DARRYN CAMPBELL
Mobile computing and enterprise software development

Tutorial: Scan with Datawedge Intent output on Zebra devices (with Xamarin)

16th January 2018 0 By DARRYNCAMPBELL

This tutorial will take you through scanning barcode data via Android Intents using DataWedge on Zebra devices with a Xamarin application.

An earlier tutorial walked through how to do this with a native application but this tutorial will go through a Xamarin application.

There is a dedicated Xamarin component for Zebra devices for API access which can give greater control than DataWedge but may be overkill for many applications.

DataWedge configuration

This step is common with the previous tutorial for native applications

The first step is to configure the DataWedge service with an INPUT (the barcode scanner) and an OUTPUT (send an Intent)

Launch the DataWedge application on the device, there is no need to install anything as it comes pre-installed on all Zebra Android mobile devices.

Select the profile which will be associated with your application, unless you configure a separate profile then DataWedge will use “Profile0 (default)”

Ensure:

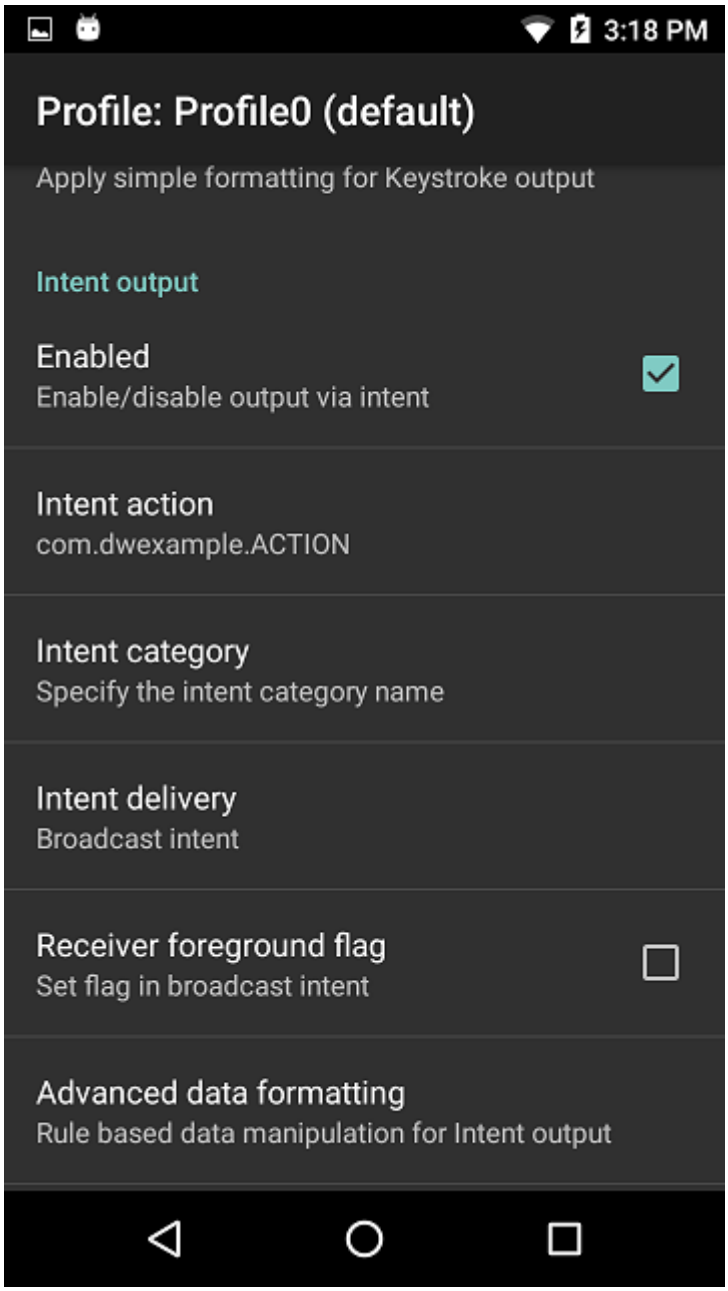
- The profile is enabled
- Barcode input is enabled
- Intent output is enabled
- All other inputs and outputs can be disabled.

Configure the Intent Output as follows (as shown in the screenshot below):

- Intent action: This is an **implicit** intent that will be sent by DataWedge, it is up to your application to ensure it is configured to receive this intent. For the purposes of this tutorial, specify com.dwexample.ACTION.

- Intent category: The category that is associated with the intent sent by DataWedge following each scan. Leave this blank for this tutorial.
- Intent delivery, One of
  - “Send via StartActivity”, analogous to calling Context.startActivity
  - “Send via StartService”, analogous to calling Context.startService
  - “Broadcast intent”, analogous to calling Context.sendBroadcast. For this tutorial, select ‘Broadcast intent”.

For the tutorial, your Intent output should match the screenshot below.



The application

Moving over to the application now, there are several key parts to ensure we are able to receive the intent data that DataWedge is sending.

First, we can pre-define some of the strings to make it easier to receive and extract the scanned data. The intent’s action is defined as “com.dwexample.ACTION” and once we receive an intent it will contain extras representing the scanned data for source, type and data as listed in the official docs

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">Datawedge Intent Example 3</string>
  <string name="activity_intent_filter_action">
        com.dwexample.ACTION</string>
  <string name="datawedge_intent_key_source">
        com.symbol.datawedge.source</string>
  <string name="datawedge_intent_key_label_type">
        com.symbol.datawedge.label_type</string>
  <string name="datawedge_intent_key_data">
        com.symbol.datawedge.data_string</string>
</resources>
```

Because we configured DataWedge to send a broadcast intent our application must now register a broadcast receiver.

> *Obviously if we had configured DW to start an activity we could register for that in our manifest and call getIntent() in onCreate() or if we had configured it as start service we could create a service to receive the intent.*

For this example we will register a dynamic broadcast receiver in the onCreate() call of our application. If you were doing this in a production application you would more likely register / unregister in the onResume() / onPause().

Note that the action we are filtering on matches the action that we configured the DataWedge service to send.

```
protected override void OnResume()
{
    base.OnResume();
    //  Register the broadcast receiver dynamically
    RegisterReceiver(receiver,
        new IntentFilter(Resources.GetString(
        Resource.String.activity_intent_filter_action)));
}
```

Having registered a broadcast receiver we had better define one. You could do this in a separate class but for this tutorial you can just define a receiver within your MainActivity.cs

```
//  Broadcast receiver to receive our scanned data from Datawedge
[BroadcastReceiver(Enabled = true)]
public class myBroadcastReceiver : BroadcastReceiver
{
    public override void OnReceive(Context context, Intent intent)
    {
        String action = intent.Action;
        if (action.Equals(MainActivity.Instance.Resources.GetString(
            Resource.String.activity_intent_filter_action)))
        {
            //  A barcode has been scanned
            MainActivity.Instance.RunOnUiThread(() =>
                MainActivity.Instance.DisplayResult(intent));
        }
    }
}
```

Plumb together the broadcast receiver and the main activity in the onCreate() method using an instance variable

```
//  Instance used to communicate from broadcast receiver back to main activity
public static MainActivity Instance;
```

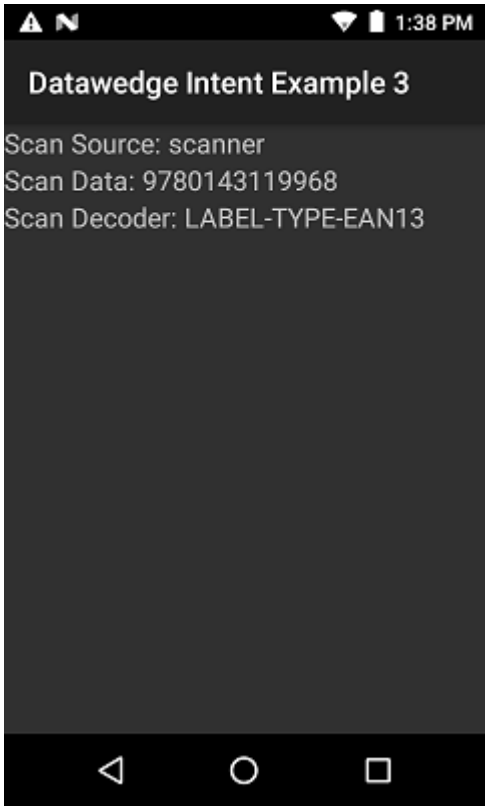
```
3.  myBroadcastReceiver receiver;
4.
5.  protected override void OnCreate(Bundle savedInstanceState)
6.  {
7.      base.OnCreate(savedInstanceState);
8.      MainActivity.Instance = this;
9.      receiver = new myBroadcastReceiver();
10.
11.     // Set our view from the "main" layout resource
12.     SetContentView(Resource.Layout.Main);
13. }
```

The logic of extracting the scanned data and displaying it on the screen is handled by its own method. Note that the extra keys were defined earlier in the strings.xml file. The below code assumes a UI exists in which to place the data but a production application will be driving a lot of its behaviour following a scan.

```
1.  public void DisplayResult(Intent intent)
2.  {
3.      //  Output the scanned barcode on the screen.  Bear in mind older JB devices
    will use the legacy DW parameters on unbranded devices.
4.      String decodedSource = intent.GetStringExtra(Resources.GetString(
5.          Resource.String.datawedge_intent_key_source));
6.      String decodedData = intent.GetStringExtra(Resources.GetString(
7.          Resource.String.datawedge_intent_key_data));
8.      String decodedLabelType = intent.GetStringExtra(Resources.GetString(
9.          Resource.String.datawedge_intent_key_label_type));
10.
11.     TextView scanSourceTxt = FindViewById<TextView>
12.         (Resource.Id.txtScanScource);
13.     TextView scanDataTxt = FindViewById<TextView>
14.         (Resource.Id.txtScanData);
15.     TextView scanLabelTypeTxt = FindViewById<TextView>
16.         (Resource.Id.txtScanDecoder);
17.     scanSourceTxt.Text = "Scan Source: " + decodedSource;
18.     scanDataTxt.Text = "Scan Data: " + decodedData;
19.     scanLabelTypeTxt.Text = "Scan Decoder: " + decodedLabelType;
20. }
```

Those are all of the key points to receive scanned data via intent, the flexibility of both Android and DataWedge allow for many other possible configurations for receiving data but the above should be a good jumping off point.

There is a sample application that accompanies this blog and shows all the code at https://github.com/darryncampbell/DataWedge-Intent-Example-3



Share this:

Related

Tutorial: Scan with Datawedge Intent output on Zebra devices
13th December 2017
In "Zebra Technologies"

Tutorial: The DataWedge Intent API (with Xamarin)
10th April 2018
In "Zebra Technologies"

Controlling DataWedge via Key Presses
7th July 2017
In "Zebra Technologies"

Category *Zebra Technologies*

Tags *Android* *DataWedge* *Tutorial* *Xamarin*

Leave a Reply

Your email address will not be published. Required fields are marked *