonnes de la table ContactsContract.RawContacts :

|  |  |
| --- | --- |
| Colonne | Description |
| RAW\_CONTACT\_ID | Identifiant du contact auquel appartient l'information. |
| MIMETYPE | Type d'enregistrement, le content provider utilise les types définis dans les sous-classes de ContactsContract.CommonDataKinds. |
| IS\_PRIMARY | C'est la donnée principale de son MIMETYPE pour le contact brut auquel elle appartient. |
| DATA1 à DATA15 | Colonnes de données génériques. C'est le MIMETYPE qui donne des informations sur le contenu de chaque colonne. |
| SYNC1 à SYNC4 | Colonnes génériques à utiliser par des adaptateurs de synchronisation. |

* La classe ContactsContract.CommonDataKinds

ContactsContract.CommonDataKinds fournit les types (d'enregistrements) utilisés par la table ContactsContract.RawContacts.

Type proposé par la classe ContactsContract.CommonDataKinds :

|  |  |
| --- | --- |
| Type | Description |
| Email | Type de donnée représentant une adresse email. |
| Event | Type de donnée représentant un événement. |
| Identity | Type de donnée représentant une identité liée au contact. |
| Nickname | Type de donnée représentant le surnom du contact. |
| Note | Type de donnée représentant une note sur le contact. |
| Organization | Type de donnée représentant une organisation. |
| Phone | Type de donnée représentant un numéro de téléphone. |
| Photo | Type de donnée représentant une photo. |
| Relation | Type de donnée représentant une relation. |
| StructuredName | Type de donnée représentant le nom propre du contact. |
| StructuredPostal | Type de donnée représentant une adresse postale. |
| Website | Type de donnée représentant un site internet. |

Pour faciliter l'accès aux colonnes intéressantes d'un type d'enregistrement, le provider fournit aussi des constantes de noms de colonnes spécifiques au type, ces constantes sont définies dans les sous-classes de ContactsContract.CommonDataKinds.

Exemples de constantes proposées pour le type Email :

|  |  |
| --- | --- |
| Constante | Valeur |
| ADDRESS | DATA1 (nom d'une colonne de la table RawContacts). |
| DISPLAY\_NAME | DATA4 (nom d'une colonne de la table RawContacts). |

* La table ContactsContract.Contacts

Le provider combine les lignes de contacts bruts sur tous les types de comptes et les noms de comptes pour former un seul contact et c'est donc dans la table ContactsContract.Contacts qu'est stockée la combinaison.

Le provider crée un nouveau contact lorsqu'un nouveau contact brut est créé et qu'il ne correspond à aucun contact existant.

Principales colonnes de la table ContactsContract.Contacts :

|  |  |
| --- | --- |
| Colonne | Description |
| \_ID | Identifiant unique du contact. Cet identifiant permet de faire le lien entre les tables Contacts Contract.RawContacts et ContactsContract.Data. |
| LOOKUP\_KEY | Valeur qui contient des indications sur la façon de trouver le contact si son identifiant de ligne a changé à la suite d'une synchronisation ou d'une agrégation. |
| NAME\_RAW\_CONTACT\_ID | ID du contact brut qui fournit le nom complet au contact agrégé. Lors de l'agrégation, l'un des contacts bruts constitutifs est choisi. |
| DISPLAY\_NAME\_PRIMARY | Le nom d'affichage pour le contact. C'est le nom d'affichage fourni par le contact brut référencé par la colonne NAME\_RAW\_CONTACT\_ID. |
| PHOTO\_ID | Référence de la ligne de la table ContactsContract.Data contenant la photo. |
| PHOTO\_URI | Un URI qui peut être utilisé pour récupérer la photo en taille réelle du contact. |
| PHOTO\_THUMBNAIL\_URI | Un URI qui peut être utilisé pour récupérer la vignette de la photo du contact. |
| IN\_VISIBLE\_GROUP | Indique si le contact doit être visible par l'utilisateur. |
| HAS\_PHONE\_NUMBER | Indique si le contact a un numéro de téléphone. |
| TIMES\_CONTACTED | Le nombre de fois que le contact a été contacté. |
| LAST\_TIME\_CONTACTED | Indique la dernière fois que le contact a été contacté. |
| STARRED | Indique que ce contact fait partie des contacts favoris. |
| CUSTOM\_RINGTONE | Une sonnerie personnalisée associée au contact. |
| SEND\_TO\_VOICEMAIL | Indique si les appels de ce contact doivent être transférés directement vers la messagerie vocale. |
| CONTACT\_PRESENCE | Statut de présence du contact. |
| CONTACT\_STATUS | Dernière mise à jour du statut du contact. |
| CONTACT\_STATUS\_TIMESTAMP | L'heure absolue en millisecondes lorsque le dernier statut a été mis à jour. |
| CONTACT\_STATUS\_RES\_PACKAGE | Le package contenant des ressources pour ce statut : libellé et icône. |

* Permission

La permission suivante est nécessaire pour accéder aux informations des contacts du téléphone :

android.permission.READ\_CONTACTS

* Usage

Dans cet exemple, l’activité va accéder aux contacts du téléphone puis les afficher dans les logs.

La démonstration est réalisée dans un nouveau projet dont l’activité principale se nomme MainActivity et est de type Empty Activity.

MainActivity.kt

package fr.acos.readcontactswithkotlin   
   
import android.Manifest   
import android.content.pm.PackageManager   
import android.support.v7.app.AppCompatActivity   
import android.os.Bundle   
import android.provider.ContactsContract   
import android.support.v4.app.ActivityCompat   
import org.jetbrains.anko.AnkoLogger   
import org.jetbrains.anko.info   
   
class MainActivity : AppCompatActivity(),AnkoLogger {   
   
    override fun onCreate(savedInstanceState: Bundle?) {   
        super.onCreate(savedInstanceState)   
        setContentView(R.layout.activity\_main)   
   
        //Envoi d'une requête de demande de permission pour READ\_CONTACTS   
        ActivityCompat.requestPermissions(this,   
arrayOf(Manifest.permission.READ\_CONTACTS), 14540)   
    }   
   
    override fun onRequestPermissionsResult(requestCode: Int,   
permissions: Array<String>, grantResults: IntArray)   
    {   
        super.onRequestPermissionsResult(requestCode, permissions,   
grantResults)   
   
        //Traitement effectué pour la demande de permission ayant   
        //l'identifiant 14540   
        if (requestCode == 14540)   
        {   
            //Test si l'utilisateur a donné sa permission pour la   
            //permission READ\_CONTACT   
            if (grantResults.size == 1   
                && grantResults[0] == PackageManager.PERMISSION\_GRANTED   
                    )   
            {   
                //Appel de la fonction qui utilise le content provider   
                consultationContacts()   
            }   
        }   
    }   
   
    /\*\*   
     \* Fonction permettant d'afficher dans les logs   
     \* l'id, le nom, les numéros de téléphone et emails des contacts   
     \* du téléphone.   
     \*/   
    fun consultationContacts()   
    {   
        //Liste des colonnes qui seront récupérées sur les contacts   
        val projection = arrayOf<String>(   
                ContactsContract.Contacts.\_ID,   
                ContactsContract.Contacts.DISPLAY\_NAME\_PRIMARY)   
   
        //Récupération de tous les contacts dans un cursor   
        val contacts =   
getContentResolver().query(ContactsContract.Contacts.CONTENT\_URI,  
projection,null,null,null)   
   
        //Itération sur tous les contacts   
        while (contacts.moveToNext())   
        {   
            //Récupération de l'identifiant du contact   
            val contactId =   
contacts.getString(contacts.getColumnIndex(ContactsContract.  
Contacts.NAME\_RAW\_CONTACT\_ID))   
            //Récupération de son nom principal   
            val contactNom =   
contacts.getString(contacts.getColumnIndex(ContactsContract.  
Contacts.DISPLAY\_NAME\_PRIMARY))   
            //Affichage dans les logs   
            info("ID = $contactId ; Nom = $contactNom")   
   
   
            //Récupération de tous les numéros de téléphone de tous   
            //les contacts bruts du contact non brut   
            val telephones = getContentResolver().  
query(ContactsContract.CommonDataKinds.Phone.CONTENT\_URI, null,   
ContactsContract.CommonDataKinds.Phone.CONTACT\_ID + " = " +   
contactId, null, null)   
            //Itération sur tous les numéros de téléphone   
            while (telephones.moveToNext())   
            {   
                //Récupération du numéro de téléphone   
                val numero = telephones.getString(telephones.getColumnIndex  
(ContactsContract.CommonDataKinds.Phone.NUMBER))   
                //Affichage dans les logs   
                info("Telephone = $numero")   
            }   
            telephones.close()   
   
            //Récupération de tous les emails de tous les contacts   
            //bruts du contact non brut   
            val emails =   
getContentResolver().query(ContactsContract.CommonDataKinds.  
Email.CONTENT\_URI, null, ContactsContract.CommonDataKinds.  
Email.CONTACT\_ID + " = " + contactId, null, null)   
            //Itération sur tous les emails   
            while (emails.moveToNext())   
            {   
                //Récupération de l'email   
                val mail =   
emails.getString(emails.getColumnIndex(ContactsContract.  
CommonDataKinds.Email.DATA))   
                //Affichage dans les logs   
                info("EMail = $mail")   
            }   
            emails.close()   
        }   
        contacts.close()   
    }   
}

* Démonstration : création d'un content provider
* Présentation

Cet exemple montre comment exposer les données stockées via l'ORM Room (cf. chapitre Android : quelques librairies incontournables - Room) avec un content provider.

* Sources

Cette classe va permettre à Room de créer une table personnes. Cette table contient trois colonnes :

Personne.kt

package fr.acos.contentpriverwithkotlin.entities   
   
import android.arch.persistence.room.ColumnInfo   
import android.arch.persistence.room.Entity   
import android.arch.persistence.room.PrimaryKey   
   
@Entity(tableName = "personnes")   
data class Personne(   
        @ColumnInfo(name="id") @PrimaryKey(autoGenerate = true) var   
id: Long = 0,   
        @ColumnInfo(name="nom") var nom: String = "",   
        @ColumnInfo(name="prenom") var prenom: String = ""   
)   
{   
    companion object {   
        val TABLE\_NAME = "personnes"   
    }   
}

DAO offrant des fonctions manipulant la bdd :

PersonneDAO.kt

package fr.acos.dao   
   
import android.arch.persistence.room.\*   
import android.database.Cursor   
import fr.acos.contentpriverwithkotlin.entities.Personne   
   
@Dao   
interface PersonneDao {   
   
    @Query("SELECT \* FROM personnes")   
    fun get(): List<Personne>   
   
    @Query("SELECT \* FROM personnes")   
    fun getInCursor(): Cursor   
   
    @Query("SELECT \* FROM personnes WHERE id = :id")   
    fun get(id: Long): Personne   
   
    @Query("SELECT \* FROM personnes WHERE id = :id")   
    fun getInCursor(id: Long): Cursor   
   
    @Insert   
    fun insertAll(vararg listePersonne: Personne)   
   
    @Insert   
    fun insert(personne: Personne):Long   
   
    @Update   
    fun updatePersonne(item: Personne):Int   
   
    @Delete   
    fun deletePersonne(item: Personne):Int   
}

Classe de type Database représentant la base de données et fournissant l'accès aux Dao :

MaBaseDeDonnees.kt

package fr.acos.dao   
   
import android.arch.persistence.room.Database   
import android.arch.persistence.room.RoomDatabase   
import fr.acos.contentpriverwithkotlin.entities.Personne   
import android.arch.persistence.room.Room   
import android.content.Context   
   
   
@Database(entities = [(Personne::class)], version = 1)   
abstract class MaBaseDeDonnees : RoomDatabase()   
{   
    /\*\*   
     \* Singleton   
     \*/   
    companion object {   
        private var sInstance: MaBaseDeDonnees? = null   
   
        fun getInstance(context: Context): MaBaseDeDonnees {   
            if (sInstance == null) {   
                sInstance = Room   
.databaseBuilder(context.getApplicationContext(),   
MaBaseDeDonnees::class.java!!, "personnesbdd.db")   
                        .build()   
            }   
            return sInstance as MaBaseDeDonnees   
        }   
    }   
   
    abstract fun personneDao(): PersonneDao   
}

Content provider complet manipulant la base de données avec Room :

MonContentProvider.kt

package fr.acos.contentpriverwithkotlin.provider   
   
import android.content.ContentProvider   
import android.content.ContentUris   
import android.content.ContentValues   
import android.content.UriMatcher   
import android.database.Cursor   
import android.net.Uri   
import android.util.Log   
import fr.acos.contentpriverwithkotlin.entities.Personne   
import fr.acos.dao.MaBaseDeDonnees   
   
/\*\*   
 \* Classe représentant le ContentProvider   
 \*/   
class MonContentProvider : ContentProvider()   
{   
    companion object   
        {   
            private val TAG = "XXX"   
            /\*   
            \* Nom de content provider   
            \*/   
            private val AUTHORITY = "fr.acos.MonContentProvider"   
   
            /\*\*   
             \* Code indiquant qu'il y a un id de personne dans l'URI   
             \*/   
            private val CODE\_PERSONNE\_DIR = 1   
   
            /\*\*   
             \* Code indiquant qu'il n'y a pas d'id de personne dans l'URI   
             \*/   
            private val CODE\_PERSONNE\_ITEM = 2   
   
            /\*\*   
             \* Objet de type UriMatcher permettant d'analyser facilement   
             \*les URI.   
             \*/   
            private val MATCHER = UriMatcher (UriMatcher.NO\_MATCH)   
   
            /\*\*   
             \* Utilisé pour la fonction getType()   
             \*/   
            private val TYPE\_MIME\_PERSONNE = "vnd.android.cursor.item/  
fr.acos.contentpriverwithkotlin.entities.Personne";   
        }   
   
    init   
    {   
        //Initialisation de l'objet MATCHER   
        //Le code CODE\_PERSONNE\_DIR est associé à l'URI   
content://fr.acos.MonContentProvider/personnes   
        MATCHER.addURI(   
                AUTHORITY,   
                Personne.TABLE\_NAME,   
                CODE\_PERSONNE\_DIR   
        );   
        //Le code CODE\_PERSONNE\_DIR est associé à l'URI   
        //content://fr.acos.MonContentProvider/personnes/\*   
        // \* représente n'importe quel nombre. Ce nombre correspond à un   
        //identifiant de personne.   
        MATCHER.addURI(   
                AUTHORITY,   
                "${Personne.TABLE\_NAME}/\*",   
                CODE\_PERSONNE\_ITEM   
        );   
    }   
   
    /\*\*   
     \* Permet d’initialiser les ressources pour le ContentProvider.   
     \*   
     \* @return Boolean   
     \*/   
    override fun onCreate(): Boolean   
    {   
        return false   
    }   
   
    /\*\*   
     \* Fonction permettant d'ajouter une personne   
     \*   
     \* @param uri URI du content provider   
     \* @param values informations relatives à la personne à inserer.   
     \*/   
    override fun insert(uri: Uri, values: ContentValues?): Uri?   
    {   
        Log.i(TAG,"Fonction insert. uri = $uri")   
        when (MATCHER.match(uri))   
        {   
            //Si l'URI est égal à content://fr.acos.MonContentProvider/  
            //personnes   
            CODE\_PERSONNE\_DIR ->   
            {   
                //Création d'un objet représentant la base de données.   
                val db = MaBaseDeDonnees.getInstance(context)   
                //Création d'un objet représentant la DAO pour les   
                //personnes.   
                val dao = db.personneDao()   
                //Transformation du ContentValue en objet de type Personne.   
                val personneAInserer = itemBuilder(values)   
                //Insertion en base. id est égal à l'identifiant du nouvel   
                //enregistrement.   
                val id = dao.insert(personneAInserer)   
                //Retourn l'URI avec le nouvel identifiant à la fin.   
                return ContentUris.withAppendedId(uri,id)   
            }   
            //Si l'URI est égal à content://fr.acos.MonContentProvider/  
            //personnes/\*   
            CODE\_PERSONNE\_ITEM -> throw IllegalArgumentException("Pas d'ID   
nécessaire pour l'insertion")   
            else -> throw IllegalArgumentException("Informations erronées")   
        }   
    }   
   
    /\*\*   
     \* Fonction permettant de supprimer une personne.   
     \*   
     \* @param uri URI du content provider   
     \* @param selection inutile   
     \* @param selectionArgs inutile   
     \*/   
    override fun delete(uri: Uri, selection: String?, selectionArgs:   
Array<String>?): Int   
    {   
        Log.i(TAG,"Fonction delete. uri = $uri")   
   
        when (MATCHER.match(uri))   
        {   
            //Si l'URI est égal à content://fr.acos.MonContentProvider/  
            //personnes/\*   
            CODE\_PERSONNE\_ITEM ->   
            {   
                //Récupération de l'identifiant à la fin de l'URI   
                val id = uri.lastPathSegment   
                //Création d'un objet représentant la base de données.   
                val db =  MaBaseDeDonnees.getInstance(context)   
                //Création d'un objet représentant la DAO pour les   
                //personnes.   
                val dao = db.personneDao()   
                //Création d'un objet de type Personne avec seulement un   
                //identifiant.   
                val personneASupprimer = Personne(id.toLong())   
                //Suppression de la base de données.   
                val nombreDeSuppression =   
dao.deletePersonne(personneASupprimer)   
                //retourne le nombre de lignes supprimées.   
                return nombreDeSuppression   
            }   
            //Si l'URI est égal à content://fr.acos.MonContentProvider/  
            //personnes   
            CODE\_PERSONNE\_DIR ->throw IllegalArgumentException("ID   
nécessaire pour la suppression")   
            else -> throw IllegalArgumentException("Informations erronées")   
        }   
    }   
   
    /\*\*   
     \* Fonction permettant de récupérer des personnes.   
     \*   
     \* @param uri URI du content provider   
     \* @param projection inutile   
     \* @param selection inutile   
     \* @param selectionArgs inutile   
     \* @param sortOrder inutile   
     \*/   
    override fun query(uri: Uri, projection: Array<String>?, selection:   
String?,   
                       selectionArgs: Array<String>?, sortOrder: String?):   
Cursor?   
    {   
        Log.i(TAG,"Fonction query. uri = $uri")   
        Log.i(TAG,"Fonction query. Matcher.match(uri) =   
${MATCHER.match(uri)}")   
        var cursor:Cursor? = null   
        //Création d'un objet représentant la base de données.   
        val db = MaBaseDeDonnees.getInstance(context)   
        //Création d'un objet représentant la DAO pour les personnes.   
        val dao = db.personneDao()   
   
        when (MATCHER.match(uri))   
        {   
            //Si l'URI est égal à content://fr.acos.MonContentProvider/  
            //personnes/\*   
            CODE\_PERSONNE\_ITEM ->   
            {   
                //Récupération de l'identifiant à la fin de l'URI   
                val id = uri.lastPathSegment   
                //Retourne un objet de type Cursor contenant une personne.   
                cursor = dao.getInCursor(id.toLong())   
            }   
            //Si l'URI est égal à content://fr.acos.MonContentProvider/   
            //personnes   
            CODE\_PERSONNE\_DIR ->   
            {   
                //Retourne un objet de type Cursor contenant plusieurs   
                //personnes.   
                cursor = dao.getInCursor()   
            }   
            else -> throw IllegalArgumentException("Informations erronées")   
        }   
        return cursor;   
    }   
   
    /\*\*   
     \* Fonction permettant de mettre à jour des personnes   
     \*   
     \* @param uri URI du content provider   
     \* @param values  informations relatives à la personne à modifier.   
     \* @param selection inutile   
     \* @param selectionArgs inutile   
     \*/   
    override fun update(uri: Uri, values: ContentValues?, selection:   
String?,   
                        selectionArgs: Array<String>?): Int {   
        Log.i(TAG,"Fonction update. uri = $uri")   
        Log.i(TAG,"Fonction update. Matcher.match(uri) =   
${MATCHER.match(uri)}")   
   
        when (MATCHER.match(uri))   
        {   
            //Si l'URI est égal à content://fr.acos.MonContentProvider/  
            //personnes/\*   
            CODE\_PERSONNE\_DIR ->   
            {   
                //Création d'un objet représentant la base de données.   
                val db = MaBaseDeDonnees.getInstance(context)   
                //Création d'un objet représentant la DAO pour les personnes.   
                val dao = db.personneDao()   
                //Transformation du ContentValue en objet de type Personne.  
                val personneAMettreAJour = itemBuilder(values);   
                //Mise à jour en base de données.   
                val nombreDeMiseAjour =   
dao.updatePersonne(personneAMettreAJour)   
                //Retourne le nombre d'éléments mis à jour.   
                return nombreDeMiseAjour   
            }   
            //Si l'URI est égal à content://fr.acos.MonContentProvider/  
            //personnes   
            CODE\_PERSONNE\_ITEM -> throw IllegalArgumentException("ID non   
nécessaire pour la mise à jour")   
            else -> throw IllegalArgumentException("Informations erronées")   
        }   
    }   
   
    /\*\*   
     \* Fonction permettant d'indiquer le type des données traitées par le   
ContentProvider.   
     \*   
     \* @param uri URI du content provider   
     \*/   
    override fun getType(uri: Uri) =   
            when (MATCHER.match(uri))   
            {   
                CODE\_PERSONNE\_DIR,CODE\_PERSONNE\_ITEM -> TYPE\_MIME\_PERSONNE;   
                else -> null   
            }   
   
    /\*\*   
     \* Permet de créer un objet de type Personne à l'aide d'un ContentValue   
     \*   
     \* @param valeurs objet de type ContentValues contenant les   
informations sur une personne.   
     \*   
     \* @return Un objet de type Personne   
     \*/   
    fun itemBuilder(valeurs:ContentValues?)  = Personne (   
                valeurs?.getAsLong("id")!!,   
                valeurs?.getAsString("nom"),   
                valeurs?.getAsString("prenom"))   
}

Création de deux permissions dans le manifeste et association de ces permissions au content provider :

AndroidManifest.kt

<?xml version="1.0" encoding="utf-8"?>   
<manifest xmlns:android="http://schemas.android.com/apk/res/android"   
    package="fr.acos.contentpriverwithkotlin">   
   
    <permission   
        android:name="fr.acos.contentpriverwithkotlin.READ"   
        android:label="Lecture"   
        android:description="@string/description\_droit\_lire\_personne"   
        android:permissionGroup="android.permission-group.STORAGE"   
        android:protectionLevel="dangerous" />   
   
    <permission   
        android:name="fr.acos.contentpriverwithkotlin.WRITE"   
        android:label="Ecriture"   
        android:description="@string/description\_droit\_ecrire\_personne"   
        android:permissionGroup="android.permission-group.STORAGE"   
        android:protectionLevel="dangerous" />   
   
    <application   
        android:allowBackup="true"   
        android:icon="@mipmap/ic\_launcher"   
        android:label="CP With Kotlin"   
        android:roundIcon="@mipmap/ic\_launcher\_round"   
        android:supportsRtl="true"   
        android:theme="@style/AppTheme">   
        <activity android:name=".MainActivity">   
            <intent-filter>   
                <action android:name="android.intent.action.MAIN" />   
   
                <category android:name="android.intent.category.LAUNCHER" />   
            </intent-filter>   
        </activity>   
   
        <provider   
            android:name=".provider.MonContentProvider"   
            android:authorities="fr.acos.MonContentProvider"   
            android:enabled="true"   
            android:exported="true"   
            android:readPermission="fr.acos.contentpriverwithkotlin.READ"   
            android:writePermission="fr.acos.contentpriverwithkotlin.WRITE"   
            ></provider>   
    </application>   
   
</manifest>

Utilisation sommaire du content provider via la classe MainActivity :

MainActivity.kt

package fr.acos.contentpriverwithkotlin   
   
import android.support.v7.app.AppCompatActivity   
import android.os.Bundle   
import android.content.ContentValues   
import android.net.Uri   
import android.util.Log   
   
   
class MainActivity : AppCompatActivity() {   
   
    companion object {   
        private val TAG = "XXX"   
    }   
   
    override fun onCreate(savedInstanceState: Bundle?) {   
        super.onCreate(savedInstanceState)   
        setContentView(R.layout.activity\_main)   
   
        /\*\*   
         \* Manipulation du ContentProvider personnel   
         \*/   
        //Récupération du contentResolver   
        val cr = contentResolver   
   
        /\*\*   
         \* BOUCHON   
         \*/   
        val personneAInserer = ContentValues()   
        personneAInserer.put("id", 0)   
        personneAInserer.put("nom", "nom\_test")   
        personneAInserer.put("prenom", "prenom\_test")   
   
        val personneAUpdater = ContentValues()   
        personneAUpdater.put("id", 3)   
        personneAUpdater.put("nom", "modif\_nom")   
        personneAUpdater.put("prenom", "modif\_prenom")   
   
        /\*\*   
         \* INSERT   
         \*   
         \* Insertion de trois personnes   
         \*/   
        var URL\_INSERT = "content://fr.acos.MonContentProvider/personnes"   
        Thread({   
            cr.insert(Uri.parse(URL\_INSERT), personneAInserer)   
            cr.insert(Uri.parse(URL\_INSERT), personneAInserer)   
            cr.insert(Uri.parse(URL\_INSERT), personneAInserer)   
        }).start()   
   
        /\*\*   
         \* DELETE   
         \*   
         \* Suppression de la personne ayant l'id 1.   
         \*/   
        var URL\_DELETE = "content://fr.acos.MonContentProvider/personnes/1"   
        Thread(   
                {   
                    Log.i(TAG, "DELETE Retour :   
${cr.delete(Uri.parse(URL\_DELETE), null, null)}")   
                }).start()   
   
        /\*\*   
         \* UPDATE   
         \*   
         \* Mise à jour de la personne ayant l'id 3   
         \*/   
        var URL\_UPDATE = "content://fr.acos.MonContentProvider/personnes"   
        Thread({ cr.update(Uri.parse(URL\_UPDATE),   
personneAUpdater,null,null) }).start()   
   
        /\*\*   
         \* QUERY ALL   
         \*   
         \* Récupération de toutes les personnes.   
         \*/   
        var URL\_RECUP = "content://fr.acos.MonContentProvider/personnes"   
        Thread(   
                {   
                    val cursor = cr.query(Uri.parse(URL\_RECUP),   
null, null, null, null)   
   
                    while (cursor.moveToNext()) {   
                        Log.i(TAG, "${cursor.getString(0)} -   
${cursor.getString(1)} - ${cursor.getString(2)}");   
                    }   
                }).start()   
   
        /\*\*   
         \* QUERY ONE   
         \*   
         \* Récupération de la personne ayant l'id 2   
         \*/   
        var URL\_RECUP\_ONE = "content://fr.acos.MonContentProvider/  
personnes/1"   
        Thread(   
                {   
                    val cursor = cr.query(Uri.parse(URL\_RECUP\_ONE),   
null, null, null, null)   
   
                    while (cursor.moveToNext()) {   
                        Log.i(TAG, "${cursor.getString(0)} -   
${cursor.getString(1)} - ${cursor.getString(2)}");   
                    }   
                }).start()   
    }   
   
}

Dans cet exemple, on présente comment créer et exploiter un content provider personnel.

* Utilisation de données JSON
* Présentation

JSON (JavaScript Object Notation) permet de stocker des données de manière structurée tout comme peut le faire le XML, à la différence qu'un fichier JSON est plus léger qu'un fichier XML avec une même quantité de données. C'est la raison pour laquelle JSON prend largement la place du XML dans les échanges de données et notamment dans la communication avec des web services.

* Formats

Des données JSON peuvent arriver sous deux formats :

* objet JSON : ensemble de couples nom/valeur non ordonnés
* tableau JSON : ensemble de valeurs ordonnées

Les valeurs peuvent être :

* des valeurs scalaires
* des objets JSON
* des tableaux JSON

Exemple d'un tableau JSON

[ "Bleu", "Blanc", "Rouge" ]

Exemple d'un objet JSON

{    
   
"nom": "Rob",   
   
"prénom": "Pierre",   
   
      "plats\_preferes": [    
   
           { "nom": "Pâtes à la crème", "nombre\_ingredients": 2 } ,   
   
           { "nom": "Omelette", " nombre\_ingredients ": 1 }   
   
      ]   
   
    }

* Transformer des données JSON en objets Kotlin
* Manière historique

Pour transformer des données JSON en objets Kotlin, il existe une librairie fournie nativement : org.json.JSONObject.

La classe JSONObject permet de transformer un objet JSON en objet Kotlin.

La classe JSONArray permet de transformer un tableau JSON en objet Kotlin représentant ce tableau.

Exemple d'utilisation avec objet JSON

val json = "{'Pays':" +   
        "['France'," +   
        "'Afrique du sud'," +   
        "'Burkina Faso'," +   
        "'Irlande'," +   
        "'Portugal'," +   
        "'Suisse']}"   
   
    val jObject = JSONObject(json)   
    val jArrayPays = jObject.getJSONArray("Pays")   
   
    for (i in 0 until jArrayPays.length())   
    {   
        Log.i("XXX", "Valeur à l'index $i = ${jArrayPays.get(i)}")   
    }   
}   
   
Résultat :   
   
XXX: Valeur à l'index 0 = France   
   
XXX: Valeur à l'index 1 = Afrique du sud   
   
XXX: Valeur à l'index 2 = Burkina Faso   
   
XXX: Valeur à l'index 3 = Irlande   
   
XXX: Valeur à l'index 4 = Portugal   
   
XXX: Valeur à l'index 5 = Suisse

Exemple d'utilisation avec un tableau JSON

val tableauDObjets = "[{'Designation':'ROADSTER',  
'ModeleCommercial':'ROADSTER','CNIT':'M10TSLVP000C002',  
'Id':41124,'ModeleDossier':'ROADSTER'},{'Designation':'ROADSTER',  
'ModeleCommercial':'ROADSTER','CNIT':'M10TSLVP000D003',  
'Id':41125,'ModeleDossier':'ROADSTER'}]";   
   
val array = JSONArray(tableauDObjets);   
Log.i("XXX","Nombre d'objets dans le tableau : " + array.length());   
   
for(i in 0 until array.length())   
{   
        Log.i("XXX","---------------------------------------");   
        val objet = array.getJSONObject(i);   
        Log.i("XXX","Contenu de l'objet numéro  :  $i  :   
${objet.toString()}");   
        Log.i("XXX","Désignation de la voiture :   
${objet.get("Designation")}");   
        Log.i("XXX","Modele commercial de la voiture :   
${objet.get("ModeleCommercial")}");   
        Log.i("XXX","CNIT de la voiture : ${objet.get("CNIT")}");   
        Log.i("XXX","Id de la voiture : ${objet.get("Id")}");   
        Log.i("XXX","Modele dossier de la voiture :   
${objet.get("ModeleDossier")}");   
        Log.i("XXX","---------------------------------------");   
}   
   
Résultat :   
   
XXX: Nombre d'objets dans le tableau : 2   
   
XXX: ---------------------------------------   
   
XXX: Contenu de l'objet numéro  :  0  :   
{"Designation":"ROADSTER","ModeleCommercial":"ROADSTER","CNIT":  
"M10TSLVP000C002","Id":41124,"ModeleDossier":"ROADSTER"}   
   
XXX: Désignation de la voiture : ROADSTER   
   
XXX: Modèle commercial de la voiture : ROADSTER   
   
XXX: CNIT de la voiture : M10TSLVP000C002   
   
XXX: Id de la voiture : 41124   
   
XXX: Modèle dossier de la voiture : ROADSTER   
   
XXX: ---------------------------------------   
   
XXX: ---------------------------------------   
   
XXX: Contenu de l'objet numéro  :  1  :   
{"Designation":"ROADSTER","ModeleCommercial":"ROADSTER","CNIT":  
"M10TSLVP000D003","Id":41125,"ModeleDossier":"ROADSTER"}   
   
XXX: Désignation de la voiture : ROADSTER   
   
XXX: Modèle commercial de la voiture : ROADSTER   
   
XXX: CNIT de la voiture : M10TSLVP000D003   
   
XXX: Id de la voiture : 41125   
   
XXX: Modèle dossier de la voiture : ROADSTER   
   
XXX: ---------------------------------------

* Utilisation de services distants
* Présentation

Il est fréquent de vouloir accéder à des services distants (web service), sur Internet, pour récupérer certaines informations. Dans ce chapitre, il est présenté comment envoyer une requête de demande d'informations à un service distant et comment récupérer la réponse provenant de ce service.

* Permission

La permission suivante doit être définie dans le fichier AndroidManifest.xml :

<uses-permission android:name="android.permission.INTERNET" />

Cette permission permet d’accéder à internet.

* Consommer un web service

Pour consommer un web service, il suffit de créer un objet de type URL qui attend en paramètre l'URL du web service, puis d'utiliser la fonction readText() pour lire la réponse du web service. Il est obligatoire de créer un nouveau thread pour consommer un web service afin de ne pas bloquer l'interface de l'application.

Syntaxe

val reponse = URL("<url\_web\_service>").readText()

Le paramètre url\_web\_service est le lien qui représente l'adresse HTTP du service distant.

Exemple de consommation d'un web service

Thread(   
        {   
            val result = URL("https://api.chucknorris.io/jokes/  
random").readText()   
            Log.i("XXX","Retour : $result");   
        }   
).start()

Dans cet exemple, une requête est envoyée au service distant situé à l'adresse : https://api.chucknorris.io/jokes/random. La réponse du service distant est enregistrée dans la variable result. La réponse est renvoyée sous forme de JSON.

* Démonstration

Dans cette démonstration, il va être présenté une application avec un bouton. Lors d'un clic sur le bouton, un appel à un service distant est fait. Le résultat est récupéré sous forme de JSON, puis parsé en objet et finalement affiché dans l'IHM.

La démonstration est réalisée dans un nouveau projet dont l'activité principale se nomme MainActivity.

AndroidManifest.xml

<?xml version="1.0" encoding="utf-8"?>   
<manifest xmlns:android="http://schemas.android.com/apk/res/android"   
    package="fr.acos.espressowithkotlin">   
   
    <uses-permission android:name="android.permission.INTERNET" />   
   
    <application   
        android:allowBackup="true"   
        android:icon="@mipmap/ic\_launcher"   
        android:label="@string/app\_name"   
        android:roundIcon="@mipmap/ic\_launcher\_round"   
        android:supportsRtl="true"   
        android:theme="@style/AppTheme">   
        <activity android:name=".MainActivity">   
            <intent-filter>   
                <action android:name="android.intent.action.MAIN" />   
   
                <category android:name="android.intent.category.LAUNCHER" />   
            </intent-filter>   
        </activity>   
    </application>   
   
</manifest>

Pour accéder à un service distant, l'application doit avoir la permission de se connecter à Internet.

Fact.kt

package fr.acos.chucknorrisrandomfactwithkotlin.bo   
   
/\*\*   
 \* Classe permettant d'enregistrer un fait.   
 \*/   
data class Fact(var id:String,var fact:String)

La classe Fact permet de stocker la réponse du service distant.

FactDao.kt

package fr.acos.chucknorrisrandomfactwithkotlin.dao   
   
import android.util.Log   
import fr.acos.chucknorrisrandomfactwithkotlin.bo.Fact   
import org.json.JSONObject   
import java.net.URL   
   
/\*\*   
 \* Classe permettant d'accéder à l'API fournissant les Fact   
 \*/   
class FactDao {   
   
    companion object {   
        /\*\*   
         \* Fonction qui accède à l'API   
         \*/   
        fun getFact(): Fact {   
            var result: String   
            result = URL("https://api.chucknorris.io/jokes/  
random").readText()   
            return parseJSonToFact(result)   
        }   
   
        /\*\*   
         \* Fonction qui parse le JSON en objet Fact.   
         \*/   
        private fun parseJSonToFact(json:String):Fact   
        {   
            val jObject = JSONObject(json)   
            val id = jObject.get("id").toString()   
            val fact = jObject.get("value").toString()   
            val result =  Fact(id,fact)   
            Log.i("XXX",result.toString())   
            return result   
        }   
    }   
}

Cette classe contient deux fonctions :

* getFact(): cette fonction accède au service distant, récupère la réponse sous forme de JSON et la parse à l'aide de la fonction parseJSonToFact().
* parseJSonToFact(): cette fonction parse la réponse JSON en objet de type Fact.

Activity\_main.xml

<?xml version="1.0" encoding="utf-8"?>   
<android.support.constraint.ConstraintLayout   
xmlns:android="http://schemas.android.com/apk/res/android"   
    xmlns:app="http://schemas.android.com/apk/res-auto"   
    xmlns:tools="http://schemas.android.com/tools"   
    android:layout\_width="match\_parent"   
    android:layout\_height="match\_parent"   
    tools:context="fr.acos.espressowithkotlin.MainActivity">   
   
    <TextView   
        android:id="@+id/tv\_fact"   
        android:layout\_width="wrap\_content"   
        android:layout\_height="wrap\_content"   
        android:text="Hello World!"   
        app:layout\_constraintBottom\_toBottomOf="parent"   
        app:layout\_constraintLeft\_toLeftOf="parent"   
        app:layout\_constraintRight\_toRightOf="parent"   
        app:layout\_constraintTop\_toTopOf="parent" />   
   
    <Button   
        android:id="@+id/btn\_aleatoire"   
        android:layout\_width="wrap\_content"   
        android:layout\_height="wrap\_content"   
        android:text="Random Fact"   
        app:layout\_constraintLeft\_toLeftOf="parent"   
        app:layout\_constraintRight\_toRightOf="parent"   
        app:layout\_constraintTop\_toBottomOf="@id/tv\_fact"   
        android:onClick="onClickButtonRandom"   
        />   
   
</android.support.constraint.ConstraintLayout>

Cette IHM contient un bouton pour faire appel au service distant et une TextView pour afficher la réponse.

MainActivity.kt

package fr.acos.chucknorrisrandomfactwithkotlin   
   
import android.support.v7.app.AppCompatActivity   
import android.os.Bundle   
import android.view.View   
import fr.acos.chucknorrisrandomfactwithkotlin.dao.FactDao   
import kotlinx.android.synthetic.main.activity\_main.\*   
   
class MainActivity : AppCompatActivity() {   
   
    override fun onCreate(savedInstanceState: Bundle?)   
    {   
        super.onCreate(savedInstanceState)   
        setContentView(R.layout.activity\_main)   
    }   
   
    fun onClickButtonRandom(view: View)   
    {   
        Thread({   
            //Consommation du web service   
            val fact  = FactDao.getFact()   
            //Quand le web service renvoie une réponse : mise à   
            //jour de l'IHM   
            runOnUiThread({   
                tv\_fact.text = fact.fact   
            })   
        }).start()   
    }   
}

L'activité MainActivity contient la fonction onClickButtonRandom(), cette fonction est exécutée lors du clic sur le bouton de l'IHM. L'accès à un service distant peut être long, c'est pourquoi le traitement de cette fonction est mis dans un thread afin de ne pas bloquer l'IHM.

* Les WebViews
* Présentation

Une WebView permet d'afficher une page web au sein d'une application. Le code des pages web (HTML, CSS et JS) peut être stocké sur des sites distants ou bien en local.

Les étapes pour définir une WebView dans une application sont les suivantes :

* Déclarer la WebView dans le layout de l’IHM.
* Charger du contenu dans la WebView.
* Déclaration dans l'interface

Voici comment définir une WebView dans un fichier d'interface. Les trois attributs permettent simplement de définir l'identifiant et la taille de la WebView.

Activity\_main.xml

<?xml version="1.0" encoding="utf-8"?>   
<android.support.constraint.ConstraintLayout   
xmlns:android="http://schemas.android.com/apk/res/android"   
    xmlns:app="http://schemas.android.com/apk/res-auto"   
    xmlns:tools="http://schemas.android.com/tools"   
    android:layout\_width="match\_parent"   
    android:layout\_height="match\_parent"   
    tools:context="fr.acos.webviewwebdistantwithkotlin.MainActivity">   
   
    <WebView   
        android:id="@+id/ma\_webview"   
        android:layout\_width="match\_parent"   
        android:layout\_height="match\_parent">   
    </WebView>   
   
</android.support.constraint.ConstraintLayout>

* Charger une page web dans une WebView

La fonction loadUrl() de la classe WebView permet de charger une page web. La page web peut être locale ou bien distante. Les deux sont présentées dans les points ci-dessous.

* Accès à des pages web distantes

Présentation

Pour accéder à des pages web distantes, il suffit de connaître l'adresse HTTP de la ressource à afficher et déclarer dans le manifeste la permission permettant d'accéder à Internet.

Permission

La permission suivante doit être définie dans le fichier AndroidManifest.xml :

<uses-permission android:name="android.permission.INTERNET" />

Usage

AndroidManifest.xml

<?xml version="1.0" encoding="utf-8"?>   
<manifest xmlns:android="http://schemas.android.com/apk/res/android"   
    package="fr.acos.webviewwebdistantwithkotlin">   
   
    <uses-permission android:name="android.permission.INTERNET" />   
   
    <application   
        android:allowBackup="true"   
        android:icon="@mipmap/ic\_launcher"   
        android:label="@string/app\_name"   
        android:roundIcon="@mipmap/ic\_launcher\_round"   
        android:supportsRtl="true"   
        android:theme="@style/AppTheme">   
        <activity android:name=".MainActivity">   
            <intent-filter>   
                <action android:name="android.intent.action.MAIN" />   
   
                <category android:name="android.intent.category.LAUNCHER" />   
            </intent-filter>   
        </activity>   
    </application>   
   
</manifest>

Dans cet exemple, la déclaration de la permission android.permission.  
INTERNET permet à la WebView d'accéder à des pages web distantes.

activity\_main.xml

<?xml version="1.0" encoding="utf-8"?>   
<android.support.constraint.ConstraintLayout   
xmlns:android="http://schemas.android.com/apk/res/android"   
    xmlns:app="http://schemas.android.com/apk/res-auto"   
    xmlns:tools="http://schemas.android.com/tools"   
    android:layout\_width="match\_parent"   
    android:layout\_height="match\_parent"   
   
tools:context="fr.acos.webviewwebdistantwithkotlin.MainActivity">   
   
     <WebView   
        android:id="@+id/ma\_webview"   
        android:layout\_width="match\_parent"   
        android:layout\_height="match\_parent">   
    </WebView>   
   
</android.support.constraint.ConstraintLayout>

Dans cet exemple, l'IHM de l'activité contient uniquement une WebView aussi large et haute que l'écran du téléphone.

MainActivity.kt

package fr.acos.webviewwebdistantwithkotlin   
   
import android.support.v7.app.AppCompatActivity   
import android.os.Bundle   
import android.view.KeyEvent   
import android.webkit.WebResourceRequest   
import android.webkit.WebView   
import android.webkit.WebViewClient   
import kotlinx.android.synthetic.main.activity\_main.\*   
   
/\*\*   
 \* Classe de l'activité pour tester les WebView.   
 \*/   
class MainActivity : AppCompatActivity() {   
   
    override fun onCreate(savedInstanceState: Bundle?) {   
        super.onCreate(savedInstanceState)   
        setContentView(R.layout.activity\_main)   
   
        //Chargement d'une page web distante.   
        ma\_webview.loadUrl("https://www.wikipedia.org")   
   
        //Activation du javaScript.   
        ma\_webview.settings.javaScriptEnabled = true   
   
        //Pour rester sur l'application lors d'un clic sur un lien.   
        ma\_webview.webViewClient = object : WebViewClient()   
        {   
            //Lors d'un clic sur un lien.   
            override fun shouldOverrideUrlLoading(view: WebView?,   
request: WebResourceRequest?): Boolean   
            {   
                //Le lien est chargé au sein de la WebView   
                view!!.loadUrl(request!!.url.toString())   
                return true;   
            }   
        }   
    }   
   
    /\*\*   
     \* Modification du comportement du bouton BACK.   
     \* Pour ne pas quitter l'application mais   
     \* retourner sur la page web précédente.   
     \*/   
    override fun onKeyDown(keyCode: Int, event: KeyEvent?): Boolean   
    {   
        //Si l'utilisateur clique sur le bouton back et   
        //qu'il est possible de retourner sur une page web précédente   
        //alors on retourne sur la page web précédente   
        if(keyCode == KeyEvent.KEYCODE\_BACK && ma\_webview.canGoBack())   
        {   
            ma\_webview.goBack()   
            return true   
        }   
        return super.onKeyDown(keyCode, event)   
    }   
}

Dans cet exemple, les parties relatives au chargement et à la configuration de la WebView sont en gras et commentées.

* Accès à des pages web locales

Présentation

Pour accéder à des pages web locales, il suffit d'avoir dans son application les fichiers HTML, CSS et JavaScript constituant la page web. Stocker les données dans le dossier Assets Folder convient bien à cet exercice. En effet, le lien entre les fichiers HTML, CSS et JavaScript se fait grâce à un chemin d'arborescence, chose non possible hors du dossier Assets.

Assets Folder

La différence entre le dossier res et le dossier Assets est que dans le répertoire res, chaque fichier reçoit un ID auquel il est possible d'accéder facilement via R.id… Ceci est utile pour accéder rapidement et facilement aux ressources via les fichiers d'IHM, ou via une classe Activity.

Le répertoire Assets est un système de fichiers et offre davantage de liberté pour y placer n'importe quel fichier. Il est possible d'accéder à chacun des fichiers de ce système, comme avec le système de fichiers Windows.

Usage

index.html

<!DOCTYPE html>   
<html>   
   
<head>   
   <title>Exemple</title>   
   <link rel="stylesheet" href="style.css">   
</head>   
   
<body>   
   
<div id="info">Je suis un exemple de page web</div>   
<p/>   
<div class="contenu">   
   <a href="hello.html">Lien vers une autre page</a>   
</div>   
</body>   
   
</html>

Page HTML simple qui sera affichée par la WebView.

Style.css

h1   
{   
   color:#FF0000;   
}   
#info   
{   
   background:pink;   
   width:50%;   
   margin-left:auto;   
   margin-right:auto;   
   text-align:center;   
}   
.contenu   
{   
   margin-left:auto;   
   margin-right:auto;   
   text-align:center;   
}

Style CSS de la page HTML précédemment définie.

hello.html

<!DOCTYPE html>   
<html>   
   
<head>   
   <title>Exemple</title>   
   <link rel="stylesheet" href="style.css">   
</head>   
   
<body>   
   
<div id="info">Inter has ruinarum varietates a Nisibi quam   
tuebatur accitus Vrsicinus</div>   
<p/>   
<div class="contenu">   
   <a href="hello.html" onclick="alert('coucou');">Fonction   
Javascript</a>   
</div>   
</body>   
   
</html>

Seconde page HTML simple, qui permettra de mettre en évidence le fonctionnement des liens et de se rendre compte qu'elle s'affiche toujours dans la WebView.

Activity\_main.xml

<?xml version="1.0" encoding="utf-8"?>   
<android.support.constraint.ConstraintLayout   
xmlns:android="http://schemas.android.com/apk/res/android"   
    xmlns:app="http://schemas.android.com/apk/res-auto"   
    xmlns:tools="http://schemas.android.com/tools"   
    android:layout\_width="match\_parent"   
    android:layout\_height="match\_parent"   
    tools:context="fr.acos.webviewweblocalwithkotlin.MainActivity">   
   
    <WebView   
        android:id="@+id/ma\_webview"   
        android:layout\_width="match\_parent"   
        android:layout\_height="match\_parent">   
    </WebView>   
   
</android.support.constraint.ConstraintLayout>

Dans cet exemple, l'IHM de l'activité contient uniquement une WebView aussi large et haute que l'écran du téléphone.

MainActivity.kt

package fr.acos.webviewweblocalwithkotlin   
   
import android.support.v7.app.AppCompatActivity   
import android.os.Bundle   
import android.view.KeyEvent   
import android.webkit.WebResourceRequest   
import android.webkit.WebView   
import android.webkit.WebViewClient   
import kotlinx.android.synthetic.main.activity\_main.\*   
import android.webkit.JsResult   
import android.webkit.WebChromeClient   
   
   
/\*\*   
 \* Classe de l'activité pour tester les WebView   
 \* avec des pages web locales.   
 \*/   
class MainActivity : AppCompatActivity() {   
   
    override fun onCreate(savedInstanceState: Bundle?) {   
        super.onCreate(savedInstanceState)   
        setContentView(R.layout.activity\_main)   
   
        //Chargement d'une page web distante.   
        ma\_webview.loadUrl("file:///android\_asset/index.html")   
   
        //Activation du JavaScript.   
        ma\_webview.settings.javaScriptEnabled = true   
   
        //Pour rester sur l'application lors d'un clic sur un lien.   
        ma\_webview.webViewClient = object : WebViewClient()   
        {   
            //Lors d'un clic sur un lien.   
            override fun shouldOverrideUrlLoading(view: WebView?,   
request: WebResourceRequest?): Boolean   
            {   
                //Le lien est chargé au sein de la WebView   
                view!!.loadUrl(request!!.url.toString())   
                return true;   
            }   
        }   
   
        //Pour afficher le contenu de la fonction (JavaScript) alert()   
        ma\_webview.webChromeClient = object : WebChromeClient()   
        {   
            override fun onJsAlert(view: WebView, url: String,    
message: String, result: JsResult): Boolean   
            {   
                return super.onJsAlert(view, url, message, result)   
            }   
        }   
    }   
   
    /\*\*   
     \* Modification du comportement du bouton BACK.   
     \* Pour ne pas quitter l'application mais   
     \* retourner sur la page web précédente.   
     \*/   
    override fun onKeyDown(keyCode: Int, event: KeyEvent?): Boolean   
    {   
        //Si l'utilisateur clique sur le bouton BACK et   
        //qu'il est possible de retourner sur une page web précédente   
        //alors on retourne sur la page web précédente   
        if(keyCode == KeyEvent.KEYCODE\_BACK && ma\_webview.canGoBack())   
        {   
            ma\_webview.goBack()   
            return true   
        }   
        return super.onKeyDown(keyCode, event)   
    }   
}

Dans cet exemple, les parties relatives au chargement et à la configuration de la WebView sont en gras et commentées.

* Appel de fonctions Kotlin avec du JavaScript
* Présentation

Pour accéder à des pages web locales, il suffit d'avoir dans son application les fichiers HTML, CSS et JavaScript constituant la page web. Il est alors possible de définir des fonctions JavaScript faisant appel à des fonctions Kotlin. C'est ce qui est présenté dans l'exemple ci-dessous.

* Usage

Cette démonstration affiche une page HTML contenant un lien. Lors d'un clic sur le lien, une fonction Kotlin est exécutée.

La démonstration est réalisée dans un nouveau projet dont l'activité principale se nomme MainActivity.

Activity\_main.xml

<?xml version="1.0" encoding="utf-8"?>   
<android.support.constraint.ConstraintLayout   
xmlns:android="http://schemas.android.com/apk/res/android"   
    xmlns:app="http://schemas.android.com/apk/res-auto"   
    xmlns:tools="http://schemas.android.com/tools"   
    android:layout\_width="match\_parent"   
    android:layout\_height="match\_parent"   
tools:context="fr.acos.fonctionjavascriptappellefonctionkotlin.  
MainActivity">   
   
    <WebView   
        android:id="@+id/ma\_webview"   
        android:layout\_width="match\_parent"   
        android:layout\_height="match\_parent">   
    </WebView>   
   
</android.support.constraint.ConstraintLayout>

Dans cet exemple, l'IHM de l'activité contient uniquement une WebView aussi large et haute que l'écran du téléphone.

MainActivity.kt

package fr.acos.fonctionjavascriptappellefonctionkotlin   
   
import android.support.v7.app.AppCompatActivity   
import android.os.Bundle   
import android.view.KeyEvent   
import android.webkit.\*   
import kotlinx.android.synthetic.main.activity\_main.\*   
import android.widget.Toast   
import android.webkit.JavascriptInterface   
import android.content.Context   
   
   
   
/\*\*   
 \* Classe de l'activité pour tester les WebView   
 \* avec des pages web locales.   
 \*/   
class MainActivity : AppCompatActivity() {   
   
    override fun onCreate(savedInstanceState: Bundle?) {   
        super.onCreate(savedInstanceState)   
        setContentView(R.layout.activity\_main)   
   
        //Chargement d'une page web distante.   
        ma\_webview.loadUrl("file:///android\_asset/hello.html")   
   
        //Activation du JavaScript.   
        ma\_webview.settings.javaScriptEnabled = true   
   
        //Lien entre la webView et la classe contenant les fonctions   
appelables par le JavaScript   
        ma\_webview.addJavascriptInterface(JavascriptPerso(this),"Android")   
   
        //Pour rester sur l'application lors d'un clic sur un lien.   
        ma\_webview.webViewClient = object : WebViewClient()   
        {   
            //Lors d'un clic sur un lien.   
            override fun shouldOverrideUrlLoading(view: WebView?, request:   
WebResourceRequest?): Boolean   
            {   
                //Le lien est chargé au sein de la WebView   
                view!!.loadUrl(request!!.url.toString())   
                return true;   
            }   
        }   
   
        //Pour afficher le contenu de la fonction (JavaScript) alert()   
        ma\_webview.webChromeClient = object : WebChromeClient()   
        {   
            override fun onJsAlert(view: WebView, url: String, message:   
String, result: JsResult): Boolean   
            {   
                return super.onJsAlert(view, url, message, result)   
            }   
        }   
    }   
   
    /\*\*   
     \* Modification du comportement du bouton BACK.   
     \* Pour ne pas quitter l'application mais   
     \* retourner sur la page web précédente.   
     \*/   
    override fun onKeyDown(keyCode: Int, event: KeyEvent?): Boolean   
    {   
        //Si l'utilisateur clique sur le bouton BACK et   
        //qu'il est possible de retourner sur une page web précédente   
        //alors on retourne sur la page web précédente   
        if(keyCode == KeyEvent.KEYCODE\_BACK && ma\_webview.canGoBack())   
        {   
            ma\_webview.goBack()   
            return true   
        }   
        return super.onKeyDown(keyCode, event)   
    }   
   
    /\*\*   
     \* Classe contenant des fonctions pouvant être appelées depuis le   
JavaScript   
     \*/   
    inner class JavascriptPerso(internal var context: Context)   
    {   
        /\*\*   
         \* Fonction avec l'annotation JavascriptInterface, ce qui indique   
que cette fonction est appelable depuis le JavaScript.   
         \*/   
        @JavascriptInterface   
        fun showToastPersonnalise(saisie: String)   
        {   
            Toast.makeText(context, saisie, Toast.LENGTH\_LONG).show()   
        }   
    }   
   
}

Dans cet exemple, les parties relatives au chargement et à la configuration de la WebView sont en gras et commentées.

hello.html

<!DOCTYPE html>   
<html>   
   
<head>   
    <title>Exemple</title>   
    <link rel="stylesheet" href="style.css">   
    <script>   
      function showAndroidToast(text)   
      {   
         Android.showToastPersonnalise(text);   
      }   
   
    </script>   
</head>   
   
<body>   
   
<div id="info">Inter has ruinarum varietates a Nisibi quam tuebatur   
accitus Vrsicinus</div>   
<p/>   
<div class="contenu">   
    <a href="hello.html" onclick="showAndroidToast('wololo');">  
Appel fonction Kotlin</a>   
</div>   
</body>   
   
</html>

Dans cet exemple, les parties relatives au lien entre JavaScript et Kotlin sont en gras et commentées.

Style.css

h1   
{   
   color:#FF0000;   
}   
#info   
{   
   background:pink;   
   width:50%;   
   margin-left:auto;   
   margin-right:auto;   
   text-align:center;   
}   
.contenu   
{   
   margin-left:auto;   
   margin-right:auto;   
   text-align:center;   
}

Style CSS de la page HTML précédemment définie.

Lors d'un clic sur le lien HTML "Appel fonction Kotlin", un toast affichant "wololo" apparaît.

* Les récepteurs d’événement
* Présentation

Un Broadcast Receiver (récepteur d'événement) est un composant qui s'abonne à un ou plusieurs événements. Le système achemine automatiquement les événements vers les Broadcast Receiver qui se sont abonnées à ce type d'événement. Quand un Broadcast Receiver capte un événement, il exécute un traitement sur le thread principal.

Les événements sont envoyés par le système d'exploitation Android ou par les applications.

Un événement est représenté par un objet de type Intent.

Ce chapitre présente :

* la création d'un Broadcast Receiver,
* l'abonnement d'un Broadcast Receiver à des événements,
* la diffusion d'événements.
* Création d'un Broadcast Receiver

Pour créer un BroadCast Receiver, il suffit de faire un clic droit sur un package. Puis choisir : New > Other > Broadcast Receiver, puis définir un nom de classe.

Ci-dessous, la classe générée par Android Studio avec des commentaires :

Classe MyReceiver

package fr.acos.broadcastreceiverwithkotlin.broadcastreceivers   
   
import android.content.BroadcastReceiver   
import android.content.Context   
import android.content.Intent   
   
/\*\*   
 \* Classe représentant le Broadcast Receiver   
 \*/   
class MyReceiver : BroadcastReceiver() {   
   
    /\*\*   
     \* Fonction exécutée lors de la réception d'un événement.   
     \* @param context Contexte de l'application.   
     \* @param Intent représentant l'événement receptionné.   
     \*/   
    override fun onReceive(context: Context, intent: Intent)   
    {   
        throw UnsupportedOperationException("Not yet implemented")   
    }   
}

La fonction onReceive() contient le traitement du Broadcast Receiver.

* Abonnement à des événements

Pour abonner un Broadcast Receiver à un événement, il est nécessaire de lui définir un intent-filter dynamiquement à l'aide de la fonction registerReceiver().

La fonction registerReceiver() attend en premier paramètre une instance du Broadcast Receiver et en second paramètre un objet représentant un Intent-filter.

Exemple d'abonnement dynamique

//Objet représentant un intent-filter sur l'événement    
"fr.acos.event" val intentFilter = IntentFilter("fr.acos.event")   
//Instance du Broascast Receiver à qui on veut ajouter un événement. val mReceiver = MyReceiver()   
   
//Abonnement du Broadcast Receiver à l'événement    
"fr.acos.event" this.registerReceiver(mReceiver, intentFilter)

Pour désabonner un Broadcast Receiver d'un événement, il est nécessaire d'utiliser la fonction unregisterReceiver().

* Diffusion d'un événement

La fonction sendBroadcast() permet de diffuser un événement. Elle attend en paramètre un objet de type intent qui représente l'événement à diffuser.

Exemple de diffusion d'un événement

//Objet représentant un événement sous forme d'Intent   
val intent = Intent("fr.acos.event")   
//Diffusion de l'événement   
sendBroadcast(intent)

* Démonstration

Dans cette démonstration, il n'y a pas d'IHM, un intent est envoyé à un broadcast Receiver au moment de la création de l'activité MainActivity. Le broadcast Receiver affiche un log de la réception de l’intent.

La démonstration est réalisée dans un nouveau projet dont l'activité principale se nomme MainActivity et est de type "Empty Activity".

MyReceiver.kt

package fr.acos.broadcastreceiverwithkotlin.broadcastreceivers   
   
import android.content.BroadcastReceiver   
import android.content.Context   
import android.content.Intent   
import org.jetbrains.anko.AnkoLogger   
import org.jetbrains.anko.info   
   
/\*\*   
 \* Classe représentant le BroadCast Receiver   
 \*/   
class MyReceiver : BroadcastReceiver(), AnkoLogger   
{   
    /\*\*   
     \* Fonction exécutée lors de la réception d'un événement.   
     \* @param context Contexte de l'application.   
     \* @param Intent représentant l'événement réceptionné.   
     \*/   
    override fun onReceive(context: Context, intent: Intent)   
    {   
        info("Réception de l'événement ${intent.action}")   
    }   
}

Lors de l'exécution du Broadcast Receiver, celui-ci émet un log d'info.

AndroidManifest.xml

<?xml version="1.0" encoding="utf-8"?>   
<manifest xmlns:android="http://schemas.android.com/apk/res/android"   
    package="fr.acos.broadcastreceiverwithkotlin">   
   
    <application   
        android:allowBackup="true"   
        android:icon="@mipmap/ic\_launcher"   
        android:label="@string/app\_name"   
        android:roundIcon="@mipmap/ic\_launcher\_round"   
        android:supportsRtl="true"   
        android:theme="@style/AppTheme">   
        <activity android:name=".MainActivity">   
            <intent-filter>   
                <action android:name="android.intent.action.MAIN" />   
                <category android:name="android.intent.category.LAUNCHER" />   
            </intent-filter>   
        </activity>   
   
        <receiver   
            android:name=".broadcastreceivers.MyReceiver"   
            android:enabled="true"   
            android:exported="true">   
        </receiver>   
    </application>   
   
</manifest>

Le Broadcast Receiver doit être déclaré dans le manifeste, cela se fait automatiquement lorsque le Broadcast Receiver est créé via l'assistant d'Android Studio.

MainActivity.kt

package fr.acos.broadcastreceiverwithkotlin   
   
import android.content.Intent   
import android.support.v7.app.AppCompatActivity   
import android.os.Bundle   
import android.content.IntentFilter   
import   
fr.acos.broadcastreceiverwithkotlin.broadcastreceivers.MyReceiver   
   
   
class MainActivity : AppCompatActivity() {   
   
    override fun onCreate(savedInstanceState: Bundle?) {   
        super.onCreate(savedInstanceState)   
        setContentView(R.layout.activity\_main)   
   
        //Objet représentant un intent filter sur l'événement   
"fr.acos.event"   
        val intentFilter = IntentFilter("fr.acos.event")   
        //Instance du Broascast Receiver à qui on veut ajouter un   
événement.   
        val mReceiver = MyReceiver()   
   
        //Abonnement du Broadcast Receiver à l'événement   
"fr.acos.event"   
        this.registerReceiver(mReceiver, intentFilter)   
    }   
   
    override fun onResume() {   
        super.onResume()   
   
        //Objet représentant un événement sous forme d'Intent   
        val intent = Intent("fr.acos.event")   
        //Diffusion de l'événement   
        sendBroadcast(intent)   
    }   
}

Dans cet exemple, les instructions relatives au broadcastReceiver sont en gras et commentées.