5 Steps to Creating and Deploying Android™ Data Acquisition Apps

Android™-based tablets and smartphones are growing at a phenomenal rate. You can do so much on them – why not use them for data acquisition applications?

With the Universal Library (UL) for Android API, you can easily create custom data acquisition apps for Android devices. UL for Android communicates with supported Measurement Computing DAQ devices over the Android 3.1 platform (API level 12) and later.

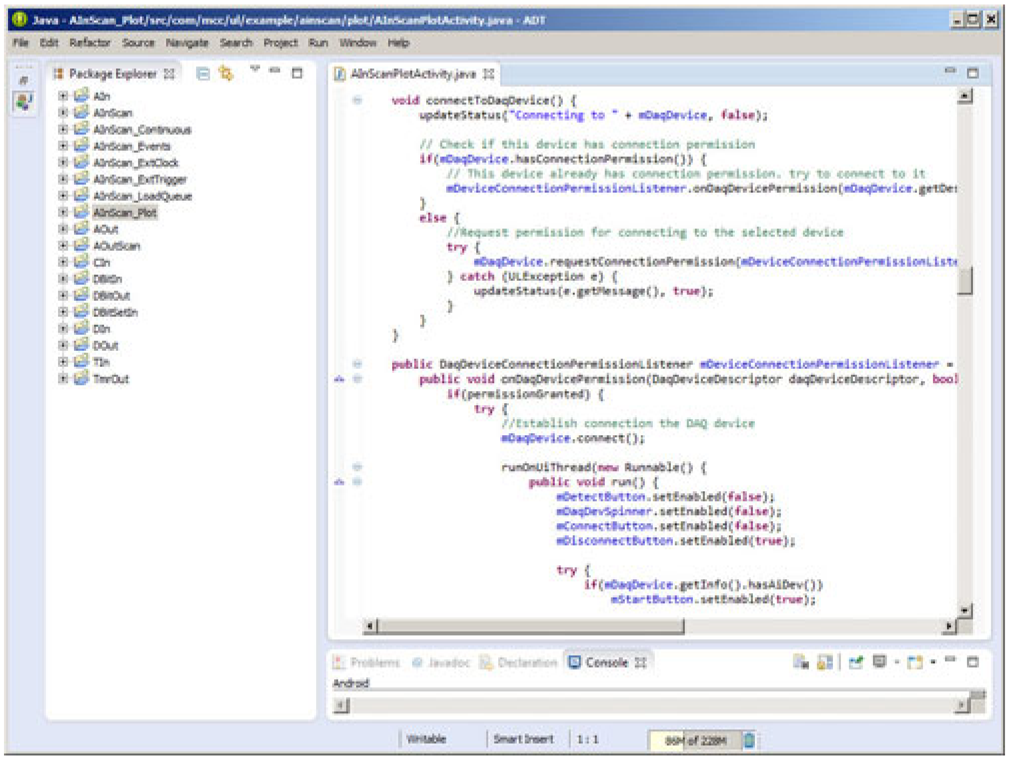
The first data acquisition board supported by UL for Android is the BTH-1208LS (/wireless-data-acquisition/BTH-1208LS.aspx). The BTH-1208LS was designed with tablets and smartphones in mind as it offers reliable and easy-to-set-up Bluetooth connectivity along with rechargeable batteries. Later in 2013, UL for Android will be expanded to support a full line of MCC USB data acquisition devices. For a full list of data acquisition devices supported by UL for Android, please visit http://www.mccdaq.com/ULforAndroid (/daq-software/universal-library-android.aspx).

Step 1 – Download Android SDK
Download the Android Software Development Kit (SDK) for free at http://developer.android.com/sdk/index.html (http://developer.android.com/sdk/index.html).

While most text-based data acquisition programs are created with C#, most Android development is performed in Java. Luckily for C# programmers, the two languages have similar syntax.

Even though the syntax's are similar, if you are unfamiliar with the Android OS and Java programming, you should first study the Android OS features and the Java API.

Step 2 – Download "UL for Android"
Download "UL for Android" for free at http://www.mccdaq.com/ULforAndroid (/daq-software/universal-library-android.aspx).



```
void connectToDaqDevice() {
    updateStatus("Connecting to " + mDaqDevice, false);

    // Check if this device has connection permission
    if(mDaqDevice.hasConnectionPermission()) {
        // This device already has connection permission. try to connect to it
        mDeviceConnectionPermissionListener.onDaqDevicePermission(mDaqDevice.getDe
    }
    else {
        //Request permission for connecting to the selected device
        try {
            mDaqDevice.requestConnectionPermission(mDeviceConnectionPermissionListe
        } catch (ULException e) {
            updateStatus(e.getMessage(), true);
        }
    }
}

public DaqDeviceConnectionPermissionListener mDeviceConnectionPermissionListener =
    public void onDaqDevicePermission(DaqDeviceDescriptor daqDeviceDescriptor, bool
        if(permissionGranted) {
            try {
                //Establish connection the DAQ device
                mDaqDevice.connect();

                runOnUiThread(new Runnable() {
                    public void run() {
                        mDetectButton.setEnabled(false);
                        mDaqDevSpinner.setEnabled(false);
                        mConnectButton.setEnabled(false);
                        mDisconnectButton.setEnabled(true);

                        try {
                            if(mDaqDevice.getInfo().hasAiDev())
                                mStartButton.setEnabled(true);
```

Before you begin programming your DAQ application, it is important to understand the classes that you need to work with to communicate with MCC data acquisition products. The following table describes the most important Universal Library for Android classes. For a complete list and detailed description of the API, please consult the "UL for Android" help file.

Class	Description
DaqDeviceManager	Allows you to detect and create DAQ devices.
DaqDevice	Represents a DAQ device and contains methods to access its I/O subsystems, identifying information, and configurations.

AiDevice	Represents an analog input subsystem on a DAQ Device, and contains analog input methods such as aIn() and aInScan(). This class also contains methods to access AI subsystem information and configuration.
AoDevice	Represents an analog output subsystem on a DAQ device, and contains analog output methods such as aOut() and aOutScan(). This class also contains methods to access AO subsystem information and configuration.
DioDevice	Represents a digital I/O subsystem on a DAQ device, and contains digital I/O methods such as dIn() and dOut(). This class also contains methods to access DIO subsystem information and configuration.
CioDevice	Represents a counter I/O subsystem on a DAQ device, and contains counter I/O methods such as cIn() and cClear() This class also contains methods to access CIO subsystem information and configuration.
TmrDevice	Represents a timer I/O subsystem on a DAQ device, and contains timer I/O methods such as tmrOutStart() and tmrOutStop(). This class also contains methods to access Tmr subsystem information and configuration.

Step 3 – Download Android Example Projects
MCC provides an array of examples to get you up a running quickly. The examples, listed in the table below, offer a variety of DAQ functionality. The example projects can be compiled into downloadable apps.

Some of the examples below are also available as ready-to-run demo apps on Google Play (https://play.google.com/store/search?q=mccdaq&c=apps&hl=en)™.

Example Project Name	**Description**
AIn	Reads an A/D input channel.
AIn_Log	Reads a range of A/D input channels at a specified rate, and stores the acquired data in a .csv file.
AInScan	Scans a range of A/D input channels and stores the sample data in an array.
AInScan_Continuous	Scans a range of A/D input channels continuously in the background and stores the data in an array.
AInScan_Events	Scans D/A channels, displays the latest sample acquired every EventSize or more samples, and updates the latest sample upon scan completion or end.
AInScan_ExtClock	Scans a range of A/D input channels and stores the sample data in an array at a sample rate specified by an external clock.

AInScan_ExtTrigger	Selects the trigger source used to initiate the A/D conversion using DaqDevice.aInScan() with the AiScanOption.EXTTRIGGER option.
AInScan_LoadQueue	Prepares a channel gain queue and loads it to the board. An analog input function demonstrates how the queue values work.
AInScan_Plot	Scans a range of A/D input channels continuously in the background, and plots the latest acquired samples.
AOut	Writes a value to a specified D/A output channel.
AOutScan	Writes values to a specified D/A output channel.
CIn	Resets and reads the event counter.
DBitIn	Reads the status of single digital input bit.
DBitOut	Sets the state of a single digital output bit.
DBitSetIn	Configures the selected port for input, if necessary, then reads and displays the value on the port.
DIn	Reads a digital input port.
DOut	Writes a value to a specified digital output port.
TIn	Reads a temperature channel and displays the value.
TmrOut	Sends a frequency output to a specified timer.

Step 4 – Import Example Source Code into Eclipse
Eclipse is an Integrated Development Environment (IDE) that is included with the Android SDK (ADT Bundle). The steps below will import the "UL for Android" example project into the Eclipse workspace:

- Run Eclipse, choose a workspace folder, and click OK. To create a new folder, enter a name in the Workspace textbox.
- Select File»New»Project. The New Project dialog box opens.
- Expand the Android folder, select Android Project from Existing Code, and click Next. . The Import Projects dialog box opens.
- Click Browse and navigate to the folder containing the example projects.
- Select the Copy projects into workspace checkbox. . When this checkbox is selected, the projects are copied into the current workspace. This is useful if you want to maintain a copy of the example project that is separate from the source file. If you do not select this checkbox, any modifications to the example project are made to the source file.
- Click Finish.
- Select Window»Show View»Package Explorer to open the example in the Package Explorer.

Step 5 – Deploying and Running the Example App on a Connected Device
Before you can run the example apps, you must set up the device as follows:

- Enable USB debugging on the Android device. Refer to the device documentation for more information about how to do this.

- If using a Bluetooth data acquisition device, make sure that the device is already paired with the host Android system.

The steps below will complete the process to deploy and run the app on an Android tablet or smartphone.

- From the Package Explorer, select the folder containing the project to run.
- Select Run»Run and choose Android Application from the Run As dialog box that opens; click OK. Eclipse installs the example app on the connected device; the example automatically opens when installation is complete.