# Binder

Binder je mehanizam za međuprocesnu komunikaciju Android operativnog sistema (IPC -eng. Inter Process Communication). Implementacija Binder programske sprege bazirana je na razmeni poruka, organizovana u vidu klijent - server modela. Pomenuta organizacija Binder programske sprege pruža mogućnost komunikacije procesa operativnog sistema realizovanih na različitim nivoima apstrakcije, različitim programskim jezicima.
Binder omogucava serijalizaciju i deserijalizaciju standardnih tipova(Marshalling ili Unmarshalling - "prosledjivanje objekata"). Serijalizacija podataka (proces pretvaranja struktura „visokog" nivoa u parcele) sa ciljem ugrađivanja u Binder transakciju, deserijalizacija podrazuvema obrnut proces.

Komunikacija se zasniva na dobavljanju instance programske klase kojom je definisana programska sprege (Binder objekat) i pozivanju odgovarajućih metoda. AOSP definiše Service menadžer kao centralnu tačku sistema, odnosno servis koristi Service menadžer da registruje svoj Binder objekat, odakle će ga klijentski procesi preuzeti i koristiti za komunikaciju sa servisom.

```
public class LocalService extends Service {
    // Binder given to clients
    private final IBinder mBinder = new LocalBinder();
    // Random number generator
    private final Random mGenerator = new Random();

    /**
     * Class used for the client Binder.  Because we know this service always
     * runs in the same process as its clients, we don't need to deal with IPC
     */
    public class LocalBinder extends Binder {
        LocalService getService() {
            // Return this instance of LocalService so clients can call public
            return LocalService.this;
        }
    }

    @Override
    public IBinder onBind(Intent intent) {
        return mBinder;
    }
```

Na nivou Java programskog jezika Binder komunikacija se izvršava, tako što postoji alat za generisanje celokupne komunikacija, a na korisniku je samo da upotrebi dobijeno rešenje. Definisan je poseban programski jezik (AIDL - Androd Interface Definition Language) kojim se definiše programska sprega servisa. Datoteke koje sadrže AIDL programski kod imaju ekstenziju .aidl. Sledeći primer ilustruje

jedan AIDL primer opisa programske sprege  servisa.

```
// IBinderExample.aidl

// Declare any non-default types here with import statements

interface IBinderExample {

    int getValue();

    void setValue(int value);


    void printMessage();
```

AIDL interfejs se kreira u okviru project/app foldera. Desni klik na app folder file/AIDL/aidl file.

U aidl interfejsu treba da se nalaze metode koje ce da predstavljaju funkcionalnosti IBinder klase.

Nakon kreiranja ovog interfejsa potrebno je uraditi clean projekta I proveriti da li se u projektu na putanji app/build/generated/source/aidl/debug/ime_paketa/ nalazi izgenerisan source file aidl interfejsa koji je kreiran u prethodnom koraku.

Upravo u tom generisanom fajlu treba da se nalazi apstraktna klasa Stub koja ce se koristiti u narednom koraku.

Nakon uspesnog kreiranja aidl interfejsa, klasa koja predstavlja Binder treba da nasledi apstraktnu klasu Stub iz naseg generisanog fajla:

```
public class BinderExample extends IBinderExample.Stub
```

U tom trenutku klasa BinderExample morace da pozove (override) metode definisane u IBinderExample.aild interfejsu.

Kada se implementacija IBinderExample klase zavrsi potrebno je kreirati servis koji ce se u onBind() metodi povezati sa Binder-om koji smo upravo kreirali:

```
public class BindService extends Service {

    private BinderExample binder = null;

    public MyService() {

    }

    @Override

    public IBinder onBind(Intent intent) {

        if (binder == null){

            binder = new BinderExample();

        }

        return binder;

    }

    @Override

    public boolean onUnbind(Intent intent) {

        binder.stop();

        return super.onUnbind(intent);

    }}
```

U aktivitiju u kom hocemo da iskoristimo funkcionalnosti Binder-a potrebno je da implementiramo interfejs ServiceConnection koji ima metode onServiceConnected() I onServiceDisconnected():

```
@Override
public void onServiceConnected(ComponentName name, IBinder service) {
    mService = IBinderExample.Stub.asInterface(service);
    try {
         mService.printMessage();
           } catch (RemoteException e) {
           e.printStackTrace();
           }
        }
@Override
 public void onServiceDisconnected(ComponentName name) {
    Log.d(TAG, "onServiceDisconnected");
    mService = null;
    }
```