Android Open Accessory 분석
(ODROID-ADK)

2011.10.20

한기태

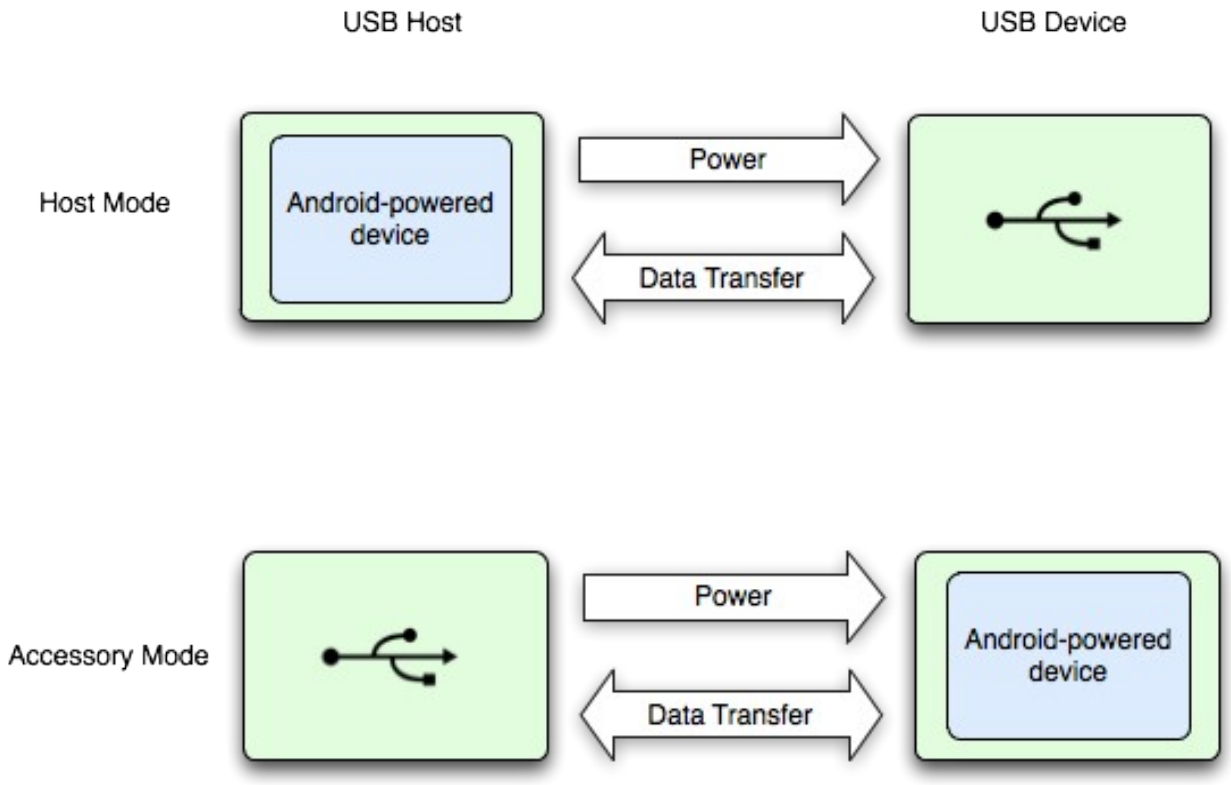
Android Open Accessory

이번 google IO 2011에 Android device를 가지고 외부장치(Accessory)를 제어할 수 있는
Android Open Accessory가 발표됐다.

Android Open Accessory의 가장 큰 특징은 Accessory를 제어하는 방법으로 USB를 사용하는 것이다.
기존의 Android에서 사용하던 USB 기능들

- USB mass storage
- adb
- USB tethering

에 Android Open Accessory기능이 추가된 것으로 보면 되는데, Android Open Accessory를 사용하기 위해서는
Android 3.1(API level 12), Android 2.3.4(API level 10) 이 후 version의 Android device를 사용해야한다.



그림에서 보듯이 Android device는 USB Host mode와 Accessory mode(USB Device)를 가질 수 있지만,
Android device 가 USB Host Mode가 되기 위해서는 몇 가지 제약 조건을 갔는다.

- H/W에세 USB host 지원
- Android 3.1이상에서 USB Host 관련 API를 제공

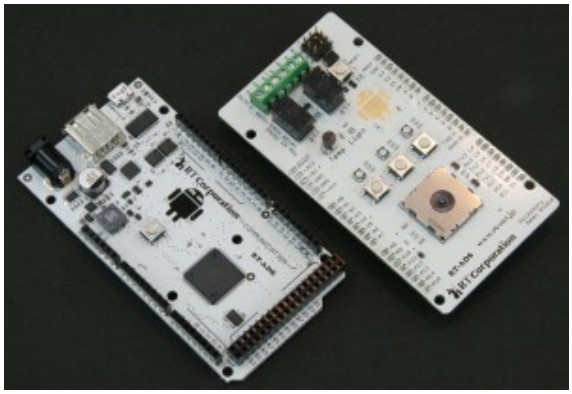
위의 제약 조건 왜에 여러가지 이유가 있겠지만 Accessory Mode는 USB Device로 동작한다.

Android Open Accessory Development Kit

Android Open Accessory Development Kit (이하 ADK) 는 Android의 Accessory를 만들수 있게 제공되는 target bord와 target bord의 동작을 정의하는 command set을 말한다.
다시 말해 ADK는 한정된 기능을 실행할수 있는 Accessory를 만들고, Accessory기능을 수행할 수 있는 command를 명시해서 Android device에서 정의된 command로 Accessory를 동작시키는 H/W와 command를 말한다.

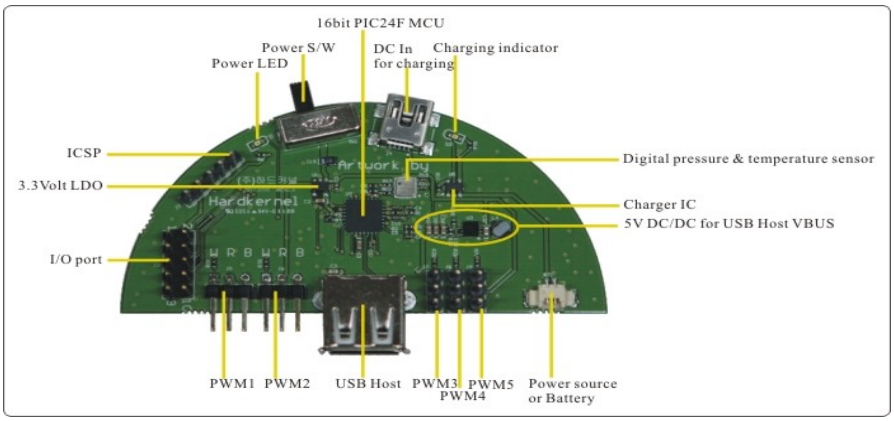
2011/10 현재 ADK를 구현한 bord는 아래와 같다.

- Arduino(ATMEL사의 ATmega2560)
  http://www.arduino.cc/en/Main/ArduinoBoardADK



- Microchip(PIC24F)

http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en553673



- FTDI(VNC2) http://www.ftdichip.com/Products/Modules/DevelopmentModules.htm#Vinco

USB Accessory Android Application

Android device는 Android accessory protocol을 구현한 Accessory device에 USB device로 붙어서 Accessory와 상호작용한다.
Accessory와 연결되는 Android device는 Android accessory mode를 지원해야 하면 현재(2011/10) Android 3.1(API level 12), Android 2.3.4(API level 10)이 Ancroid accessory mdoe를 지원한다.

Version별 USB Accessory API
Android version에 따라 지원되는 library가 다르기 때문에 Android device의 Android version을 확인 후 적합한 library를 사용해야 한다.

- Android 2.3.4 : Google add-on library로 지원 "com.android.future.usb"
- Android 3.1 : Android framework에서 지본 지원 "android.hardware.usb"

USB Acessory API Overview

USB accessory API 지원 class

Class	설명
UsbManager	연결된 USB accessory를 열거하고 통신한다.
UsbAccessory	USB accessory를 나타내고 Accessory 정보에 접근하는 method를 포함한다.

Version별 library 사용방법

Android 2.3.4(add-on library)

```
UsbManager 획득
UsbManager manager = UsbManager.getInstance(this);
UsbAccessory 획득
UsbAccessory accessory = UsbManager.getAccessory(intent);
```

Android 3.1

```
UsbManager 획득
UsbManager manager = (UsbManager) getSystemService(Context.USB_SERVICE);
UsbAccessory 획득
UsbAccessory accessory = (UsbAccessory) intent.getParcelableExtra(UsbManager.EXTRA_ACCESSORY);
```

Working with Accessory

Android device에 USB accessory가 연결됐을 때, application이 여결된 Accessory를 지원하는지 확인하고 지원하면 Accessory와의 통신을 설정한다.

1. Accessory attach event를 위한 intent filter를 사용해서 Accessory연결의 확인
2. accessory와 통신하기위한 permission을 요청
3. interface endpoint를 통해서 Accessory와 data를 read, write

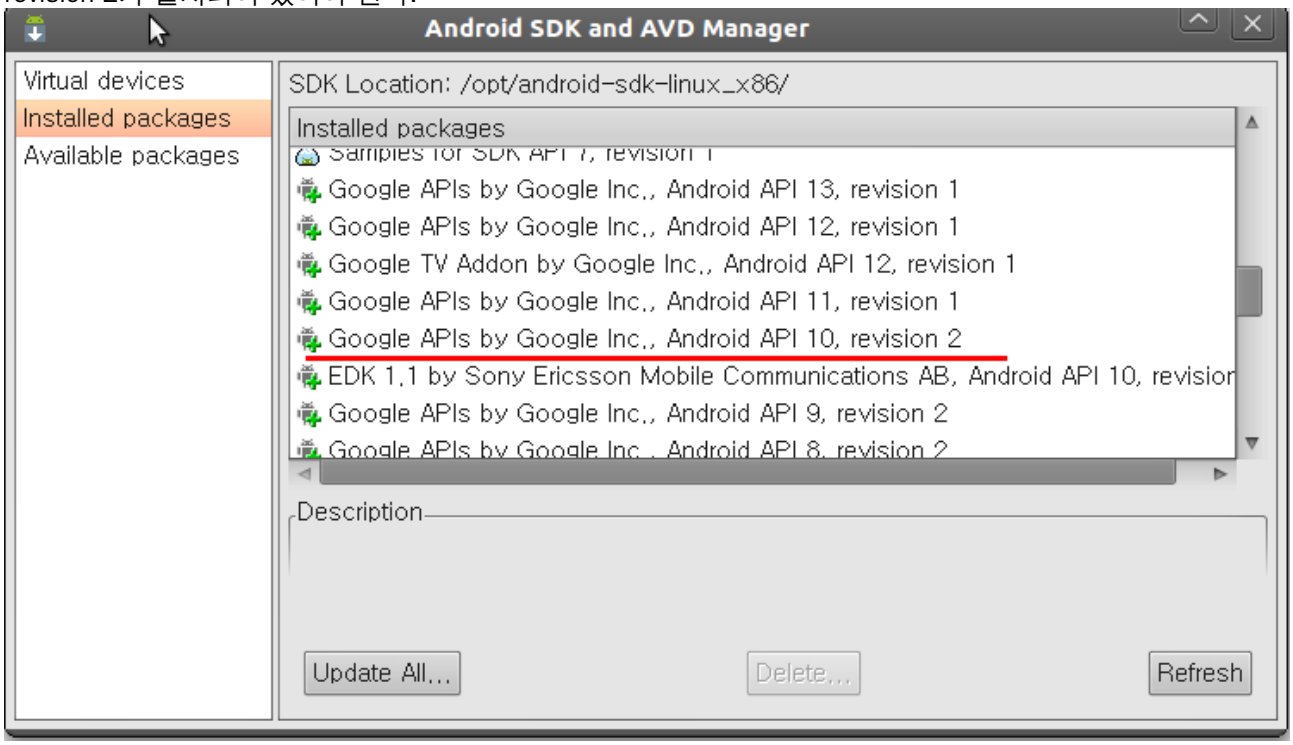
USB Host

Android device의 USB host mode 지원은 Android 3.1 이상의 버전에서만 지원한다.
USB Host를 사용하기 위해서는 H/W에서 USB host 기능을 제공해야 한다는 제약이있다.
아직까지 USB host를 지원하는 Android device는 별루 없다.

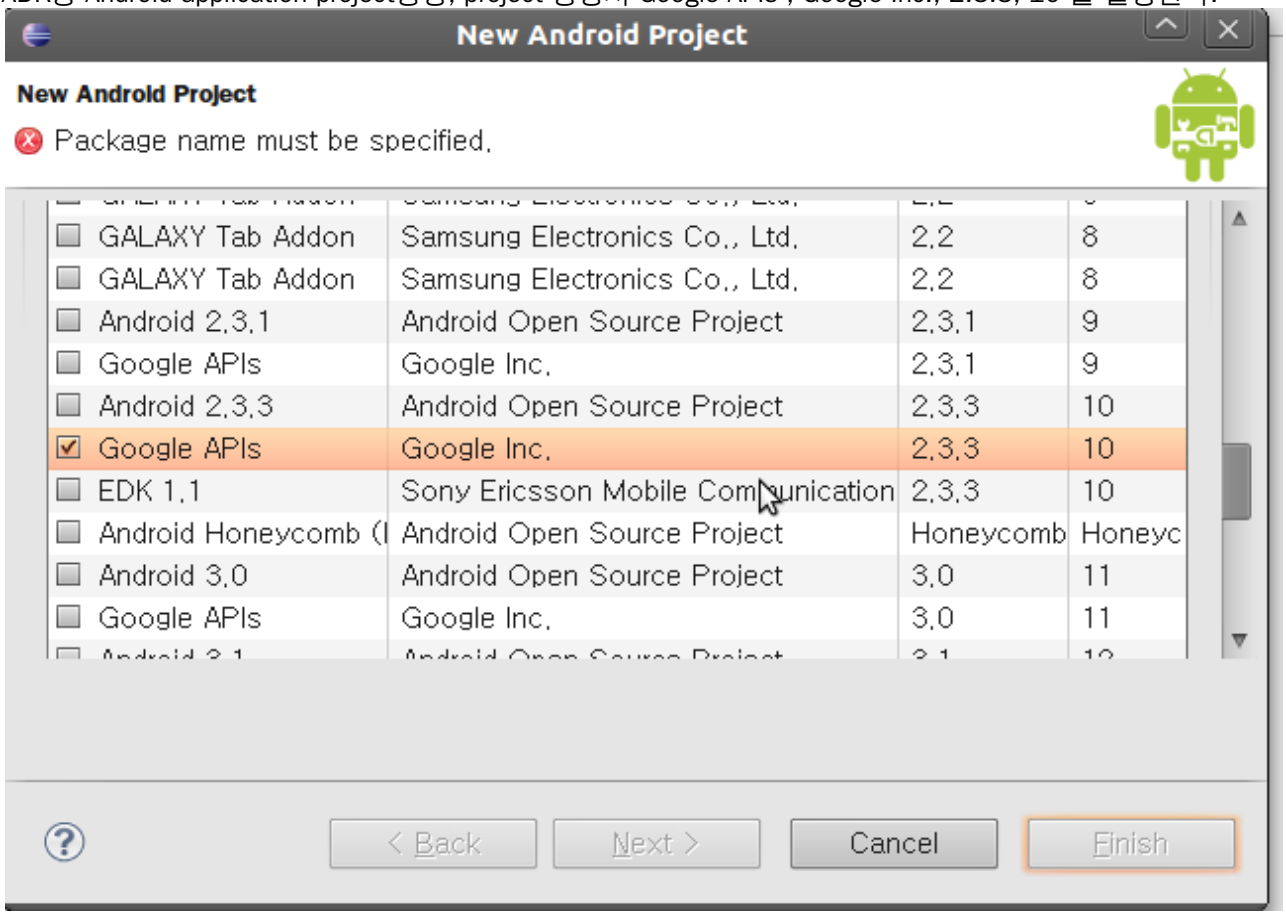
ADK용 Android application 작성

여기서는 Android 2.3.4 version을 기반으로 한다.

ADK용 Android application을 작성하기 위해서는 Android SDK에 Google APIs by Google Inc., Andoird API 10, revision 2가 설치되어 있어야 한다.



ADK용 Android application project생성, project 생성시 Google APIs , Google Inc., 2.3.3, 10 을 설정한다.



USB Accessory를 지원하기 위한 AndroidManifest.xml 성분을 추가한다.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.bnp.ADK.TestLED"
    android:versionCode="1"
    android:versionName="1.0">
  <uses-sdk android:minSdkVersion="10" />

  <uses-feature android:name="android.hardware.usb.accessory" />

  <application android:icon="@drawable/icon" android:label="@string/app_name">

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

    <activity android:name=".TestLED"
          android:label="@string/app_name">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>

      <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" />

    </activity>

  </application>
</manifest>
```

Accessory가 Android device에 연결됐을때 Accessory를 구별할 수 있는 Id string을 Android device에 전송한다. Application은 연결된 Accessory에 반응하기 위한 정보로 Accessory에서 보내는 Id string과 일치하는 값을 resource에 추가해야 한다.

accessory_filter.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <usb-accessory manufacturer="bnp, Inc." model="TestLED" version="1.0" />
</resources>
```

application에서 ADK를 다루기 위해서는 UsbManager부터 얻어와야 한다.

```
import android.app.Activity;
import android.os.Bundle;

import com.android.future.usb.UsbManager;
import com.android.future.usb.UsbAccessory;

public class TestLED extends Activity {

    private UsbManager mUsbManager;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        mUsbManager = UsbManager.getInstance(this);   ← UsbManager 획득


        setContentView(R.layout.main);
    }
}
```

AndroidManifest.xml에 정의한 intent filter를 사용하기 위한 추가 코드

```
    private UsbManager mUsbManager;

    private PendingIntent mPermissionIntent;
    private static final String ACTION_USB_PERMISSION =
        "com.bnp.ADK.TestLED.USB_PERMISSION";

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        mUsbManager = UsbManager.getInstance(this);
        mPermissionIntent = PendingIntent.getBroadcast(this, 0, new Intent(ACTION_USB_PERMISSION), 0);
        IntentFilter filter = new IntentFilter(ACTION_USB_PERMISSION);
```

Permission 처리

USB accessory와의 통신을 위해서는 Application에서 적합한 permission을 가저야 한다.

정확한 permission의 확인을 위해 intent filter를 이용, BroadcastReceiver를 통해서 확인한다.

BroadcastReceiver생성

```
private PendingIntent mPermissionIntent;
private static final String ACTION_USB_PERMISSION =
   "com.bnp.ADK.TestLED.USB_PERMISSION";

private final BroadcastReceiver mUsbReceiver = new BroadcastReceiver() {
   public void onReceive(Context context, Intent intent) {
      String action = intent.getAction();
      if(ACTION_USB_PERMISSION.equals(action)) {
         UsbAccessory accessory = UsbManager.getAccessory(intent);
         if (intent.getBooleanExtra(UsbManager.EXTRA_PERMISSION_GRANTED, false)) {
            if(accessory != null) {
               // call method to set up accessory communication
            }
            else {
               Log.d(TAG,"permission denied for accessory " + accessory);
            }
         }
      }
   }
};
```

BroadcastReceiver등록

```
mUsbManager = UsbManager.getInstance(this);
mPermissionIntent = PendingIntent.getBroadcast(this, 0, new Intent(ACTION_USB_PERMISSION), 0);
IntentFilter filter = new IntentFilter(ACTION_USB_PERMISSION);
registerReceiver(mUsbReceiver, filter);
```

연결된 Accessory의 permission을 사용자에게 묻기 위해 requestPermission()을 호출한다.

```
public void onResume() {
   super.onResume();

   Intent intent = getIntent();
   Log.d(TAG, "intent: " + intent);
   UsbAccessory[] accessories = mUsbManager.getAccessoryList();
   UsbAccessory accessory = (accessories == null ? null : accessories[0]);
   if (accessory != null) {
      if (mUsbManager.hasPermission(accessory)) {
         //call method to set up accessory communication
      } else {
         synchronized (mUsbReceiver) {
            mUsbManager.requestPermission(accessory, mPermissionIntent);
         }
      }
    } else {
      Log.d(TAG, "mAccessory is null");
   }
}
```

requestPermission()을 호출하면 다음과 같은 같은 Dialog가 pop된다.

Dialog가 에서 사용자가 “OK”를 클릭하면
broadcast receiver의 intent의 EXTRA_PERMISSION_GRANTED값이 true이면 permission을 정상적으로 획득한 경우로 Accessory와 통신을 연결할 수 있다.

```
private final BroadcastReceiver mUsbReceiver = new BroadcastReceiver() {
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        if(ACTION_USB_PERMISSION.equals(action)) {
            UsbAccessory accessory = UsbManager.getAccessory(intent);
            if (intent.getBooleanExtra(UsbManager.EXTRA_PERMISSION_GRANTED, false)) {
                if(accessory != null) {
                    // call method to set up accessory communication
                }
                else {
                    Log.d(TAG,"permission denied for accessory " + accessory);
                }
            }
        }
    }
};
```

USB Accessory 정보
Application은 연결된 USB Accessory를 UsbAccessory Class의 instance로 Accessory 정보에 접근할수 있다.
UsbAccessory를 얻는 방법은 AndroidManifest.xml에서 정의한 intent filter의 intent를 사용하는

```
UsbAccessory accessory = UsbManager.getAccessory(intent);
```

방법과 Android device에 연결된 모든 USB Accessory를 UsbAccessory배열로 받아오는

```
UsbAccessory[] accessoryList = manager.getAcccessoryList();
```

두가지 방법을 사용한다.

연결된 USB Accessory와 통신
USB Accessory와의 통신은 UsbManager의 file descriptor을 사용해서 Accessory와 data I/O를 수행한다.
USB Accessory와의 통신은 input용 file descriptor와 output용 file descriptor 가 필요하다.

```
private ParcelFileDescriptor mFileDescriptor;
private FileInputStream mInputStream;
private FileOutputStream mOutputStream;
```

USB Accessory의 통신을 위한 file descriptor획득 방법은 아래와 같다

```
private void openAccessory(UsbAccessory accessory) {
    Log.d(TAG, "openAccessory: " + accessory);
    mFileDescriptor = mUsbManager.openAccessory(accessory);
    if (mFileDescriptor != null) {
        FileDescriptor fd = mFileDescriptor.getFileDescriptor();
        mInputStream = new FileInputStream(fd);
        mOutputStream = new FileOutputStream(fd);
        Log.d(TAG, "openAccessory succeeded");
    } else {
        Log.d(TAG, "openAccessory fail");
    }
}
```

Android accessory protocol은 data I/O를 위해 각 16384 byte packet buffer를 지원한다.

USB Accessory와의 통신종료

통신이 끝나거나 Accessory가 분리되면, file descriptor를 닫는다.

```
public void onPause() {
    super.onPause();
    if (mFileDescriptor != null) {
        try {
            mFileDescriptor.close();
        } catch (IOException e) {
        } finally {
            mFileDescriptor = null;
        }
    }
}
```

ADK의 Firmware 작성

여기서는 Firmware를 작성하기 위한 ADK로 hardkernel사의 ODROID-ADK를 기반으로 한다.
ODROID-ADK의 정보는 아래에서 확인할 수 있다.
http://www.hardkernel.com/renewal_2011/products/prdt_info.php?g_code=G131063014581

ODROID-ADK는 Microchip사의 PIC24FJ64GB002를 MCU로 사양한다.
PIC24FJ64GB002의 정보는 아래에서 확인할 수 있다.
http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en536121

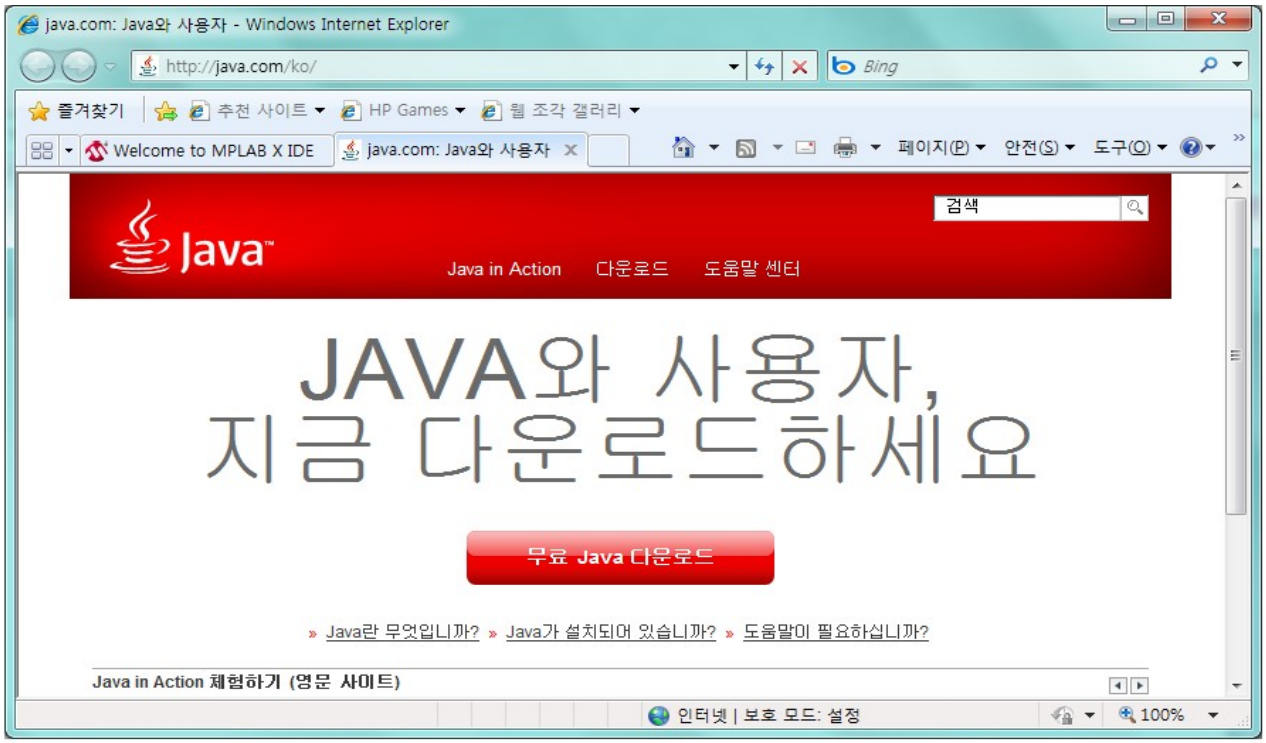
ADK의 Firmware를 개발하기위해서는 ADK board에서 사용한 MCU의 개발환경을 구축해야 한다.
여기서는 Microchip 개발을 위해서

- MPLAB_x IDE
- mplabc30 toolchain (PIC24용 compiler)
- pICkit3 debuger

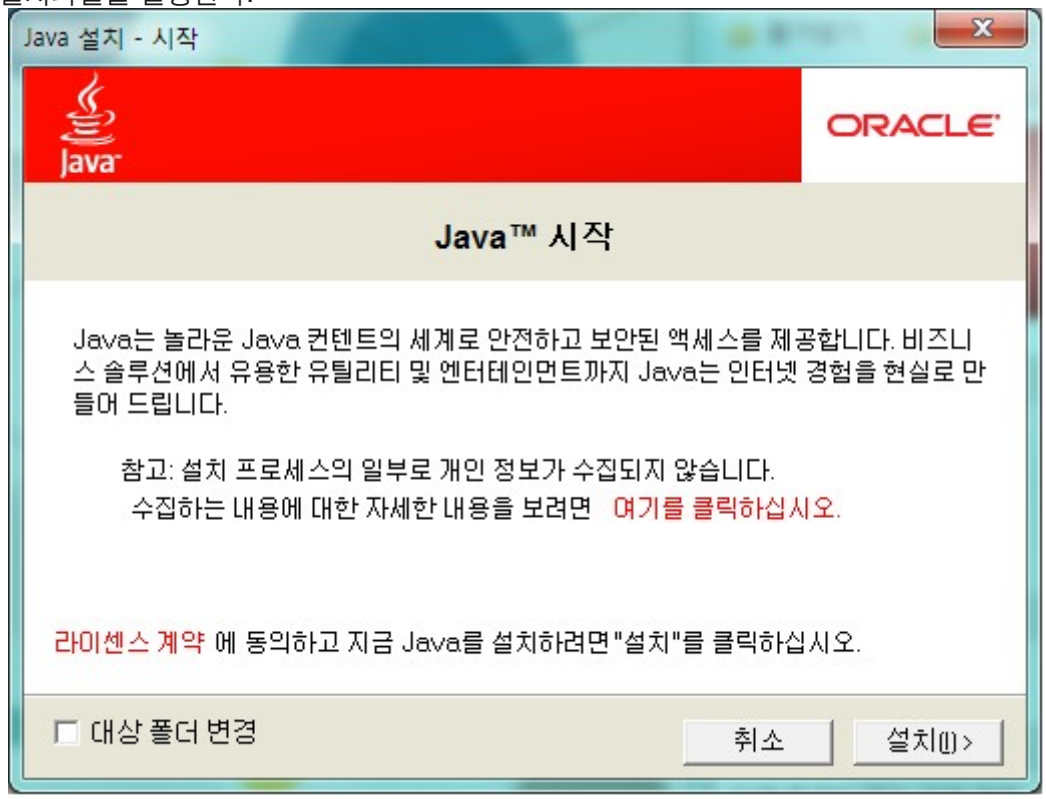
를 사용한다.

Windows에 Microchip(PIC24) 개발환경 구축

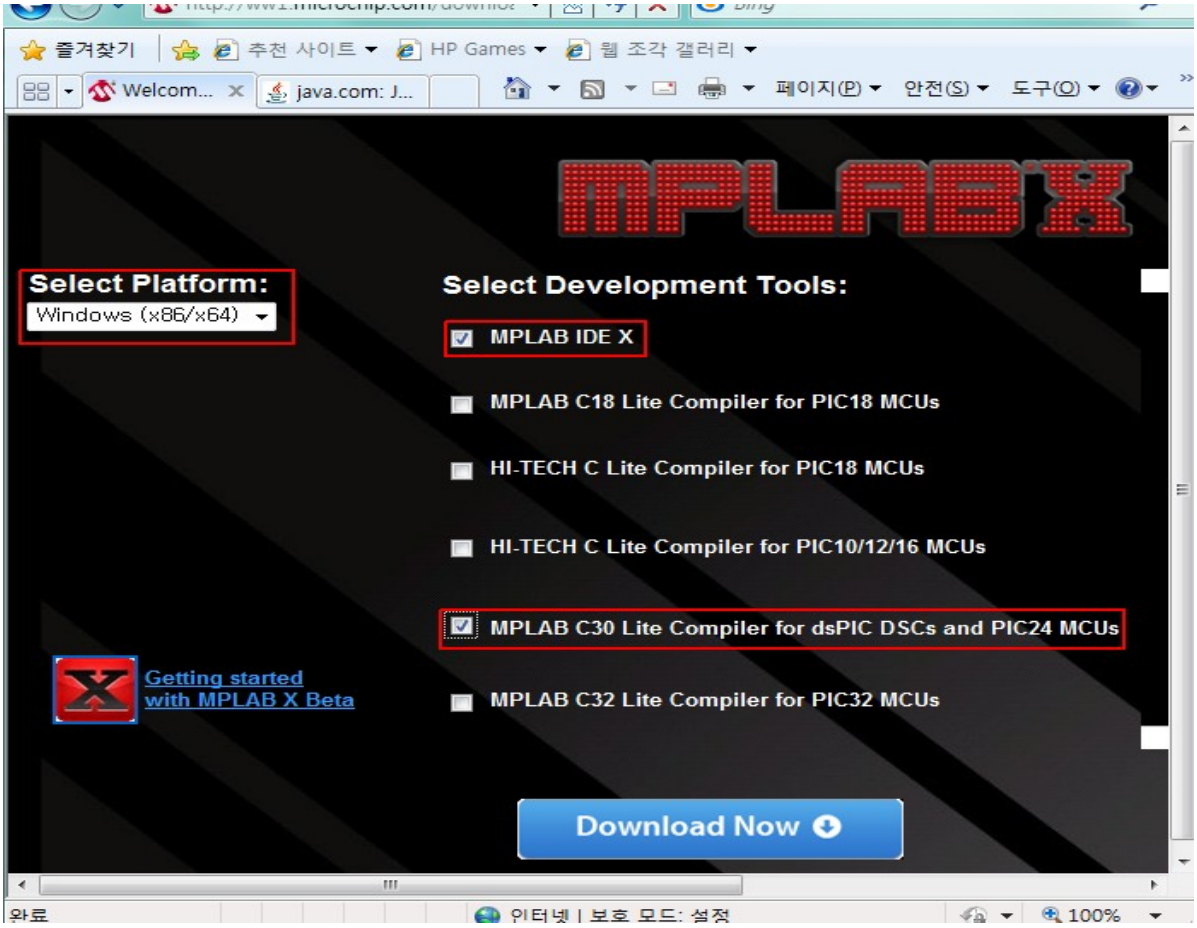
MPLAB_X IDE는 java로 만들어 졌기 때문에 java를 설치해야 한다.
홈페이지에서 다운받는다. (http://www.java.com/ko/)



다운받은 설치파일을 실행한다.

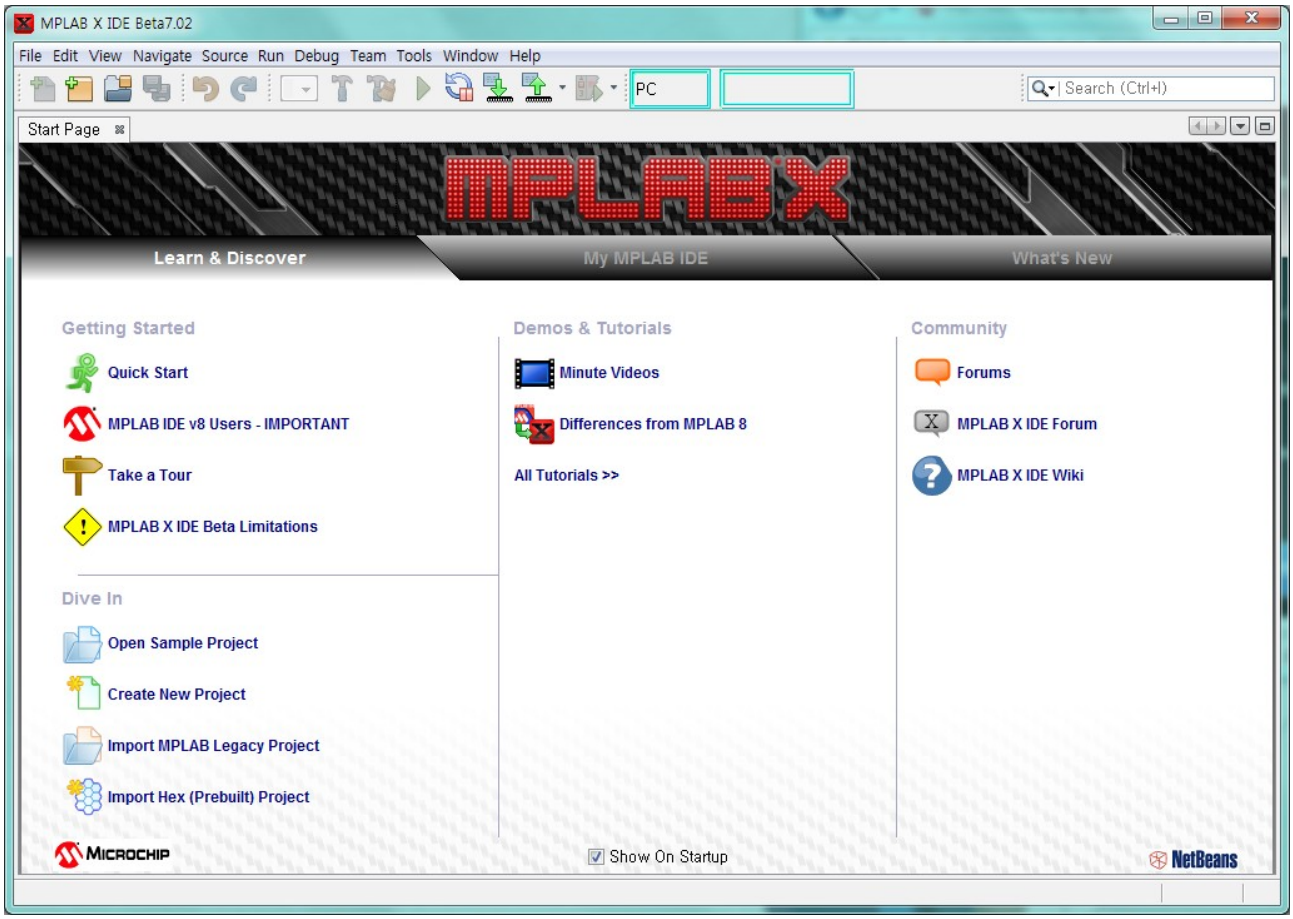


MPLAB_X 다운로드 사이트에서 설치할 OS에 맞는 설치파일을 다운 받는다.
다운 받을 파일은 MPLAB IDE X와 MPLAB C30 compiler이다.

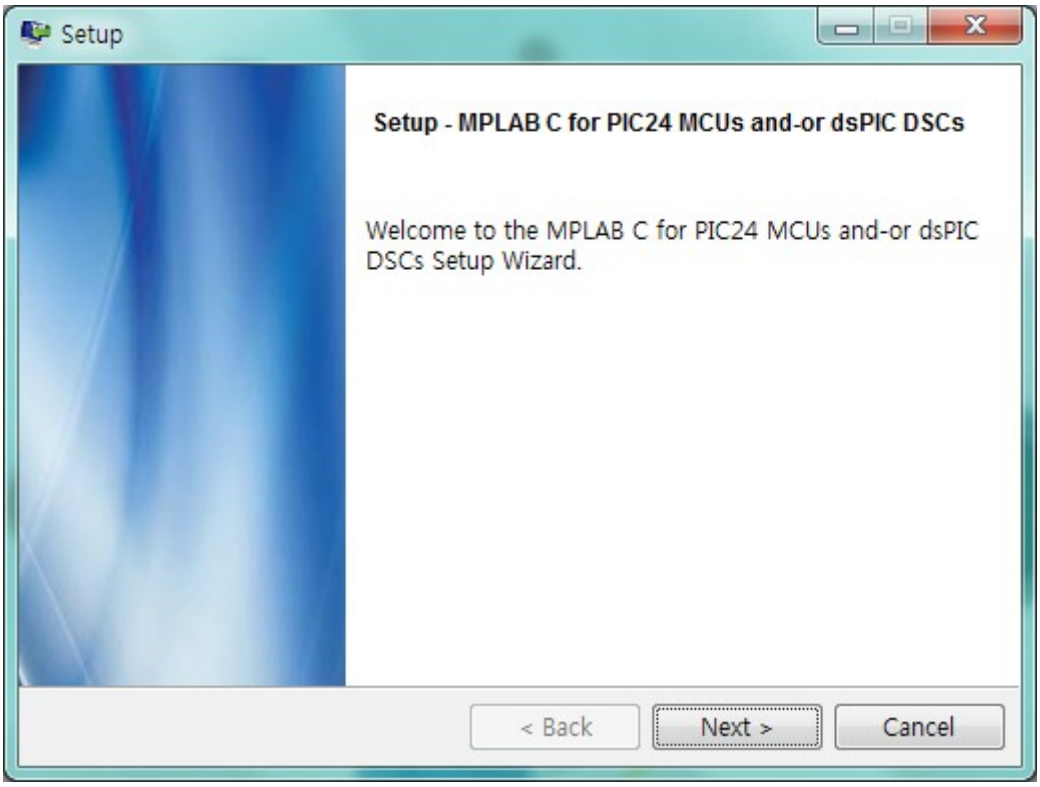


먼저 MPLAB IDE X를 설치한다.

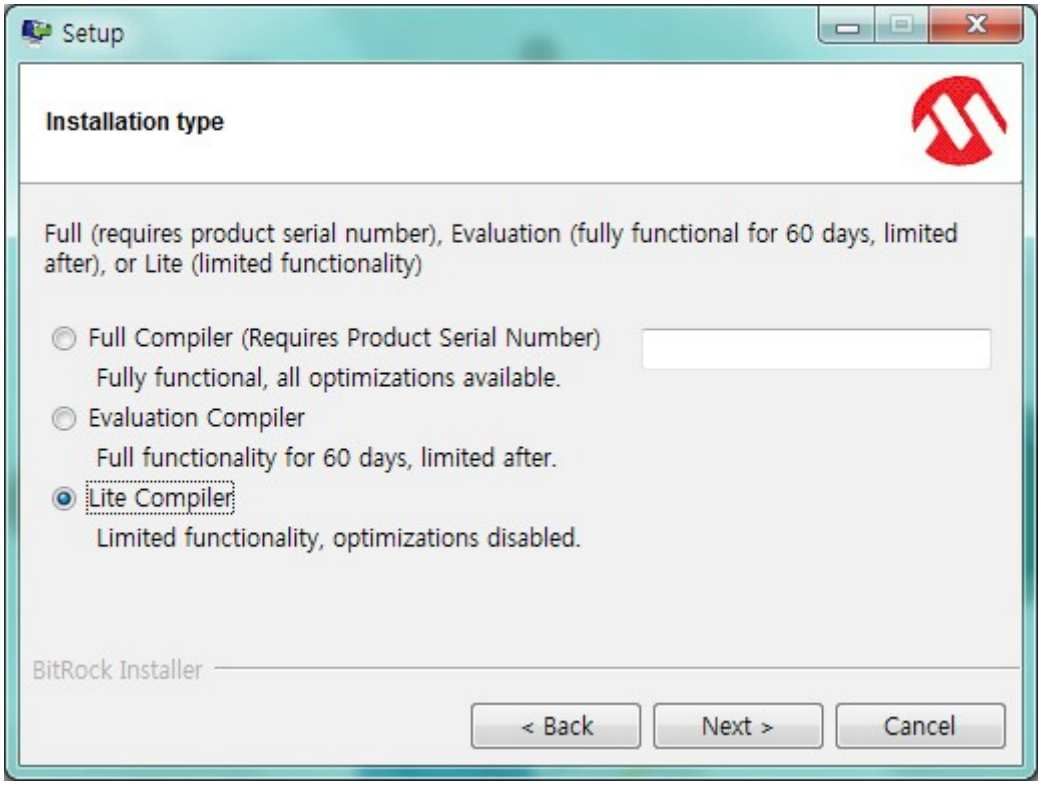
MPLAB_X 설치 완료



다음으로 MPLAB C30 compiler를 설치한다.



원하는 version을 선택한다. Light version도 개발하는데 무리가 없다.



개발환경에 대한 정보는 아래에서 확인할 수 있다.

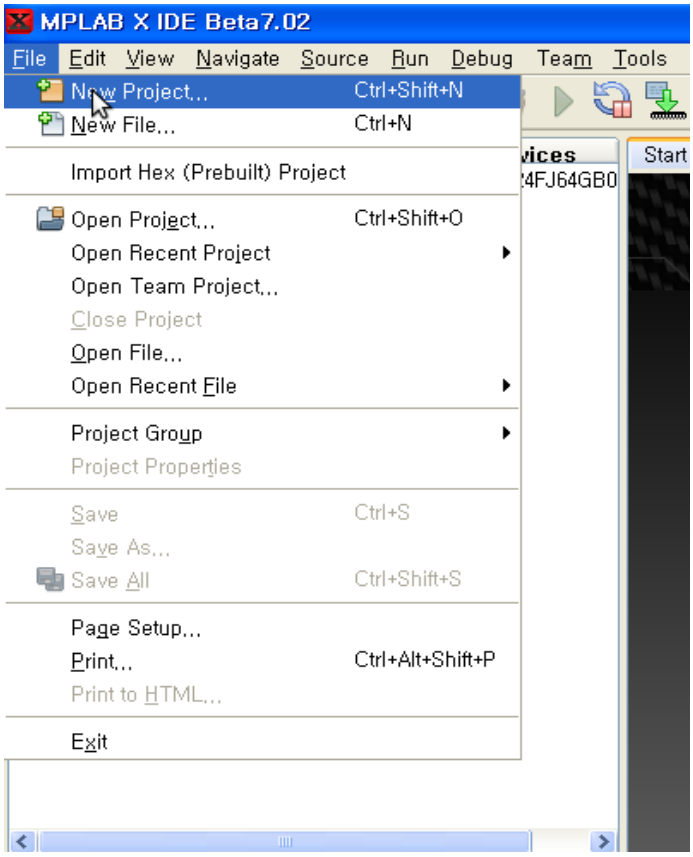
MPLAB_X : http://microchip.wikidot.com/mplab:_start

PICkit3 :

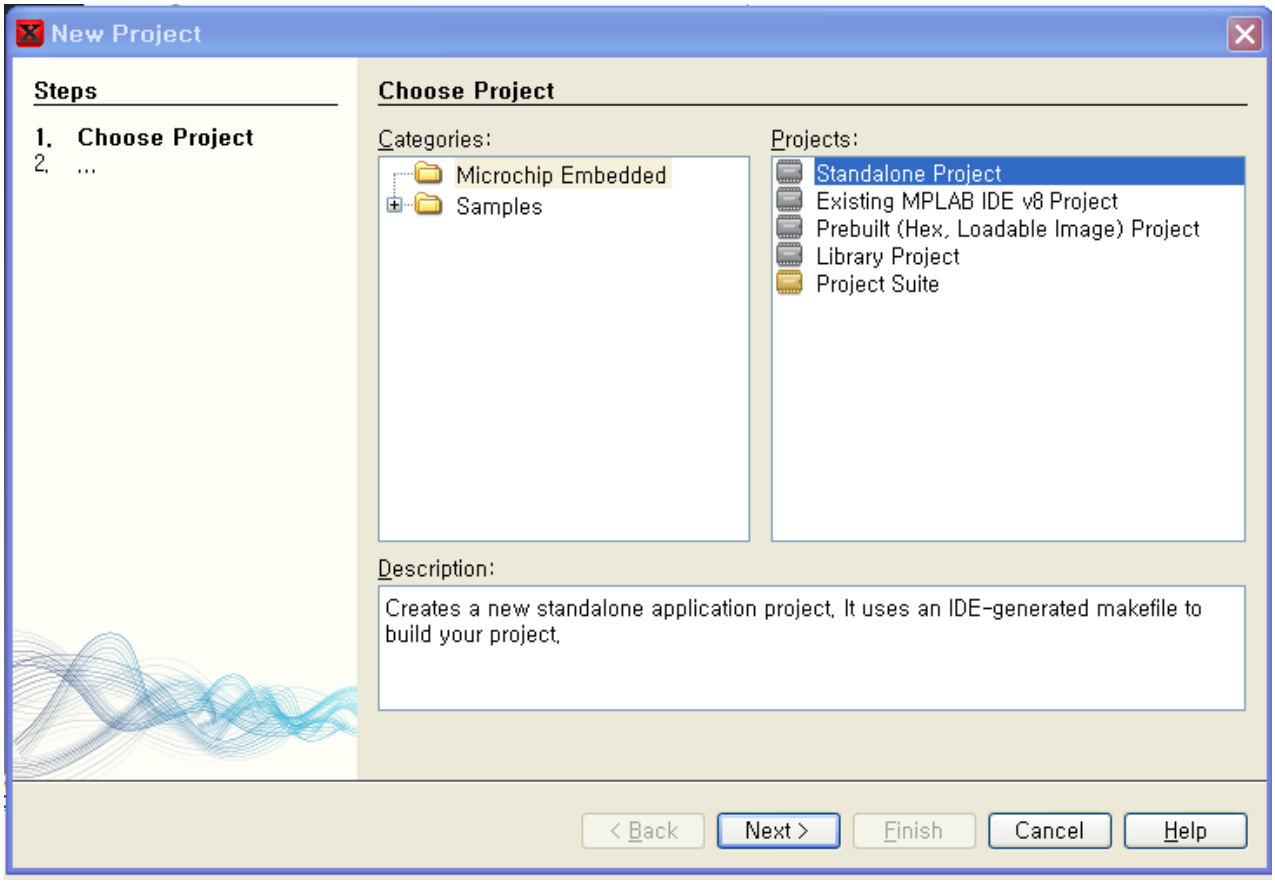
http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en538340

MPLAB_X 설치를 완료 했으면 다음의 순서로 firmwrare를 작성한다.

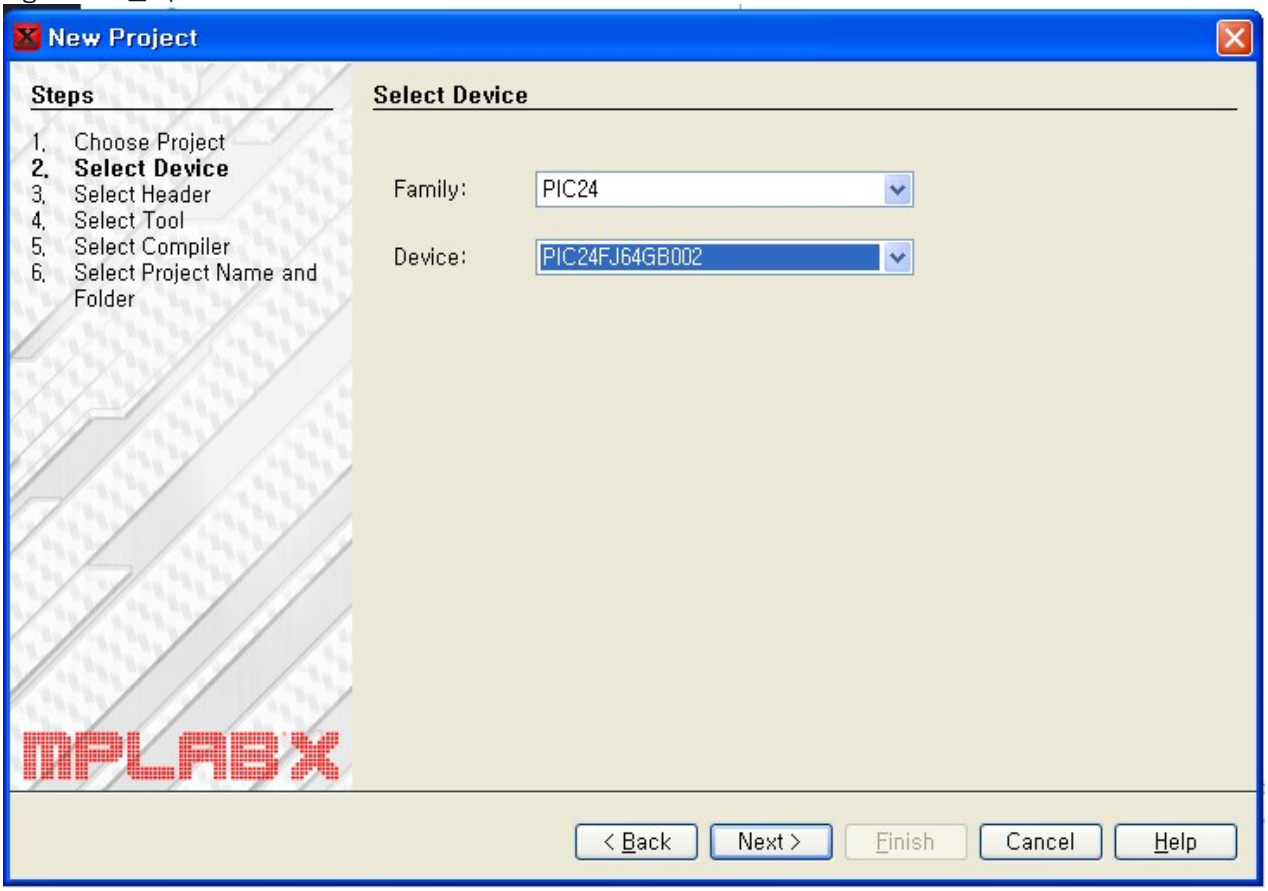
MPLAB_X 새로운 프로젝트 project생성



Project 선택

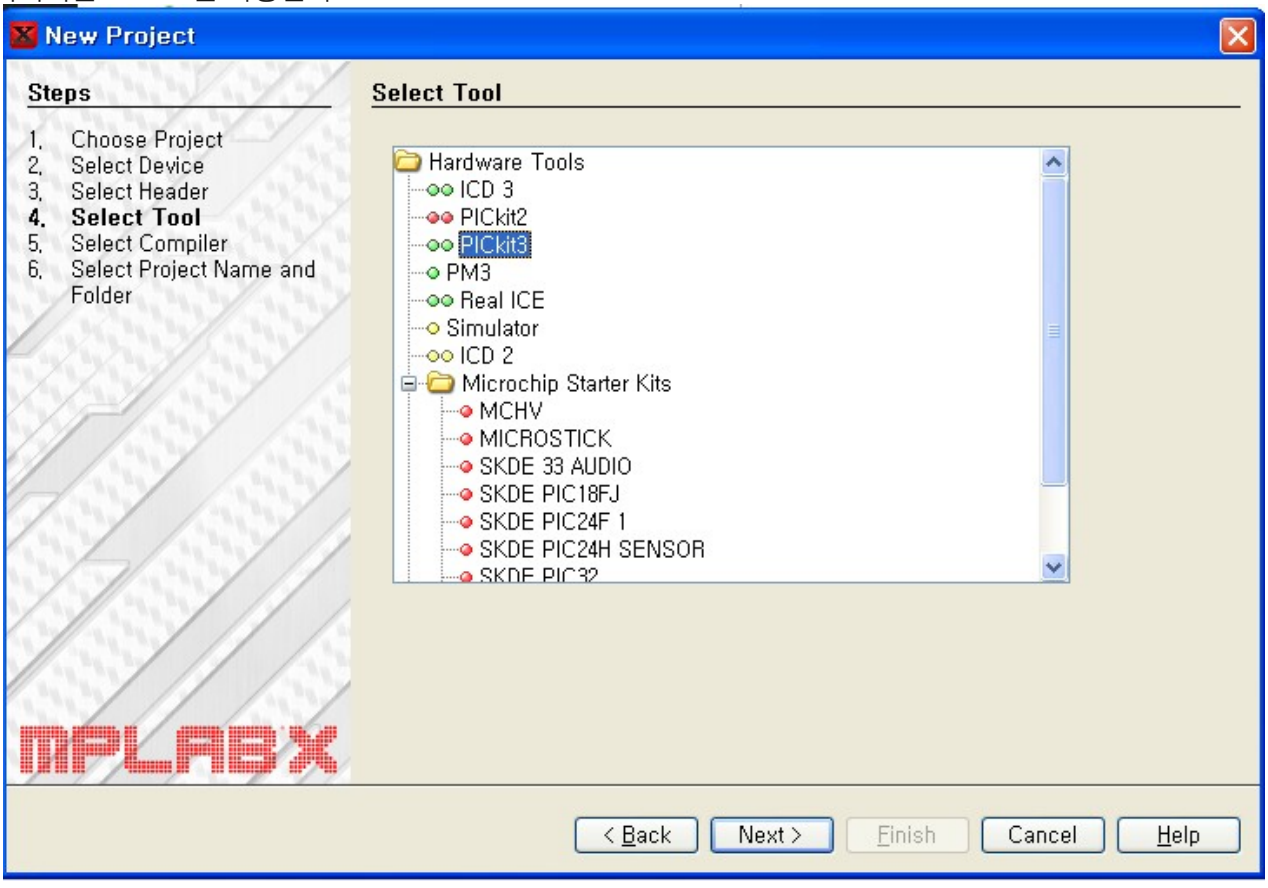


target MUC선택



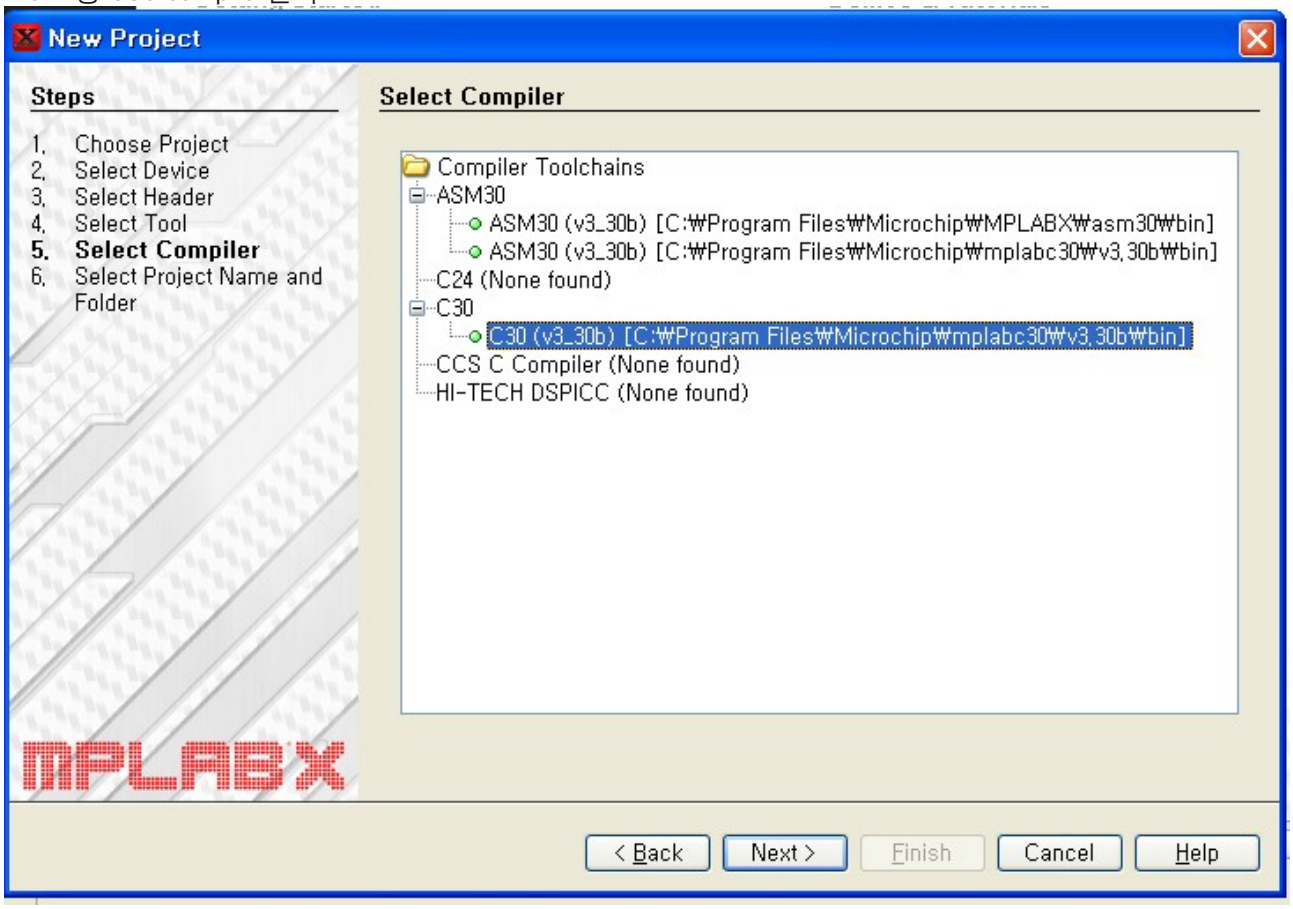
firmware writer(debuger) 선택

여기서는 PICKit3를 사용한다.

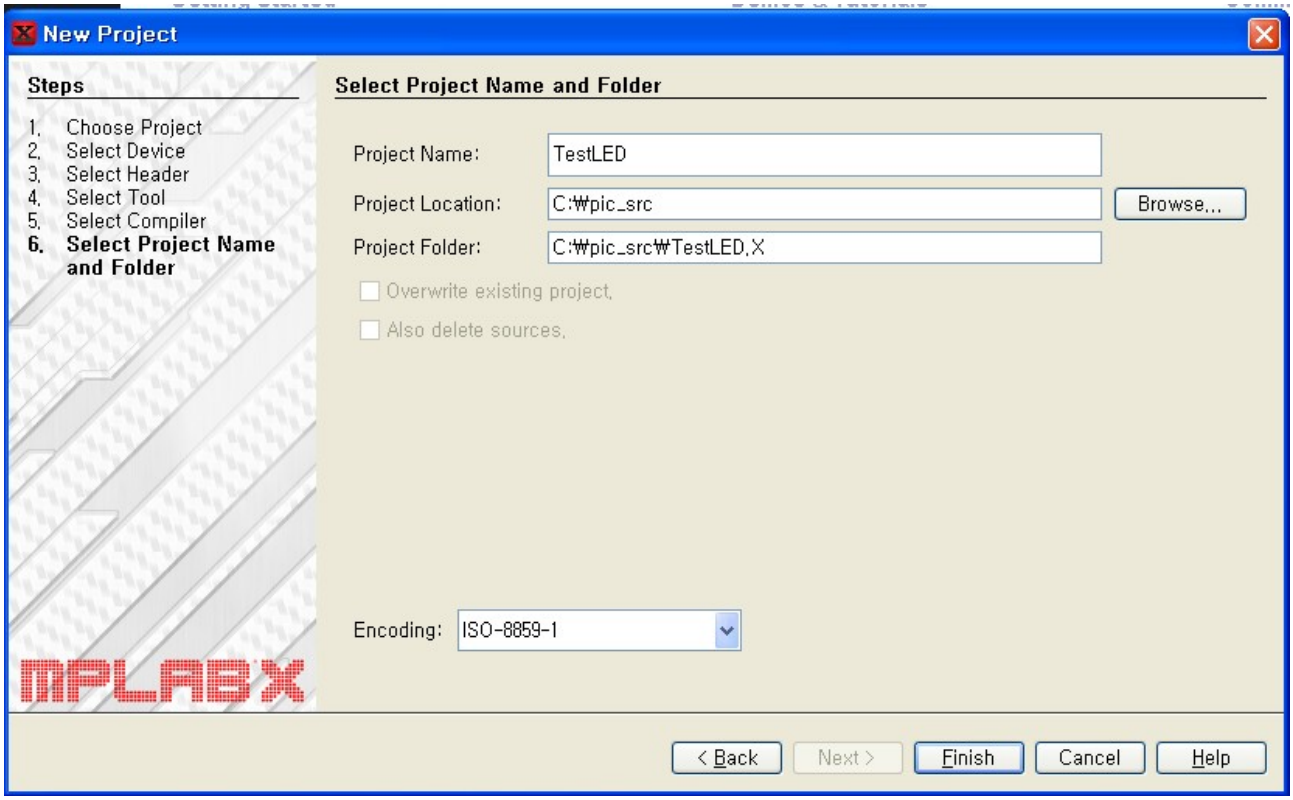


Compiler선택

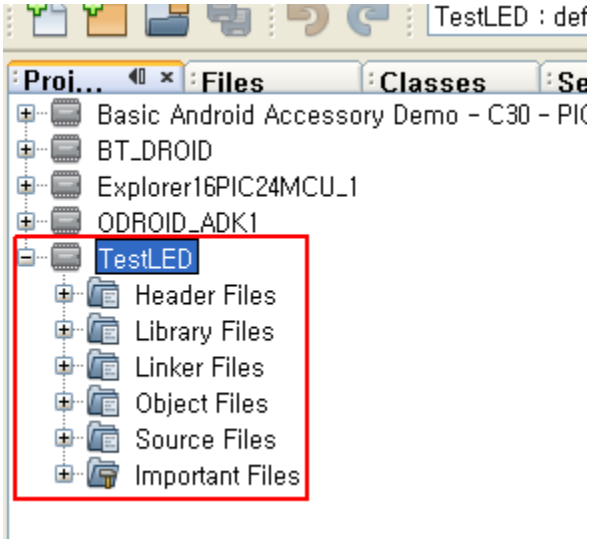
PIC24용 C30 compiler선택



Project Name & Directory path 선택



project 생성



firmware를 작성하기 위해서는 target mcu의 기능을 datasheet를 보고 직접 구현할 수도 있지만
MCU의 library를 사용하는 방법이 더 쉽고 빠르게 개발할 수 있다.

Microchip에서 제공하는 library는 아래에서 확인 가능하다.
http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=2680&dDocName=en547784

홈페이지 하단에 library를 os별로 제공한다.

Microchip Application Libraries

Downloads

Microchip Application Libraries v2011-07-14 Windows

Microchip Application Libraries v2011-07-14 Beta Linux

Microchip Application Libraries v2011-07-14 Beta Mac OS X

자신의 os에 맞는 파일을 다운받아 설치하면 microchip에서 제공하는 library source가 설치되는데,
우리는 USB Host, Android Accessory library를 주로 사용한다.

Accessory Firmware 작성

Firmware 작성 순서를 보면 대략 아래와 같다.

1. target board(MCU) 정보 설정
2. Microchp에서 제공한 library에서 필요한 file을 project에 복사
3. USB Host stack 초기화 및 handler 정의
4. Accessory library 최기화
5. Accessory data I/O 구현
6. Accessory 기능 구현

1. target board(MCU) 정보 설정

HardwareProfile.h :

system clock speed와 같은 configuration에 필요한 정보를 저장 한다.

```
…
#if defined (__C30__)
   // Various clock values
   #define GetSystemClock()           32000000UL
   #define GetPeripheralClock()       (GetSystemClock())
   #define GetInstructionClock()      (GetSystemClock() / 2)
#endif

// Define the baud rate constants
#define BAUDRATE2       57600 //19200
#define BRG_DIV2        4 //16
#define BRGH2           1 //0

#define DEMO_TIMEOUT_LIMIT  0xF000

#if defined(__PIC24F__) || defined(__PIC24H__)
   #include <p24Fxxxx.h>
#endif
…
```

main.c :

MUC의 Configuration Bit 설정 (Configuration은 datasheet 참조)

```
_CONFIG1(...)
_CONFIG2(...)
_CONFIG3(...)
_CONFIG4(...)
```

2. Microchp에서 제공한 library에서 필요한 file을 project에 복사
   Microchip은 ADK개발에 필요한 library를 prebuild해서 제공하고 있지만
   여기서는 ibrary source에서 직접 복사하는 방법을 사용한다.

```
Microchip
|-- Include
|    |-- Compiler.h
|    |-- GenericTypeDefs.h
|    |-- USB
|    |    |-- usb.h
|    |    |-- usb_ch9.h
|    |    |-- usb_common.h
|    |    |-- usb_hal.h
|    |    |-- usb_hal_pic24.h
|    |    |-- usb_host.h
|    |    `-- usb_host_android.h
|    `-- struct_queue.h
`-- USB
     |-- Android Host Driver
     |    |-- usb_host_android.c
     |    |-- usb_host_android_protocol_v1.c
     |    `-- usb_host_android_protocol_v1_local.h
     |-- usb_hal_local.h
     |-- usb_host.c
     `-- usb_host_local.h
```

   위의 파일을 Microchip library source에서 project로 복사한다.

3. USB Host stack 초기화 및 handler 정의
   USB Host stack의 사용과 Accessory에서 USB Host stack을 연결하기 위해서는
   usb_config.h , usb_config.c 를 만들어야 한다.

   usb_config.h :
   USB stack을 사용하기위한 여러가지 option을 제공하는 파일이다.

필수 enable

```
#define USB_SUPPORT_HOST
#define USB_ENABLE_TRANSFER_EVENT
#define USB_ENABLE_1MS_EVEN
#define USB_SUPPORT_BULK_TRANSFERS
#define NUM_TPL_ENTRIES 2
#define NUM_CLIENT_DRIVER_ENTRIES 1
#define USB_PING_PONG_MODE  USB_PING_PONG__FULL_PING_PONG  ← PIC32
#define USB_HOST_APP_DATA_EVENT_HANDLER USB_ApplicationDataEventHandler
#define USB_HOST_APP_EVENT_HANDLER USB_ApplicationEventHandle

#define USBTasks()              \
  {                             \
    USBHostTasks();             \     ←    USB Host 기능 활성
    AndroidTasks();             \     ←    Android Accessory Protocol 처리
  }
```

usb_config.c :

Targeted Peripheral List(TPL) : attach할 수 있는 USB device를 나열한다.(USB_TPL의 배열로 정의)

USB Host를 위한 Client Driver Function Pointer : CLIENT_DRIVER_TABLE 변수 정의

```
USB_TPL usbTPL[] =
{
   { INIT_VID_PID( 0x18D1ul, 0x2D00ul ), 0, 0, {0} }, // Android accessory
   { INIT_VID_PID( 0x18D1ul, 0x2D01ul ), 0, 0, {0} }, // Android accessory
};

// Accessory library에 정의된 USB Host library에서 호출하는 callback 함수
CLIENT_DRIVER_TABLE usbClientDrvTable[] =
{
   {
      AndroidAppInitialize,
      AndroidAppEventHandler,
      AndroidAppDataEventHandler,
      0
   }
};
```

host stack을 사용하기 위해서는 firmware soucrce(예 main.c)에

USB_ApplicationDataEventHandler(), USB_ApplicationEventHandler()

callback (usb_config.h 정의) 함수를 작성해야 한다.

USB_ApplicationDataEventHandler()는 Accessory에서는 별다른 처리할게 없다 만들놓기만 하면 된다.

```
BOOL USB_ApplicationDataEventHandler( BYTE address, USB_EVENT event, void *data, DWORD size )
{
   return FALSE;
}
```

USB_ApplicationEventHandler()는 Android Device의 attachment, detachment되면 USB Host stack이 호출된다.

```
BOOL USB_ApplicationEventHandler( BYTE address, USB_EVENT event, void *data, DWORD size )
{
        switch( event )
        {
                //Android device has been removed.
                case EVENT_ANDROID_DETACH:
                        ...
                        return TRUE;
                //Android device has been added. Must record the device handle
                case EVENT_ANDROID_ATTACH:
                        ...
                        return TRUE;
                //Handle other events here that are required...
                //...
      }
      ...
}
```

Firmware 초기화 시에 USB Host stack을 초기화 한다.

```
int main(void)
{
…
        //Initialize the USB host stack
        USBHostInit(0);
…
```

4. Accessory library 최기화

Accessory library의 사용하기 위해선 Accessory정보를 정의한다.

```
//Define all of my string information here
char manufacturer[] = "bnp Ltd.";
char model[] = "TestLED";
char description[] = "ADK - Accessory Development Starter Kit for Android (PIC24F)";
char version[] = "1.0";
char uri[] = "http://www.bitnpulse.com/";
char serial[] = "N/A";

//Pack all of the strings and their lengths into an
// ANDROID_ACCESSORY_INFORMATION structure
ANDROID_ACCESSORY_INFORMATION myDeviceInfo =
{
        manufacturer, sizeof(manufacturer),
        model, sizeof(model),
        description, sizeof(description),
        version, sizeof(version),
        uri, sizeof(uri),
        serial, sizeof(serial)
};
```

Firmware 초기화 시에 Android accessory library를 초기화 한다.

```
int main(void)
{
…
    // Android accessory library init
    AndroidAppStart(&myDeviceInfo);
…
```

Firmware의 main loop에선 USBTasks()(usb_config.h 정의)함수를 추가한다.
USBTasks()함수는 반복적으로 호출되야 USB Host stack에서 Android Accessory Protocol이 수행된다.

```
Int main(void)
{
   …
   while(1)
   {
      // Keep the USB stack running
      USBTasks();

      // Do my application specific stuff here
      // …
   }
}
```

5. Accessory data I/O 구현

Accessory 기능을 구현하는 Accessory library API는 아래와 같다.

Name	Description
AndroidAppIsReadComplete	Check to see if the last read to the Android device was completed
AndroidAppIsWriteComplete	Check to see if the last write to the Android device was completed
AndroidAppRead	Attempts to read information from the specified Android device
AndroidAppStart	Sets the accessory information and initializes the client driver information after the initial power cycles.
AndroidAppWrite	Sends data to the Android device specified by the passed in handle.
AndroidTasks	Tasks function that keeps the Android client driver moving

Data 송신

Data 송신에는 다음의 함수를 사용한다.

AndroidAppWrite() : Accessory에서 Android device로 data를 전송

AndroidAppIsWriteComplete() : 이 전의 전송 data가 완료여부 확인

위의 두 함수는 USBTasks() 함수를 호출한 후에 호출해야 한다.

```
while(1)
{
        USBTasks();
        …
        if(AndroidAppIsWriteComplete(device_handle, &errorCode, &size) == TRUE)
        {
                if(errorCode != USB_SUCCESS)
                {
                        //Error
                        DEBUG_ERROR("Error trying to complete write");
                }
```

```
        }
        …
        errorCode = AndroidAppWrite(device_handle,(BYTE*)&response_packet, 2);
        if( errorCode != USB_SUCCESS )
        {
                DEBUG_ERROR("Error trying to send button update");
        }
        …
}
```

Data 수신

Data 송신에는 다음의 함수를 사용한다.

AndroidAppRead () : Android device에서 data를 읽는다.

AndroidAppIsReadComplete () : data 수신의 완료여부 확인

위의 두 함수는 USBTasks() 함수를 호출한 후에 호출해야 한다.

```
while(1)
{
        //Keep the stack running
        USBTasks();
        …
        errorCode = AndroidAppRead(device_handle,
                        (BYTE*)&command_packet,
                        (DWORD)sizeof(ACCESSORY_APP_PACKET));
        //If the device is attached, then lets wait for a command from the application
        if( errorCode != USB_SUCCESS)
        {
                //Error
                DEBUG_ERROR("Error trying to start read");
        }
        …
        if(AndroidAppIsReadComplete(device_handle, &errorCode, &size) == TRUE)
        {
                //We've received a command over the USB from the Android device.
                if(errorCode == USB_SUCCESS)
                {
                        …
                }
                else
                {
                        //Error
                        DEBUG_ERROR("Error trying to complete read request");
                }
        }
        …
}
```

6. Accessory 기능 구현
   I/O되는 data의 message를 정의하고 기능을 구현한다.
   정의한 message는 Android application에서 I/O data로 사용한다.

세부사항은 “Microchip`s Accessory Framework for Android PDF” 참조

Android Accessory Protocol

Android USB accessory들은 Android device의 attach를 위한 'Android Accessory protocol'이 구현되어 있어야 한다.
Android Accessory Protocol은 USB stack을 이용하는 protocol이다.

USB

Android Accessory Protocol을 이해하는데 있어서 USB H/W적 특성까지 이해할 필요는 없어서 여기서는 설명하지 안는다. USB에 대한 자세한 정보는 http://en.wikipedia.org/wiki/Usb 참고한다.

Usb 장치는 크게

1. Host
   - entrie bus의 관리
   - 연결된 device와 hubs의 track
   - device와의 통신 초기화
   - power 제공
2. Device
   - endpoint를 통한 host와의 통신
   - host가 enumeration하는 동안 자신의 capability를 설명
   - 표준 또는 vendor 종속 기능 수행
   - host로 부터 power를 받는다.

로 이루어 진다.

USB device가 USB host에 연결되면 USB host는 연결된 device가 어떤 기능을 가지고 있는지에 대한 정보를 알아야 USB 통신이 이루어진다.
USB device의 이러한 정보는 USB Descriptors를 사용해서 USB host에게 알려준다.

USB Descriptors는 다음과 같이 구성된다.

- Device Descriptor
- Configuration Descriptor
- Interface Descriptor
- Endpoint Descriptor

Device Descriptor

device의 최사위 description이며 다음의 정보를 포함한다.

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of the Descriptor in Bytes (18 bytes)
1	bDescriptorType	1	Constant	Device Descriptor (0x01)
2	bcdUSB	2	BCD	USB Specification Number which device complies too.
4	bDeviceClass	1	Class	Class Code (Assigned by USB Org) If equal to Zero, each interface specifies it뭘 own class code If equal to 0xFF, the class code is vendor specified. Otherwise field is valid Class Code.

5	bDeviceSubClass	1	SubClass	Subclass Code (Assigned by USB Org)
6	bDeviceProtocol	1	Protocol	Protocol Code (Assigned by USB Org)
7	bMaxPacketSize	1	Number	Maximum Packet Size for Zero Endpoint. Valid Sizes are 8, 16, 32, 64
8	idVendor	2	ID	Vendor ID (Assigned by USB Org)
10	idProduct	2	ID	Product ID (Assigned by Manufacturer)
12	bcdDevice	2	BCD	Device Release Number
14	iManufacturer	1	Index	Index of Manufacturer String Descriptor
15	iProduct	1	Index	Index of Product String Descriptor
16	iSerialNumber	1	Index	Index of Serial Number String Descriptor
17	bNumConfigurations	1	Integer	Number of Possible Configurations

Configuration Descriptor

주로 하나의 Configuration을 가지지만 간혹 하나이상의 Configurationm을 가지는 device가 있다.
Configuration Descriptor은 다음의 정보를 포함한다.

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of Descriptor in Bytes
1	bDescriptorType	1	Constant	Configuration Descriptor (0x02)
2	wTotalLength	2	Number	Total length in bytes of data returned
4	bNumInterfaces	1	Number	Number of Interfaces
5	bConfigurationValue	1	Number	Value to use as an argument to select this configuration
6	iConfiguration	1	Index	Index of String Descriptor describing this configuration
7	bmAttributes	1	Bitmap	D7 Reserved, set to 1. (USB 1.0 Bus Powered) D6 Self Powered D5 Remote Wakeup D4..0 Reserved, set to 0.
8	bMaxPower	1	mA	Maximum Power Consumption in 2mA units

Interface Descriptor

device가 수행하는 특정기능을 나타낸다. Devcie는 하나이상의 Interface Descriptor를 가진다.
Interface Descriptor은 다음의 정보를 포함한다.

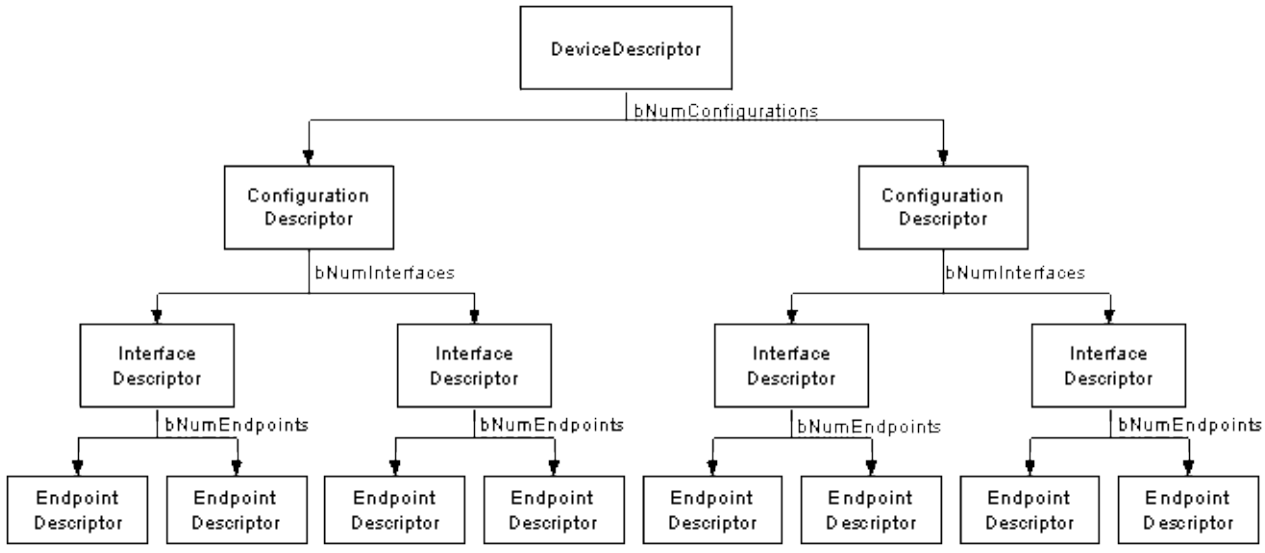
Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of Descriptor in Bytes (9 Bytes)
1	bDescriptorType	1	Constant	Interface Descriptor (0x04)
2	bInterfaceNumber	1	Number	Number of Interface
3	bAlternateSetting	1	Number	Value used to select alternative setting
4	bNumEndpoints	1	Number	Number of Endpoints used for this interface
5	bInterfaceClass	1	Class	Class Code (Assigned by USB Org)
6	bInterfaceSubClass	1	SubClass	Subclass Code (Assigned by USB Org)
7	bInterfaceProtocol	1	Protocol	Protocol Code (Assigned by USB Org)
8	iInterface	1	Index	Index of String Descriptor Describing this interface

Endpoint Descriptor

data의 송,수신을 위한 channel이다. Endpoint 0은 control endpoint를 나타낸다.
Endpoint Descriptor는 다음의 정보를 포함한다.

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of Descriptor in Bytes (7 bytes)
1	bDescriptorType	1	Constant	Endpoint Descriptor (0x05)
2	bEndpointAddress	1	Endpoint	Endpoint Address Bits 0..3b Endpoint Number. Bits 4..6b Reserved. Set to Zero Bits 7 Direction 0 = Out, 1 = In (Ignored for Control Endpoints)
3	bmAttributes	1	Bitmap	Bits 0..1 Transfer Type 00 = Control 01 = Isochronous 10 = Bulk 11 = Interrupt Bits 2..7 are reserved. If Isochronous endpoint, Bits 3..2 = Synchronisation Type (Iso Mode) 00 = No Synchonisation 01 = Asynchronous 10 = Adaptive 11 = Synchronous Bits 5..4 = Usage Type (Iso Mode) 00 = Data Endpoint 01 = Feedback Endpoint 10 = Explicit Feedback Data Endpoint 11 = Reserved
4	wMaxPacketSize	2	Number	Maximum Packet Size this endpoint is capable of sending or receiving
6	bInterval	1	Number	Interval for polling endpoint data transfers. Value in frame counts. Ignored for Bulk & Control Endpoints. Isochronous must equal 1 and field may range from 1 to 255 for interrupt endpoints.

USB Descriptors의 전체적 구조는 다음과 같다.

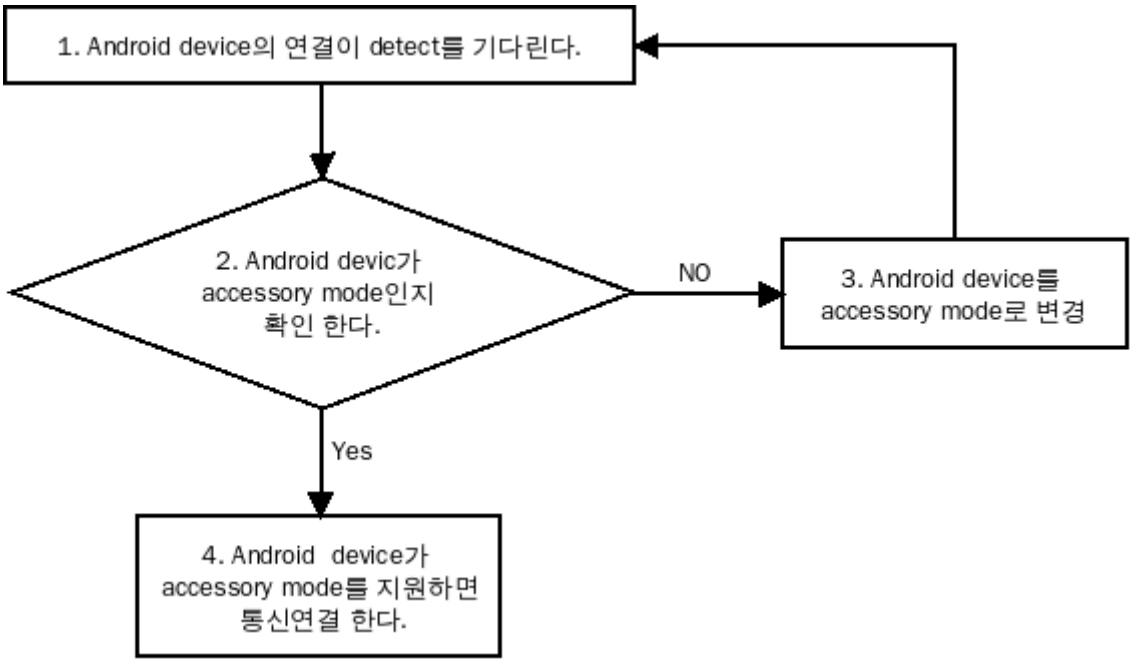


Endpoint는 4가지 종류가 있다.

* Control Endpoint : 0번 endpoint로 양방향 통신을 하며 device에게 명령을 내리는 통로이다.
* Bulk Endpoint : 많은 양의 data를 시간에 구애받지 않고 정확히 전달한다.
* Interrupt Endpoint : 일정시간 마다 비교적 적은 data를 전달한다.
* Isochronous Endpoint : 많은 양의 data를 정해진 시간 안에 전달한다. Data의 무결성은 보장 않는다.

ADK의 Android Accessory Protocol 구현

Android device와 Accessory의 통신 수행 절차는 다음과 같다.



1. Android device의 연결이 detect를 기다린다.

좀더 명확하게 말하자면 ADK는 USB Host에 USB device의 detect을 기다리는 것이다.

여기서는 ADK와 Android device의 물리적 연결을 말하며, ADK는 연결된 device가 Android device의 Accessory mode인지를 확인하기 위해서 2번 과정을 실행한다.

2. Android device가 accessory mode인지 확인 한다.

연결된 Android device는 다음 중의 하나의 상태를 가진다.

- 검출된 Android device가 accessory mode를 지원하고 이미 accessory mode이다.
- 검출된 Android device가 accessory mode를 지원하고 accessory mode가 아니다.
- 검출된 Android device가 accessory mode를 지원하지 안는다.

ADK에 연결된 device가 어떤 상태인가는 USB device descriptor의 venderID와 productID로 판단한다.

VenderID : 0x18D1(google ID) 이면 연결된 device는 Android device이다.

ProductID : 0x2D00(accessory mode), 0x2D01(accessory mode + adb) 이면 accessory mode이다.

연결된 Android device가 accessory mode이면 4번, accessory mode가 아니면 Android device에게 accessory mode를 요청하기 위해 3번을 실행한다.

3. Android device를 accessory mode로 변경

venderID와 productID가 맞지 안으면 Android accessory mode지원 유무를 확인하고 accessory mode의 실행을 요청한다.

요청 방법은 다음과 같다.

ㄱ. ADK는 USB control request로 51(“Get Protocol”)를 Android device에 보내면
Android device는 지원하는 Android accessory protocol의 version을 반환한다.
(현재는 버전 1, 반환값이 0이면 지원 안함)
* control request는 Control Endpoint(endpoint 0)으로 전달된다.

control request 값
requestType: USB_DIR_IN | USB_TYPE_VENDOR
request: 51
value: 0
index; 0
data: protocol version number(16 bit little endian sent from the device to the accessory)

ㄴ. Android device에서 적당한 protocol 버전이 반환됐다면,
Accessory는 identifying string을 Android device에게 전송한다.
이 정보는 Android device가 Accessory에 전당한 app를 찾는데 사용된다.

Control request 값
requestType: USB_DIR_OUT | USB_TYPE_VENDOR
request: 52
value: 0
index; string ID
data: zero terminated UTF8 string sent from accessory to device

다음은 Arduino string Id의 예이다.(각 string 별 최대 256 Byte이고 zero terminate이어야 한다.)
manufacturer name: Google, Inc.
model name: DemoKit
description: DemoKit Arduino Board
version: 1.0
URI: http://www.android.com
serial number: 00000012345678

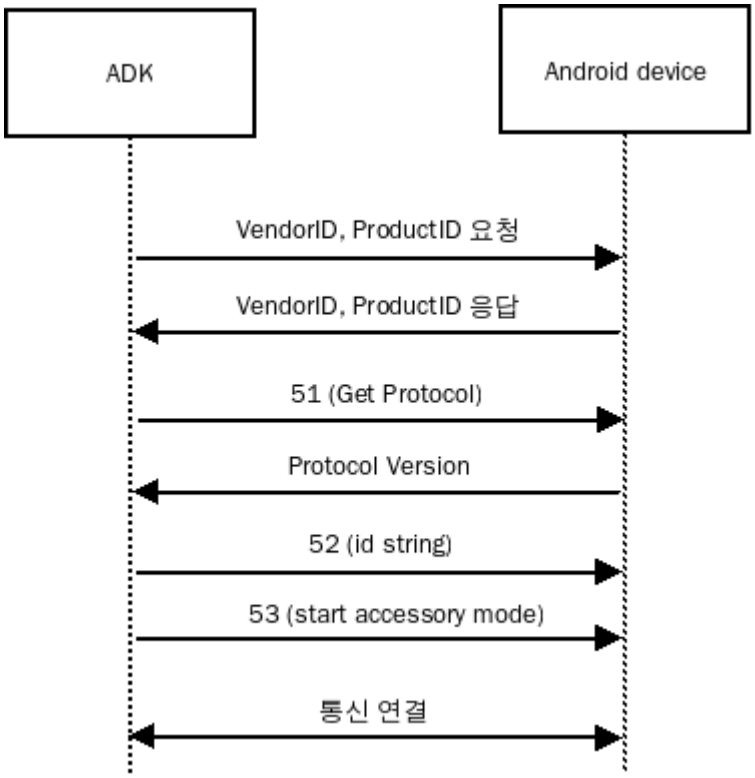
ㄷ. Android device에 accessory mode 실행(53)을 요청한다.

Control request 값
requestType: USB_DIR_OUT | USB_TYPE_VENDOR
request: 53
value: 0
index; 0
data: none

모든 요청절차가 완료되면 ADK는 1번부터 다시 실행한다.
위의 절차에서 어느 한 부분이라도 실해하면, ADK는 다음 Android device를 connect를 기다린다.

4. Android device가 accessory mode를 지원하면 통신연결 한다.
accessory mode의 Android device가 detect되면, Accessory는 endpoint descriptor를 이용해서 bulk endpoint를 연결한다.
endpoint는input, output용으로 각 한개씩 생성된다.

Android Accessory Protocol 흐름



Android의 Android Accessory

kernel

Kernel은 Accessory용 USB Gadget driver가 추가된다

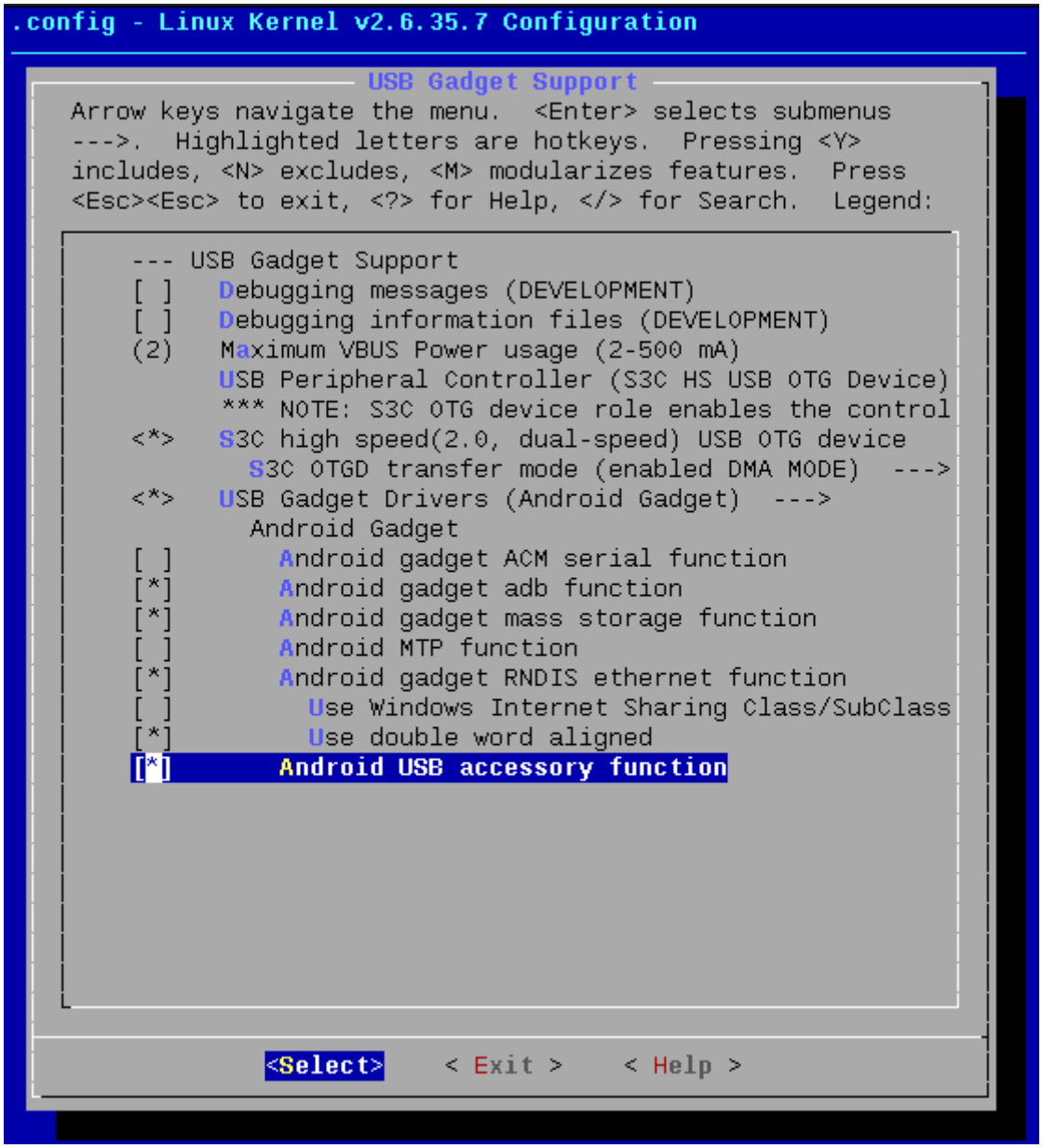
자세한 부분은 source(<kernel>/include/linux/usb/f_accessory.h ,

<kernle>/drivers/usb/gadget/f_accessory.c)를 참고한다.

Linux Kernel configuartion 추가 사항

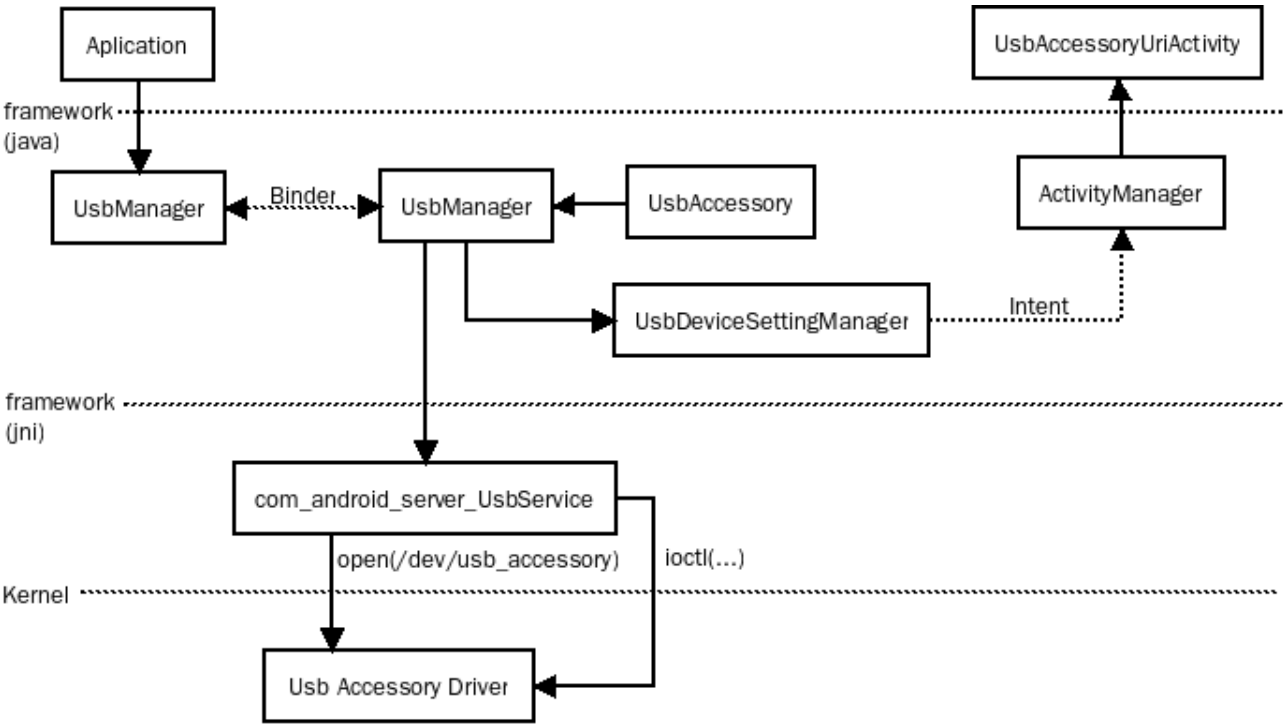
Android accessory module 추가

Device Drivers –> [*] USB support –> <*> USB Gadget Support –> [*] Android USB accessory function



Android Accessory framework구조 (version android-3.3.5)

Android Accessory framework 간단한 diagram



source

UsbManager의 Instance생성은 “usb” service의 binder node를 얻어온다.

```
Found 54 services:
0       sip: [android.net.sip.ISipService]
1       phone: [com.android.internal.telephony.ITelephony]
2       iphonesubinfo: [com.android.internal.telephony.IPhoneSubInfo]
3       simphonebook: [com.android.internal.telephony.IIccPhoneBook]
4       isms: [com.android.internal.telephony.ISms]
5       diskstats: []
6       appwidget: [com.android.internal.appwidget.IAppWidgetService]
7       backup: [android.app.backup.IBackupManager]
8       uimode: [android.app.IUiModeManager]
9       usb: [android.hardware.usb.IUsbManager]
10      audio: [android.media.IAudioService]
11      wallpaper: [android.app.IWallpaperManager]
12      dropbox: [com.android.internal.os.IDropBoxManagerService]
13      search: [android.app.ISearchManager]
```

```
/* package com.android.future.usb.UsbManager */

public static UsbManager getInstance(Context context) {
        IBinder b = ServiceManager.getService(Context.USB_SERVICE);
        return new UsbManager(context, IUsbManager.Stub.asInterface(b));
}
```

UsbAccessory instance 생성

```
/* package com.android.future.usb.UsbManager */

public UsbAccessory[] getAccessoryList() {
        try {
                android.hardware.usb.UsbAccessory accessory = mService.getCurrentAccessory();
…
}

/* package com.android.server.usb.UsbService */

private UsbAccessory mCurrentAccessory;

public UsbAccessory getCurrentAccessory() {
        return mCurrentAccessory;    // 정장되어 있던 UsbAccessory Instance 생성
}

// 연결된 Accessory의 UsbAccessory Instance생성 과정
private final void readCurrentAccessoryLocked() {
        if (mHasUsbAccessory) {
                String[] strings = nativeGetAccessoryStrings();    // com_android_server_UsbService call
                if (strings != null) {
                        mCurrentAccessory = new UsbAccessory(strings);
                …
                       // UsbDeviceSettingsManager에 연결된 accessory 등록
                        mDeviceManager.accessoryAttached(mCurrentAccessory);
                …
                }
        …
}

private final void functionEnabledLocked(String function, boolean enabled) {
…
        readCurrentAccessoryLocked();
…
}

private final Handler mHandler = new Handler() {     // UeventObserver의 event Handler
…
        public void handleMessage(Message msg) {
        …
                case MSG_FUNCTION_ENABLED:
                case MSG_FUNCTION_DISABLED:
                        functionEnabledLocked(…);
                        break;
```

```
        }
}

/* package com.android.server.usb.UsbDeviceSettingsManager */

public void accessoryAttached(UsbAccessory accessory) {
…
        resolveActivity(intent, matches, defaultPackage, accessory);
}

private void resolveActivity(Intent intent, ArrayList<ResolveInfo> matches, …) {
…
        Intent dialogIntent = new Intent();
        dialogIntent.setClassName("com.android.systemui",
                        "com.android.systemui.usb.UsbAccessoryUriActivity");
…
        // ActivityManager에  UsbAccessoryUriActivity실행 요청
        mContext.startActivity(dialogIntent);
…
}
```

file descriptor획득

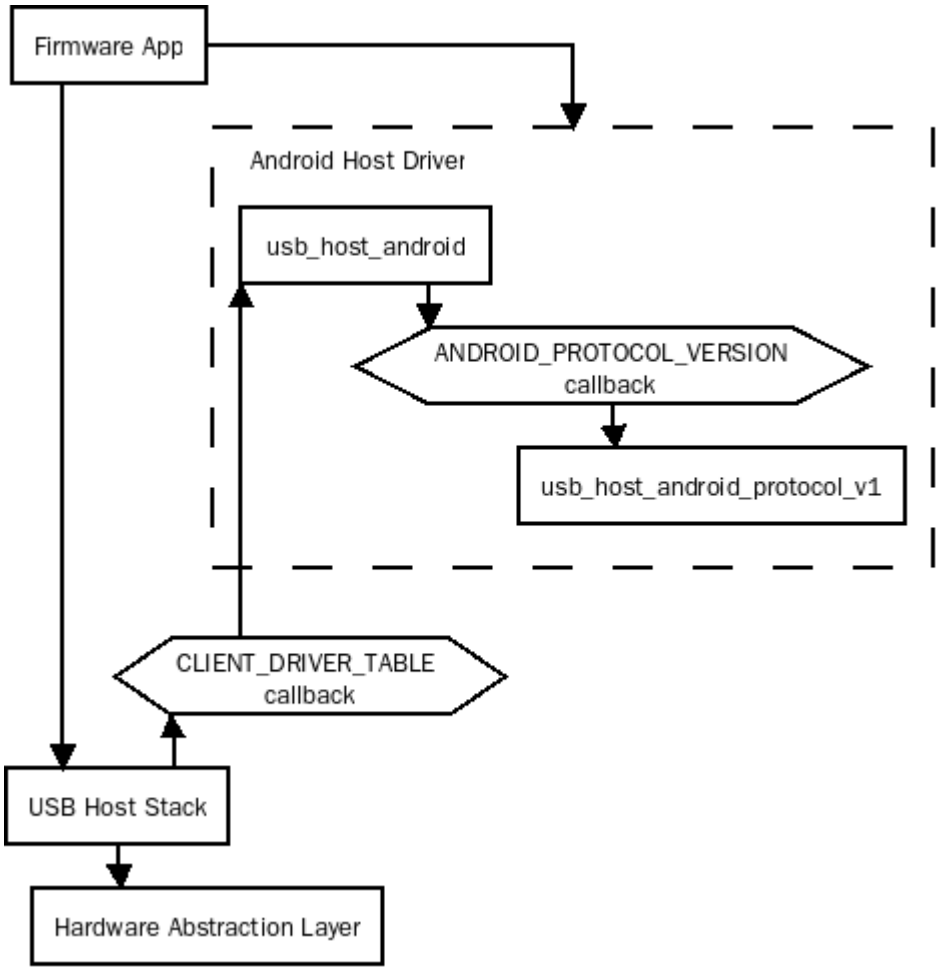
```
/* package com.android.future.usb.UsbManager */

public ParcelFileDescriptor openAccessory(UsbAccessory accessory) {
        return mService.openAccessory(...);
…
}

/* package com.android.server.usb.UsbService */
public ParcelFileDescriptor openAccessory(UsbAccessory accessory) {
…
        return nativeOpenAccessory();    // com_android_server_UsbService call
}
```

Microchip Android Accessory Protocol

Microchip에서 제공하는 **Android Accessory**의 **library Architecture**



library Architecture는 크게 USB host, Android Host Driver를 사용한다.
Android Host Driver는 USB host에서 호출할 함수들을 CLIENT_DRIVER_TABLE변수로 제공한다.

```
<< Include/USB/usb_hsot.h >>

struct  CLIENT_DRIVER_TABLE
{
    USB_CLIENT_INIT          Initialize;     // Initialization routine
    USB_CLIENT_EVENT_HANDLER EventHandler;   // Event routine
    #ifdef USB_HOST_APP_DATA_EVENT_HANDLER
        USB_CLIENT_EVENT_HANDLER DataEventHandler;  // Data Event routine
    #endif
    DWORD                    flags;          // Initialization flags
};
```

```
<< usb_config.c >>

CLIENT_DRIVER_TABLE usbClientDrvTable[] =
{
   {
      AndroidAppInitialize,
      AndroidAppEventHandler,
      AndroidAppDataEventHandler,
      0
   }
};
```

Android Host Driver는 USB Host Event와 Android device정보를 처리하는 usb_host_android와 Android Accessory Protoco versionI에 따른 구현부로 나뉘는데 현재는 usb_host_android_protocol_v1만 제공한다. usb_host_android는 ANDROID_PROTOCOL_VERSION callback을 사용해서 usb_host_android_protocol_v1 를 호출한다.

```
<< usb_host_android.c >>

struct  ANDROID_PROTOCOL_VERSION
{
   BYTE versionNumber;
   ANDROID_APP_WRITE write;
   ANDROID_APP_IS_WRITE_COMPLETE isWriteComplete;
   ANDROID_APP_READ read;
   ANDROID_APP_IS_READ_COMPLETE isReadComplete;
   ANDROID_APP_INITIALIZE init;
   ANDROID_APP_TASKS tasks;
   USB_CLIENT_EVENT_HANDLER handler;
   USB_CLIENT_EVENT_HANDLER dataHandler;
} ;

static ANDROID_PROTOCOL_VERSION protocolVersions[] =
{
   {
      1,
      AndroidAppWrite_Pv1,
      AndroidAppIsWriteComplete_Pv1,
       AndroidAppRead_Pv1,
       AndroidAppIsReadComplete_Pv1,
       AndroidInitialize_Pv1,
       AndroidTasks_Pv1,
       AndroidAppEventHandler_Pv1,
       AndroidAppDataEventHandler_Pv1
   }
};
```

Android Accessory protocol구현

Microchip library는 USB Host stack Tasks를 처리할 때 Android Accessory Protocol Task도 처리한다.

```
<< usb_config.h >>

#define USBTasks()        \
{
   // USB Host 활성화 및 Host task 처리
   USBHostTasks();
   // Android Accessory Protocol 활성화 및 task 처리
    AndroidTasks();
}
```

위의 USBTasks()는 firmware main loop에서 반복적으로 호출되어야 한다.

```
<< main.c >>

int main(void)
{
 …
   while(1)
   {
      …
      USBTasks();
      …
   }
   …
}
```

USBTasks()안에서 호출하는 AndroidTasks()는 Android Accessory Protocol 처리와 Android Accessory driver의 상태값을 변경한다.

USB host에 USB device가 연결되면 USBHostTasks()에서 USB Host stack에 등록한 callback(CLIENT_DRIVER_TABLE)함수를 사용해서 USB host client인 Android Accessory driver의 초기화 루틴을 호출한다.

```
<< usb_host.c >>

void USBHostTasks( void )
{
…
        usbClientDrvTable[temp].Initialize(…); (CLIENT_DRIVER_TABLE  Type →  AndroidAppInitialize
callback)
…
}
```

AndroidAppInitialize()는 Android Accessory driver의 변수(ANDROID_DEVICE_DATA)의 state을 변경한다.

```
<< usb_host_android.c >>

BOOL AndroidAppInitialize(BYTE address, DWORD flags, BYTE clientDriverID)
{
…
        if(devices[i].state == NO_DEVICE)
        {
                devices[i].state = DEVICE_ATTACHED;
                …
        }
…
}
```

firmware main loop에서 USBTasks() → AndroidTasks() 호출되고 state이 DEVICE_ATTACHED이면 AndroidCommandGetProtocol()를 호출해서 USB control request에 accessory protocol의 51값을 연결된 USB device에 전송한다.
전송에 성공하면 변수의 state을 GET_PROTOCOL_SENT로 변경한다.

```
<< usb_host_android.c >>

void AndroidTasks(void)
{
…
        switch(devices[i].state)
        {
                case DEVICE_ATTACHED:
                        …
                        errorCode = AndroidCommandGetProtocol(…);
                        if(errorCode == USB_SUCCESS)
                                devices[i].state = GET_PROTOCOL_SENT;
                        break;
        }
}

BYTE AndroidCommandGetProtocol(ANDROID_DEVICE_DATA* device, WORD *protocol)
{
        //  ANDROID_ACCESSORY_GET_PROTOCOL ← 51
        USBHostIssueDeviceRequest (…, ANDROID_ACCESSORY_GET_PROTOCOL, …);
}
```

USB Host library는 control request를 보내고 USB client로 부터 return message를 받아서 Android Accessory driver의 callback함수(AndroidAppEventHandler)를 호출한다.

```
<< usb_host.c >>

// usb client의 event를 받아서 client driver의 callback을 호출한다.
```

```
void _USB_NotifyClients( BYTE address, USB_EVENT event, void *data, unsigned int size )
{
…
        usbClientDrvTable[((HOST_TRANSFER_DATA *)data)->clientDriver].EventHandler(address, event,
data, size);
…
}

<< usb_host_android.c >>

BOOL AndroidAppEventHandler( BYTE address, USB_EVENT event, void *data, DWORD size )
{
…
        switch (event)
        {
        …
                case EVENT_TRANSFER:
                        …
                        // ANDROID_PROTOCOL_VERSION type ( Android protocal callback )
                        protocolVersions[j].init(…);      ←  AndroidInitialize_Pv1() callback
                        …
                        break;
        …
        }
…
}
```

Callback함수 AndroidInitialize_Pv1()로 Android Accessory protocol을 초기화 한다.
초기화는 ANDROID_PROTOCOL_V1_DEVICE_DATA 변수에 연결된 Android device의 정보를 저장한다.

Android device가 Accessory mode인경우 변수의 state을 RETURN_OF_THE_ACCESSORY로 변경하고
Accessory mode가 아닌경우 변수의 state을 DEVICE_ATTACHED로 변경한다.

```
<< usb_host_android_protocol_v1.c >>

void* AndroidInitialize_Pv1 ( BYTE address, DWORD flags, BYTE clientDriverID )
{
…
    if(tempWord == 0x18D1)
    {
        ReadWORD(&tempWord, &device_descriptor[USB_DEV_DESC_PID_OFFSET]);
        if((tempWord == 0x2D00) || (tempWord == 0x2D01))
        {
            …
            device->state = RETURN_OF_THE_ACCESSORY;
             …
```

```
        }
    }
…
    if(device == NULL)
    {
        …
        device->state = DEVICE_ATTACHED;
        …
    }
…
}
```

firmware main loop에서 USBTasks() → AndroidTasks()는 ANDROID_PROTOCOL_VERSION 구조체에 등록된 callback인 AndroidTasks_Pv1를 호출하고

```
<< usb_host_android.c >>

void AndroidTasks(void)
{
…
          protocolVersions[i].tasks();        // AndroidTasks_Pv1를 호출
…
}
```

AndroidTasks_Pv1()는 ANDROID_PROTOCOL_V1_DEVICE_DATA 변수의 state을 확인해서 Android device가 accessory mode이면 firmware에서 등록한 Event Handler에

```
<< usb_config.h >>

#define USB_HOST_APP_EVENT_HANDLER USB_ApplicationEventHandler
```

EVENT_ANDROID_ATTACH이벤트를 보내고,

```
<< usb_host_android_protocol_v1.c >>

void AndroidTasks_Pv1(void)
{
…
          switch(device->state)
          {
          …
          case RETURN_OF_THE_ACCESSORY:
                    …
                    USB_HOST_APP_EVENT_HANDLER(…,EVENT_ANDROID_ATTACH,…);
                    break;
          …
          }
…
}
```

Accessory mode가 아닐경우는 Android device가 Accessory mode를 시작하도록 id string 를 보낸다.

```
<< usb_host_android_protocol_v1.c >>

void AndroidTasks_Pv1(void)
{
…
        switch(device->state)
        {
                case DEVICE_ATTACHED:
                case SEND_MANUFACTUER_STRING:
                …
                AndroidCommandSendString_Pv1(…, ANDROID_ACCESSORY_STRING_MANUFACTURER, …);
                device->state = SEND_MODEL_STRING;
                break;
                case SEND_MODEL_STRING:
                …
                AndroidCommandSendString_Pv1(…, NDROID_ACCESSORY_STRING_MODEL, …);
                device->state = SEND_DESCRIPTION_STRING;
                break;
                case SEND_DESCRIPTION_STRING:
                …
                AndroidCommandSendString_Pv1(…,ANDROID_ACCESSORY_STRING_DESCRIPTION, …);
                device->state = SEND_VERSION_STRING;
                break;
                case SEND_VERSION_STRING:
                …
                AndroidCommandSendString_Pv1(…,ANDROID_ACCESSORY_STRING_VERSION, …);
                device->state = SEND_URI_STRING;
                break;
                case SEND_URI_STRING:
                …
                AndroidCommandSendString_Pv1(…,ANDROID_ACCESSORY_STRING_URI, …);
                device->state = SEND_SERIAL_STRING;
                break;
                case SEND_SERIAL_STRING:
                …
                AndroidCommandSendString_Pv1(…,ANDROID_ACCESSORY_STRING_SERIAL, …);
                device->state = START_ACCESSORY;
                break;
        …
        }
…
}
```

```
static BYTE AndroidCommandSendString_Pv1(..., ANDROID_ACCESSORY_STRINGS stringType,
                                         const char *string, …)
{
…
        // send id string   ← 52
        USBHostIssueDeviceRequest(..., ANDROID_ACCESSORY_SEND_STRING, …);
…
}
```

모든 id string이 전송되고 다시 irmware main loop에서 USBTasks() → AndroidTasks()는 callback인 AndroidTasks_Pv1를 호출하면, device의 state이 START_ACCESSORY이면 Android device에 Accessory mode start 신호를 보낸다.

```
<< usb_host_android_protocol_v1.c >>

void AndroidTasks_Pv1(void)
{
…
        switch(device->state)
        {
        …
                case START_ACCESSORY:
                …
                AndroidCommandStart_Pv1(device);
                device->state = ACCESSORY_STARTING;
                break;
        …
        }
…
}

static BYTE AndroidCommandStart_Pv1(void *handle)
{
…
        // start accessory mode   ← 53
        USBHostIssueDeviceRequest(..., ANDROID_ACCESSORY_START, …);
…
}
```

Accessory는 Android device의 return event로 USB Host stack에서 AndroidAppEventHandler()가 호출한다.

```
<< usb_host_android.c >>

BOOL AndroidAppEventHandler( BYTE address, USB_EVENT event, void *data, DWORD size )
{
…
        switch(event)
```

```
        {
        …
                case EVENT_TRANSFER:
                        …
                        protocolVersions[j].init(…);        //  ← AndroidInitialize_Pv1() 호출
        …
        }
…
}
```

AndroidInitialize_Pv1()는 VenerID, ProductID가 Android Accessory device가 맞으면 ANDROID_PROTOCOL_V1_DEVICE_DATA 변수 state을 RETURN_OF_THE_ACCESSORY로 변경한다.

```
void* AndroidInitialize_Pv1 ( BYTE address, DWORD flags, BYTE clientDriverID )
{
…
        if(tempWord == 0x18D1)
        {
                ReadWORD(&tempWord, &device_descriptor[USB_DEV_DESC_PID_OFFSET]);
                if((tempWord == 0x2D00) || (tempWord == 0x2D01))
                {
                        for(i=0;i<NUM_ANDROID_DEVICES_SUPPORTED;i++)
                        {
                                if(devices_pv1[i].state == WAITING_FOR_ACCESSORY_RETURN)
                                {
                                        device = &devices_pv1[i];
                                        device->state = RETURN_OF_THE_ACCESSORY;
                                        break;
                                }
                        }
                }
        }
…
}
```

Android Accessory Protocol의 절차가 다 끝나면 Android device는 Accessory mode가 시작되고 Accessory에서 보낸 Id string에 맞는 application을 실행한다.