Instrukcja implementacji sterownika wirtualnego portu szeregowego dla systemu Android.

Opracowanie:
Elzab Soft sp. z o.o.

29.06.2015

Spis treści

1. Wymagania

Aby urządzenie mogło obsługiwać połączenie USB jako host, musi ono spełniać następujące warunki:

- obecność systemu Android w wersji 3.1 lub wyższej
- wsparcie dla specyfikacji USB On-The-Go (USB OTG)

2. Uprawnienia systemowe

Następujące uprawnienia w pliku AndroidManifest.xml wymagane są do prawidłowej pracy sterownika *usb-serial-for-android*:

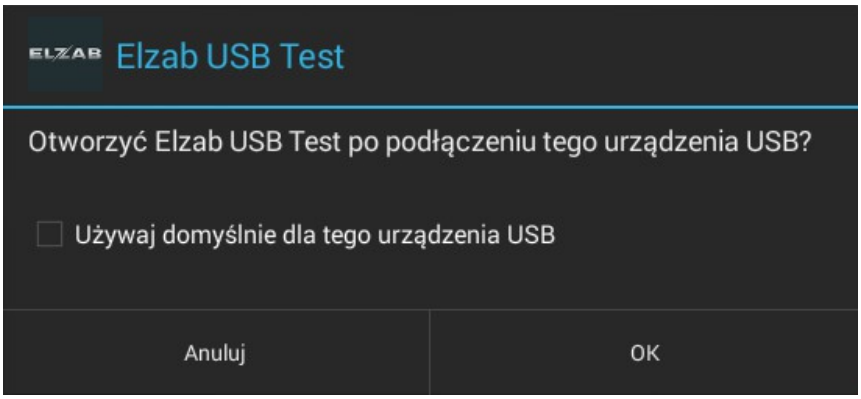
- *android.permission.WRITE_EXTERNAL_STORAGE*
- *android.permission.READ_EXTERNAL_STORAGE*
- *android.permission.READ_PHONE_STATE*

3. Uprawnienie do komunikacji z urządzeniem USB

Aby aplikacja mogła porozumiewać się z urządzeniem USB jako host, musi ona posiadać odpowiednie uprawnienie. Nie jest ono jednak przyznawane odgórnie na podstawie uprawnień systemowych określonych w pliku *AndroidManifest.xml*. Musi ono zostać nadane przez użytkownika dla poszczególnej aplikacji, przy każdorazowym podłączeniu urządzenia USB. Poniżej wymienione zostały dwa sposoby pozyskania uprawnień przez aplikację.

3.1 Uzyskiwanie uprawnienia w momencie podłączenia urządzenia

Uprawnienie nadawane jest przez użytkownika w oknie dialogowym, które pojawi się tylko i wyłącznie w momencie podłączenia urządzenia USB.
Okno pojawi się jedynie dla aplikacji, która posiada odpowiedni wpis w pliku *AndroidManifest.xml*. Wpis powoduje, że aplikacja reagować będzie na zdarzenie systemowe (akcję) `"android.hardware.usb.action.USB_DEVICE_ATTACHED"`. Zatwierdzenie okna dialogowego spowoduje uruchomienie wybranej aktywności oraz nadanie aplikacji uprawnienia do komunikacji z urządzeniem.



Poniższy wpis należy umieścić w ramach aktywności, która ma zostać uruchomiona po zatwierdzeniu okna dialogowego.

```
<intent-filter>
    <action android:name="android.hardware.usb.action.USB_DEVICE_ATTACHED" />
</intent-filter>
<meta-data
    android:name="android.hardware.usb.action.USB_DEVICE_ATTACHED"
    android:resource="@xml/device_filter" />
```

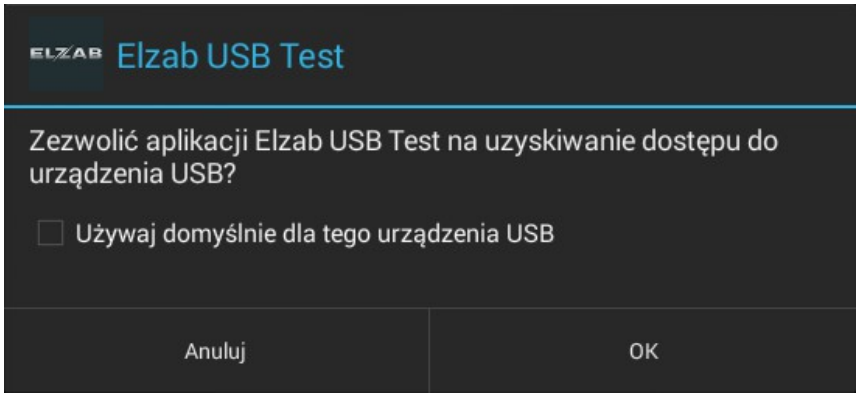
Zawartość wyżej wymienionego pliku *device_filter.xml* musi zawierać spis urządzeń po podłączeniu których uruchomi się okno dialogowe.

Przykładowa zawartość pliku *device_filter.xml* dla urządzeń Elzabu:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <usb-device vendor-id="1155" />
</resources>
```

3.2 Uzyskiwanie uprawnienia w czasie działania aplikacji

Opisana tutaj metoda w porównaniu do wcześniejszej, nie wymaga każdorazowego podłączania urządzenia w celu uzyskania uprawnień do wymiany danych. Pozwoli na uzyskanie uprawnień w dowolnym momencie działania programu, gdy urządzenie zostało wcześniej podłączone. Wymaga za to modyfikacji kodu źródłowego aplikacji. Podobnie jak w pierwszej metodzie, nadanie uprawnienia odbywa się przy pomocy okna dialogowego.



W pierwszym kroku należy zdefiniować metodę wywoływaną w momencie potrzeby uzyskania uprawnień. Spowoduje ona wyświetlenie użytkownikowi okna dialogowego z zapytaniem o udzielenie aplikacji uprawnienia do komunikacji z urządzeniem USB.

```
public static void RequestForPermission(Context context, UsbDevice usbDevice) {
    UsbManager usbManager = (UsbManager) context.getSystemService(Context.USB_SERVICE);
    if( !usbManager.hasPermission(usbDevice) )
        usbManager.requestPermission(usbDevice, getPermissionIntent(context));
}
```

Widoczna tu metoda *requestPermission* bezpośrednio odpowiada za wyświetlenie okna dialogowego. Wymaga podania obiektu klasy *PendingIntent*, który określa akcję (zdarzenie) zgłaszaną aplikacji dopiero po udzieleniu przez użytkownika odpowiedzi w oknie dialogowym. Należy zdefiniować metodę, która rejestruje w aplikacji obiekt *BroadcastReceiver* nasłuchujący akcje (zdarzenia) z systemu operacyjnego oraz z aplikacji. Akcję, która aktywna będzie w momencie zamknięcia okna można nazwać dowolnie – w poniższym przykładzie będzie to `"pl.elzab.virtualcomsample.USB_PERMISSION"`.

Ponieważ obiekt *BroadcastReceiver* rejestrowany w programie jest jednokrotnie, poniższa metoda jest singletonem, zwraca ona obiekt statyczny klasy *PendingIntent*.

```
public static synchronized PendingIntent getPermissionIntent(Context context) {
    if( sPermissionIntent != null)
        return sPermissionIntent;
    return RegisterBroadcastReceiver(context);
}
```

Metoda odpowiedzialna za rejestrację obiektu *BroadcastReceiver,* wygląda następująco:

```
public static PendingIntent RegisterBroadcastReceiver(Context context) {
    sMyBroadcastReceiver = new MyBroadcastReceiver();
    sPermissionIntent = PendingIntent.getBroadcast(
context, 0, new Intent(sMyBroadcastReceiver.ACTION_USB_PERMISSION), 0);
    IntentFilter filter = new IntentFilter(sMyBroadcastReceiver.ACTION_USB_PERMISSION);
    context.registerReceiver(sMyBroadcastReceiver, filter);
    return sPermissionIntent;
}
```

Obiekt statyczny *sMyBroadcastReceiver* zawiera w sobie metodę *onReceive*, która wywoływana jest w momencie, gdy użytkownik udzieli odpowiedzi na zapytanie wyświetlone w oknie dialogowym. Mówiąc inaczej, metoda ta zareaguje na akcję `"pl.elzab.virtualcomsample.USB_PERMISSION"`. W przypadku, gdy wynik odpowiedzi jest pozytywny, można dla przykładu wykonać połączenie z urządzeniem USB, do komunikacji z którym aplikacja otrzymała uprawnienie. Przykładowa klasa *MyBroadcastReceiver* dziedziczy po klasie *BroadcastReceiver* i może wyglądać następująco:

```
public class MyBroadcastReceiver extends BroadcastReceiver {
    public final String ACTION_USB_PERMISSION =
"pl.elzab.virtualcomsample.USB_PERMISSION";
    public final String TAG = "ElzabUSB";

    @Override
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();

 mUsbManager.requestPermission(device, mPermissionIntent)
        if (ACTION_USB_PERMISSION.equals(action)) {
            synchronized (this) {
                UsbDevice device =
(UsbDevice)intent.getParcelableExtra(UsbManager.EXTRA_DEVICE);
                if (intent.getBooleanExtra(UsbManager.EXTRA_PERMISSION_GRANTED, false)) {
                    if(device != null) {
                        Log.d(TAG, "Permission granted for device " + device);
                        USBConnection.getConnection(context, device);
                    }
                }
                else {
                    Log.d(TAG, "Permission denied for device " + device);
                    USBConnection.closeConnection(context);
                }
            }
        }
    }
}
```

4. Uzyskiwanie połączenia z wirtualnym portem szeregowym

Poniżej przedstawiono kroki implementacji sterownika w kodzie źródłowym aplikacji, które wymagane są do uzyskania połączenia z wirtualnym portem szeregowym urządzenia USB.

4.1. uzyskanie usługi USB systemu Android

```
UsbManager usbManager = (UsbManager) getSystemService(Context.USB_SERVICE);
```

4.2. uzyskanie listy dostępnych sterowników wirtualnego portu szeregowego

```
final List<UsbSerialDriver> drivers =
UsbSerialProber.getDefaultProber().findAllDrivers(usbManager);
```

W przypadku, gdy obiekt *UsbDevice* jest znany, w celu znalezienia pojedynczego sterownika dla tego urządzenia, można użyć metody *probeDevice*.

```
UsbSerialDriver driver = UsbSerialProber.getDefaultProber().probeDevice(usbDevice);
```

4.3. uzyskanie listy dostępnych wirtualnych portów szeregowych

```
final List<UsbSerialPort> result = new ArrayList<UsbSerialPort>();
for (final UsbSerialDriver driver : drivers) {
    final List<UsbSerialPort> ports = driver.getPorts();
    result.addAll(ports);
}
```

4.4. otwarcie połączenia z jednym z urządzeń USB, które sklasyfikowano jako wirtualny port szeregowy

```
UsbSerialPort serialPort = result.get(0);
UsbDeviceConnection connection =
usbManager.openDevice(serialPort.getDriver().getDevice());
serialPort.open(connection);
```

Przed wywołaniem metody *openDevice*, należy sprawdzić, czy aplikacja uzyskała uprawnienie do komunikacji z danym urządzeniem USB. Brak uprawnień uniemożliwi wymianę danych.

Jeżeli poniższa funkcja zwróci wartość *true*, oznacza to, że uprawnienie zostało nadane.

```
usbManager.hasPermission(serialPort.getDriver().getDevice())
```

Informacje dotyczące uprawnienia do komunikacji z urządzeniem opisane zostały w sekcji trzeciej.

4.5. określenie parametrów komunikacji portu szeregowego

Od strony urządzenia z systemem Android, należy ustawić następujące parametry przesyłu danych:

- szybkość transmisji: 115200 baud
- liczba bitów danych: 8
- liczba bitów stopu: 1
- ignorowanie bitu parzystości

```
serialPort.setParameters(115200, 8, UsbSerialPort.STOPBITS_1, UsbSerialPort.PARITY_NONE);
```

4.6. zamknięcie połączenia

```
serialPort.close();
```

5. Wysyłanie i odbiór danych

Wysyłanie danych możliwe jest poprzez metodę *write* obiektu *UsbSerialPort*.
Jako parametry funkcji, należy podać tablicę bajtów oraz *timeout* wyrażany w milisekundach.
Przykład:

```
serialPort.write(new byte[] {0x05, 0x06, 0x07}, 1000);
```

W celu umożliwienia odbioru danych, można wykorzystać klasę *SerialInputOutputManager* oraz *SerialInputOutputManager.Listener.* Dla klasy *SerialInputOutputManager.Listener* należy nadpisać metodę *onNewData*, oraz *onRunError.* Pierwsza metoda wywołuje się za każdym razem, gdy do portu napłyną dane z zewnątrz. Dane dostępne są jako tablica bajtów występująca w parametrze metody. Druga metoda wywoływana jest w przypadku wystąpienia błędu wątku nasłuchującego port.

W poniższym przykładzie, jako reakcja na przypływ danych, wywoływana jest funkcja *updateReceivedData* aktywności *Prima2TestActivity.*

```
SerialInputOutputManager.Listener listener = new SerialInputOutputManager.Listener() {
	@Override
	public void onNewData(final byte[] data) {
		Prima2TestActivity.this.runOnUiThread(new Runnable() {
			@Override
			public void run() {
				Prima2TestActivity.this.updateReceivedData(data);
			}
		});
	}

	@Override
	public void onRunError(Exception e) {
		Log.d(TAG, e.getMessage());
	}
};
```

W następnym kroku należy powiązać obiekt otwartego wcześniej połączenia szeregowego z obiektem reagującym na przypływ danych.

```
SerialInputOutputManager serialIOManager =
new SerialInputOutputManager(serialPort,  listener);
```

Zdefiniowane zadanie pracujące w tle, można uruchomić w ramach obiektu *ExecutorService*.

```
ExecutorService executor = Executors.newSingleThreadExecutor();
executor.submit(serialIOManager);
```

Po zakończeniu połączenia z wirtualnym portem szeregowym, należy zakończyć pracę obiektu *SerialInputOutputManager* za pomocą poniższej instrukcji.

```
mSerialIOManager.stop();
```