AIDL

В буквальном переводе – язык описания интерфейсов Android. Используется для описания композиции и декомпозиции Java объектов в примитивы ОС для непосредственно передачи между процесами.  
AIDL файлы очень похожи на стандартные интерфейсы в java за исключением:

* Импортировать нужно даже те aidl файлы, которые находятся в том же пакете.
* Ключевое слово oneway в декларации void метода означает что метод будет вызван асинхронно (клиент не дожидается его выполнения).
* Использовать можно только примитивы, String, List и Parcelable классы, объявленные в других aidl файлах.

С помощью AIDL автоматически генерируется java код для генерации stab’ов.

Архитектура приложения

Разработанное нами приложение — галерея для Android которая позволяет просматривать фотографии из карты памяти и сетей обмена фотографиями.  
Основными задачами сервиса в данном приложении являются: получение метаданных (информации о альбомах, фотографиях, друзьях), мониторинг их обновлений и всего остального, что с ними связано. Сервис постоянно хранит актуальную информацию и готов в любой момент отдать ее основной Activity для отображения.  
Ниже будут приведены ключевые участки кода и описан процесс создания примитивного сервиса:  
Для осуществления общения между сервисом и Activity используются следующие AIDL файлы:  
  
IDataSourceService.aidl – интерфейс сервиса:

packagecom.umobisoft.habr.aidlexample.common;

import com.umobisoft.habr.aidlexample.common.IDataSourceServiceListener;

interfaceIDataSourceService{

voidloadAlbums(in IDataSourceServiceListener listener);

…

}

IDataSourceServiceListener.aidl – интерфейс слушателей сообщений от сервиса:

package com.umobisoft.habr.aidlexample.common;

import com.umobisoft.habr.aidlexample.common.pojo.Album;

interface IDataSourceServiceListener{

oneway void albumItemLoaded(in Album a);

}

Данные передаются с помощью двух классов, которые реализуют интерфейс Parcelable — Album и Photo. Декларация aidl файлов для этих классов обязательна. При конвертации из примитивов ОС в java Объекты используется класс Creator.  
Для записи данных используется метод writeToParcel интерфейса Parcelable:

@Override

public void writeToParcel(Parcel out, int flags) {

try{

out.writeLong(id);

out.writeString(name);

out.writeTypedList(photos);

}catch (Exception e) {

Log.e(TAG, "writeToParcel", e);

}

}

Также существует вспомогательный метод describeContents, его задача описать специальные случаи/состояния объектакоторые когут использоватся при сериализации и десириализации:

@Override

public int describeContents() {

*// TODO Auto-generated method stub*

return 0;

}

Методы чтения данных оказались недостойными вынесения их в состав интерфейса Parcelable, но стандартная практика – использование Creator вместе с:

private void readFromParcel(Parcel in) {

try{

id = in.readLong();

name = in.readString();

photos.clear();

in.readTypedList(photos, Photo.CREATOR);

}catch (Exception e) {

Log.e(TAG, "readFromParcel", e);

}

}

Activity запускает сервис (делая его таким образом StartedService) в методе onCreate

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.main);

textView = (TextView)findViewById(R.id.album\_text);

Intent serviceIntent = newIntent(this, DataSourceService.class);

startService(serviceIntent);

connectToService();

}

На этом же этапе жизненного цикла она соединяется с сервисом:

private void connectToService() {

Intent intent = newIntent(this, DataSourceService.class);

this.bindService(intent, serviceConnection, Context.BIND\_AUTO\_CREATE);

}

Процесс соединения с сервисом асинхронный, в нем учувствует реализация интерфейса ServiceConnection. Во время соединения с сервером Activity регистрируется в сервисе как слушатель сообщений, с помощью имплементации IDataSourceServiceListener.Stub:

private ServiceConnection serviceConnection = newServiceConnection() {

@Override

public void onServiceConnected(ComponentName name, IBinder service) {

Log.i(TAG, "Service connection established");

serviceApi = IDataSourceService.Stub.asInterface(service);

try {

mainListener = newIDataSourceServiceListener.Stub() {

@Override

publicvoidalbumItemLoaded(final Album a) throwsRemoteException {

mToastHandler.post(new Thread(){

publicvoid run(){

Toast.makeText(HabrahabrAIDLExampleActivity.this, a.toString(), Toast.LENGTH\_LONG).show();

textView.setText(a.toString());

}

});

}

};

serviceApi.loadAlbums(mainListener);

} catch (RemoteException e) {

Log.e(TAG, "loadAlbums", e);

}

}

@Override

publicvoidonServiceDisconnected(ComponentName name) {

Log.i(TAG, "Service connection closed");

serviceApi = null;

connectToService();

}

};