Så talar Android med sina tillbehör

EXPERTARTIKEL

Vill du koppla ditt inbyggnadskort till en Android? Så här fungerar program-gränssnitten.

Av David Flowers, Microchip Technology

David Flowers är Principal Applications Engineer på Microchips avdelning för tillämpningar. Han utvecklar och ger support på mjukvara, bland annat för Microchips produkter för Androidtillbehör. Första tiden på Microchip – han anställdes 2004 – jobbade han med företagets åttabitars PIC18-styrkretsar. David Flowers har en magisterexamen i elektroteknik och en kandidatexamen i datavetenskap, båda från Georgia Institute of Technology. Magisteruppsatsen gjorde han som praktikant på Motorola Energy Systems Group.

De viktigaste anslutningarna för tillbehör till Android är USB, Bluetooth och Wifi. Exakt hur anlslutningarna används beror på vilken version av operativsystemet man arbetar med och vilket hårdvarustöd enheten har.

Wifi är förmodligen ett av de enklaste och mest väldokumenterade gränssnitt som finns vad gäller apputveckling generellt. Om du utvecklar ett tillbehör som har sin egen http-server behöver du inte ens ta fram en skräddarsydd tillämpning. I stället kan du använda telefonens eller surfplattans egen webbläsare som gränssnitt.

Utöver det kan du använda diverse telnet- och ftp-tillämpningar för att slippa skräddarsy egen kod. Och om det trots allt krävs skräddarsydd kod finns nätverks-API:er i Java och mängder av dokumentation som visar hur de används.

Det finns bara en liten detalj som är specifik för Androids OS innan en app får använda nätverks-API:er, och det är en kodrad som måste läggas till i filen *AndroidManifest.xml* – se figur 1.

I DAGSLÄGET SAKNAR ANDROID stöd för ad-hoc-nätverk. Detta är en stor begränsning när det gäller att använda Wifi som gränssnitt till ett tillbehör, eftersom det krävs en nätverksinfrastruktur för att Wifi-tillbehöret ska fungera. Det fungerar bra för vissa tillämpningar – exempelvis en termostat i ett hus som alltid har Wifi-uppkoppling till hemmaroutern – men är orealistiskt för nästan alla andra mobila tillbehör.

Vad gäller Bluetooth har olika versioner av Android stöd för olika Bluetoothenheter.

Android 2.x stöder serieportprofilen SPP, även om alla Android 2.x-tillbehör inte kan använda den. SPP skapar skräddarsydda tillämpningar som inte har ett fördefinierat dataformat.

Android 3.x introducerade stöd för headset och A2DP (Advanced Audio Distribution Profile), en profil för att spela upp ljudfiler.

Android 4.x kompletterades bland annat med HDP (Health Device Profile), en profil för hälsotillämpningar.

VAD GÄLLER USB så är det ett av de senaste möjligheter som erbjuds för att ladda över data från Android. Tidigare fick bara de som tillverkade telefoner och surfplattor använda USB-porten men detta förändrades med uppdateringarna till Android 2.3.4 och 3.1.

Med Android 3.1 kom även ett programgränssnitt (API) för USB Host, som gör det möjligt för utvecklare att ansluta USB-enheter av standardtyp till Android.

Operativsystemet har inbyggt stöd för HID (Human Interface Device), masslagring (Mass Storage Devices, MSD) och andra klasser av USB-enheter. Inbyggda drivrutiner gör att sådan USB-utrustning kan användas sömlöst, precis som i en vanlig dator.

För kringutrustning utan inbyggt stöd kan USB Host-API:et användas för att direkt ansluta till USB-ändpunkterna via ett lågnivå-API – se figur 2.

En app får tillgång till USB Host-API:et genom att göra en liten notering i filen *AndroidManifest.xml* – se figur 3.

Genom att sätta upp ett filter kan utvecklaren bestämma att en viss tillämpning ska startas automatiskt när en viss enhet kopplas till USB-porten. Det kallas intent-filter, skapas i *AndroidManifest.xml*-filen och knyts till händelsen USB_DEVICE_ATTACHED, som i sin tur måste knytas till en filterfil, exempelvis *xml/device_filter.xml* – se figur 4.

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

Figur 1: Tillåt en app att ansluta till internet.

```
UsbManager = (UsbManager)getSystemService(Context.USB _ SERVICE);
UsbInterface intf = device.getInterface(0);
UsbDeviceConnection connection = manager.openDevice(device);
connection.claimInterface(intf, true);
UsbEndpoint endpointOUT = intf.getEndpoint(0);
connection.bulkTransfer(endpointOUT, buffer, buffer.length, timeout _ ms);
```

Figur 2: Anslut via USB Host-API :et och skicka iväg ett paket.

```
<uses-library android:name="android.hardware.usb.host" />
```

Figur 3: Aktivera en app genom att ge den åtkomst till USB Host-API:et.

```
<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" />
```

Figur 4: Starta en app automatiskt när en enhet ansluts i USB-värdläge.

Ljuddocka, träningsutrustning, medicinskt instrument eller väderstation – allt kan kopplas till en Android via Bluetooth, Wifi eller USB.

Filen *device_filter.xml* ger information om vilka enheter som ska starta appen. Det kan ske antingen genom företags- och produktidentifiering (Vendor ID, VID och Product ID, PID) eller via klass, subklass och protokolluppsättning – se figur 5.

App-utvecklare behöver inte vara specifika utan kan låta bli att tagga vissa attribut. Om det exempelvis saknas ett attribut för produktidentifiering, så kan vilken utrustning som helst med matchande företagsidentifiering få appen att starta.

GOOGLE HAR ADDERAT ett eget gränssnitt kallat Open Accessory till standard-USB-drivrutinerna. Det gör att även Androidenheter som saknar hårdvarustöd för USB Host kan få USB-anslutning. Därmed kan tillbehörsutvecklare utnyttja en vanlig USB-port för specialsydd datatrafik.

Open Accessory-protokollet inleder med att skicka några företagsspecifika kommandon till USB-porten i enlighet med vad enhetens tillverkare bestämt. Dessa kommandon försätter USB-drivrutinerna i tillbehörsläge – USB-enheten kopplas bort från bussen och kopplas på igen i tillbehörsläge med Googles företags-ID och ett av två specifika produkt-ID:n. Därmed öppnas ett företagsspecifikt gränssnitt för appen.

Gränssnittet liknar en FileStream. Man skriver och läser data på samma sätt som man läser eller skriver en fil. Detta förfarande skiljer sig från hur USB-firmware brukar fungera – gränssnittet brukar baseras på USB-protokollets egen paketstorlek. Skillnaden medför vissa problem som utvecklare av appar och tillbehörsfirmware bör vara uppmärksamma på.

Androidenhetens USB-drivrutin tar emot en ren dataström och kan därför inte identifiera och tolka eventuella logiska gränser mellan data för olika kommandon – data från två separata anrop till appens skrivfunktion kan mycket väl hamna i ett och samma USB-paket. Så firmware måste ha beredskap för att ett mottaget USB-paket kan innehålla information från två separata anrop.

ETT ANROP FRÅN APPEN till skrivfunktionen kan dessutom delas upp i flera USB-paket. Androidenhetens USB-drivrutin delar upp ett anrop i små paket och sänder dem till tillbehöret, som sedan måste kunna sätta ihop anropet igen till rätt format.

Som om inte detta var nog trassligt, kan paketering och fragmentering dessutom förekomma samtidigt.

Här följer ett exempel. Open Accessory-ramverket använder för närvarande paket på 64 byte. Om appen anropar skrivfunktionen två gånger i följd, och det första anropet sänder 20 byte och det andra 64 byte, så kanske detta data packas till ett block på 84 byte – det beror på när USB-drivrutinen tar datat från strömmen och sänder det över bussen.

USB-drivrutinen delar i sin tur upp denna dataström i paket av USB-storlek. Den sänder först de första 64 byten i ett paket, och därefter ett paket på 20 byte. Det betyder att det första paketet innehåller 20 byte data från det första skrivanropet och 44 byte från det andra, medan det andra paketet på 20 byte är resterande data från det andra skrivanropet. ▶

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <usb-device vendor-id="1240" product-id="516"/>
   <usb-device class="256" subclass="256" protocol="5"/>
</resources>
```

Figur 5: Upprätta ett filter för att en utrustning ska starta en app i USB-värdläge.

```
<uses-library android:name="com.android.usb.accessory" />
```

Figur 6: Ge en app att tillgång till Open Accessory-ramverket.

```
<intent-filter>
 <action android:name="android.hardware.usb.action.USB _ ACCESSORY _ ATTACHED" />
</intent-filter>
<meta-data android:name="android.hardware.usb.action.USB _ ACCESSORY _ ATTACHED"
   android:resource="@xml/accessory _ filter" />
```

Figur 7: Starta en applikation i Open Accessory-läge.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<usb-accessory manufacturer="Microchip Technology Inc."
model="Basic Accessory Demo" version="1.0" />
</resources>
```

Figur 8: Använd ett filter för att bestämma vilken enhet som ska starta appen.

En utmaning återstår nu: att förstå hur bulköverföringar via USB går till.

Enligt USB-specifikationen är USB-bulköverföringar kompletta när:

- den exakta mängden av förväntade data har sänts, och
- ett paket har sänts som är mindre än ändpunktens storlek, eller tomt.

FÖR ATT SLUTFÖRA en överföring av ett datablock som är en exakt multipel av ändpunktens storlek, i dagsläget 64 byte, måste utvecklaren avsluta med ett tomt paket. Om ett sådan aldrig skickas kan data komma att ligga kvar i Androids USB-drivrutin utan att överföras till Open Accessory File Stream, och därmed aldrig föras över till appen.

Även Open Accessory-ramverket måste ansöka om tillstånd i filen *AndroidManifest.xml* – se figur 6.

Enheten kan starta en app automatiskt med hjälp av textdata som överförs under de steg som krävs för att gå in i tillbehörsläge. Detta görs med xml-filer liknande USB Host API:et – se figur 7 och 8.

Open Accessory-ramverkets största nackdel är att det är frivilligt. Vissa tillverkare har därmed valt att inte inkludera det, så man kan inte utgå från att alla enheter med en lämplig OS-version kommer att stödja dess funktioner.

Stödet kan till och med vara inkluderat i en viss produktversion, men senare ha uteslutits i senare versioner av samma produkt.

Med lanseringen av Android 4.1 Jelly Bean kom version 2 av protokollet Android Open Accessory (AOA) som ger tillbehörsutvecklare två nya funktioner: stöd för digital audioutgång och styrning av HID-gränssnitt i tillbehörsläge.

STÖD FÖR DIGITAL AUDIOUTGÅNG gör att man lätt kan skapa dockningsenheter för audio för en Androidenhet. Audiodockor var möjliga med version 1 av AOA, men då krävdes att konstruktören skapade ett specialsytt protokoll och utnyttjade en specialsydd tillämpning. Standardtillämpningar skulle fortfarande ge audioutgång via headset eller högtalare.

Med AOA version 2 routas all audio på Androidenheten till USB-porten, vilket gör det möjligt för ljudet att fungera med samtliga appar och funktioner i enheten. Dessutom gör AOA version 2 att audiogränssnittet är åtkomligt med eller utan start av appen när den är dockad. Det räcker med att tillverkar- eller modellsträngen inte sänds till Androidenheten innan den går i tillbehörsläge för att dockning av tillbehöret ska tillåtas utan att det startas.

Styrning av HID-gränssnitt i tillbehörsläge var tidigare endast tillgängligt i USB-värdläge. Med AOA version 2 kan tillbehör sända HID-rapporter till tillhörande Androidenheter och till OS för att styra användaringångar.

Detta kan användas för styrning av audiodockor liksom för att skapa styrutrustning som mus, tangentbord och joystick.

FÖR ATT OMFORMA EN HÅRDVARA till ett effektivt Androidtillbehör måste man förstå möjligheterna och begränsningarna i gränssnittet. Du behöver antagligen stort kunnande inom allt från användning av programtrådar, trådlös kommunikation och användning av Android som USB Host till digital audio. Utvecklingsresurser för samtliga dessa områden, däribland gratis firmware och exempel på Androidtillbehör, finns på Microchips webbplatser. ■