DARRYN CAMPBELL
Mobile computing and enterprise software development

Tutorial: Scan with Datawedge Intent output on Zebra devices

13th December 2017 4 By DARRYNCAMPBELL

This tutorial will take you through scanning barcode data via Android Intents using DataWedge on Zebra devices

DataWedge configuration

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)”

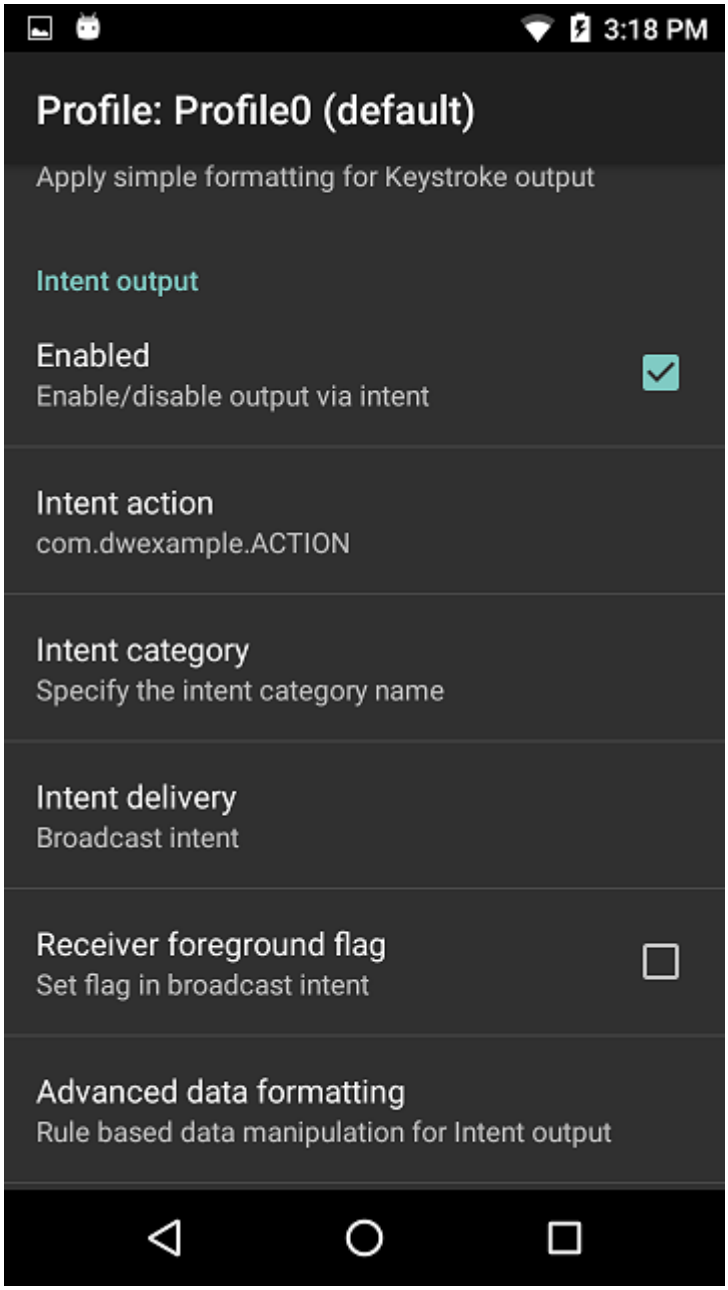
Ensure:

- 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 "dw_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

```
<resources>
    <string name="dw_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.

```
1.  @Override
2.  protected void onCreate(Bundle savedInstanceState) {
3.      ...
4.      IntentFilter filter = new IntentFilter();
5.      filter.addCategory(Intent.CATEGORY_DEFAULT);
6.      filter.addAction(getResources().getString(R.string.dw_action));
7.      registerReceiver(myBroadcastReceiver, filter);
8.  }
```

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.java

```
1.  private BroadcastReceiver myBroadcastReceiver = new BroadcastReceiver() {
2.      @Override
3.      public void onReceive(Context context, Intent intent) {
4.          String action = intent.getAction();
5.          if (action.equals(getResources().
6.            getString(R.string.dw_action))) {
7.              displayScanResult(intent);
8.          }
9.      }
10. };
```

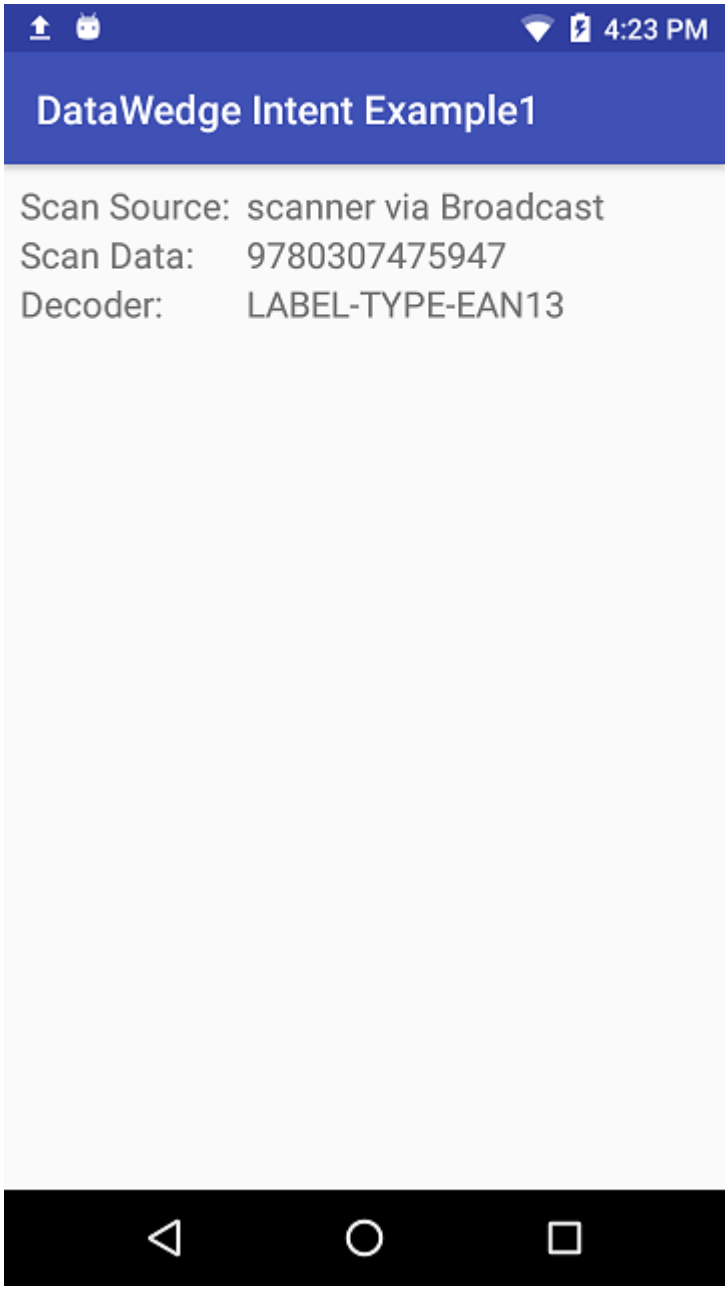
The logic of extracting the scanned data and displaying it on the screen is handled by its own method. Note that the extra keys were defined earlier in the strings.xml file. The below code assumes a UI exists in which to place the data but a production application will be driving a lot of its behaviour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

```
1.  private void displayScanResult(Intent initiatingIntent)
2.  {
3.      String decodedSource = initiatingIntent.getStringExtra(
4.        getResources().getString(R.string.datawedge_intent_key_source));
5.      String decodedData = initiatingIntent.getStringExtra(
6.        getResources().getString(R.string.datawedge_intent_key_data));
7.      String decodedLabelType = initiatingIntent.getStringExtra(
8.        getResources().getString(R.string.datawedge_intent_key_label_type));
9.
10.     final TextView lblScanSource = (TextView)
11.       findViewById(R.id.lblScanSource);
12.     final TextView lblScanData = (TextView)
13.       findViewById(R.id.lblScanData);
14.     final TextView lblScanLabelType = (TextView)
15.       findViewById(R.id.lblScanDecoder);
16.     lblScanSource.setText(decodedSource);
17.     lblScanData.setText(decodedData);
18.     lblScanLabelType.setText(decodedLabelType);
```

```
19.     }
```

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-1



Next Steps

For a better understanding of DataWedge concepts such as Profiles, different input plugins e.g. SimulScan and different decode modes such as UDI then please consult the DataWedge documentation. There is also an **unrelated** Intent API exposed by DataWedge.

For a better understanding of Android Intents and Intent filters, please consult the Android documentation.

Share this:

Related

Tutorial: Scan with Datawedge Intent output on Zebra devices (with Xamarin)
16th January 2018
In "Zebra Technologies"

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

Tutorial: The DataWedge Intent API
3rd January 2018
In "Zebra Technologies"

Category Zebra Technologies

Tags Android DataWedge Tutorial

4 Comments

Cline Newmann says:

1st March 2018 at 3:27 pm

Hi there,
thanks for this tutorial. I still have a question.
I try to figure out how to create a profile within my Java/XML code.
Let say, that my device is not with me and i dont want the user to configure anything on the DataWedge App. Is it possible to write a code in my application that automaticly does that? If yes, who can i do that?

Regards

Reply

darryncampbell says:

6th March 2018 at 9:08 am

Hi, sorry for the belated reply but yes, you can create a profile from within your Java code using the Zebra EMDK Data Capture profile: http://techdocs.zebra.com/emdk-for-android/6-8/guide/profile-manager-guides/ . There is a sample app at http://techdocs.zebra.com/emdk-for-android/6-8/samples/data-capture/. The sample app does not make it obvious but beneath the covers it is creating and configuring a Datawedge profile. There are also separate ways to do this via the Datawedge Intent API and EMDK for Xamarin APIs but since you asked about Java I only linked to the relevant bits.

Reply

Igor says:

14th March 2019 at 4:33 pm

Actually i was testing the following example in a blank app and all was working fine.
The issue comes when i've tryed to implement it in my yet functional app build for Honeywell devices and the output was that for the first 3 debugs all worked fine but then seems that the broadcast is not even registered and not giving any output.

What would be the issue of that?
It is possible that Honeywell library interferes with zebra intent?

Reply

darryncampbell says:

17th March 2019 at 10:23 pm

Hi, I am only able to test this application with Zebra devices I'm afraid as I do not have access to Honeywell hardware. I have not heard that Honeywell have copied the Zebra scanning model – though possible, it seems unlikely that an app would work on both OEM hardware without modification since the Zebra Intent extra key contains the division name ('Symbol')

Reply

Leave a Reply

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

Comment

Name *

Email *

Website

Notify me of follow-up comments by email.