DARRYN CAMPBELL
Mobile computing and enterprise software development

Controlling DataWedge via Key Presses

7th July 2017 0 By DARRYNCAMPBELL

Typically, your application does not need to handle the hardware scanner trigger, the underlying scanner framework will intercept any trigger presses and depending on the scanner engine, will emit a laser or initiate the imager.

In a small minority of cases you may find it necessary to control the scanner beam using a key press, for example if you are migrating your application from a non-zebra device where the application was required to handle the trigger press itself. This is not the standard behaviour of Zebra scanners but it is possible to work around the default behaviour to control the scanner via a simulated key press.

Let us assume the following:

- We have an existing application running on non-Zebra hardware which opens and closes the scanner when the user presses the 'F12' key
- We want to move this application to Zebra devices with only a few changes to our application. How can we do this?

To get the application running unchanged on Zebra hardware we can use a combination of Key mapping and Zebra's DataWedge utility:

1. Remap the scan trigger key to ' F12' , so whenever the user presses the trigger the application will see this as ' F12'
2. Define a DataWedge profile which will be in effect when the user application is in the foreground. This profile will have the barcode scanner input plugin enabled. How barcode data is sent to the application will depend on how the application expects the data, you might choose to send the scanned data as key strokes or Android intents if your application is already listening for them
3. When the application receives an ' F12' key down it will use the DataWedge Intent API to start the scanner. The scanner is stopped when an ' F12' key up is detected.

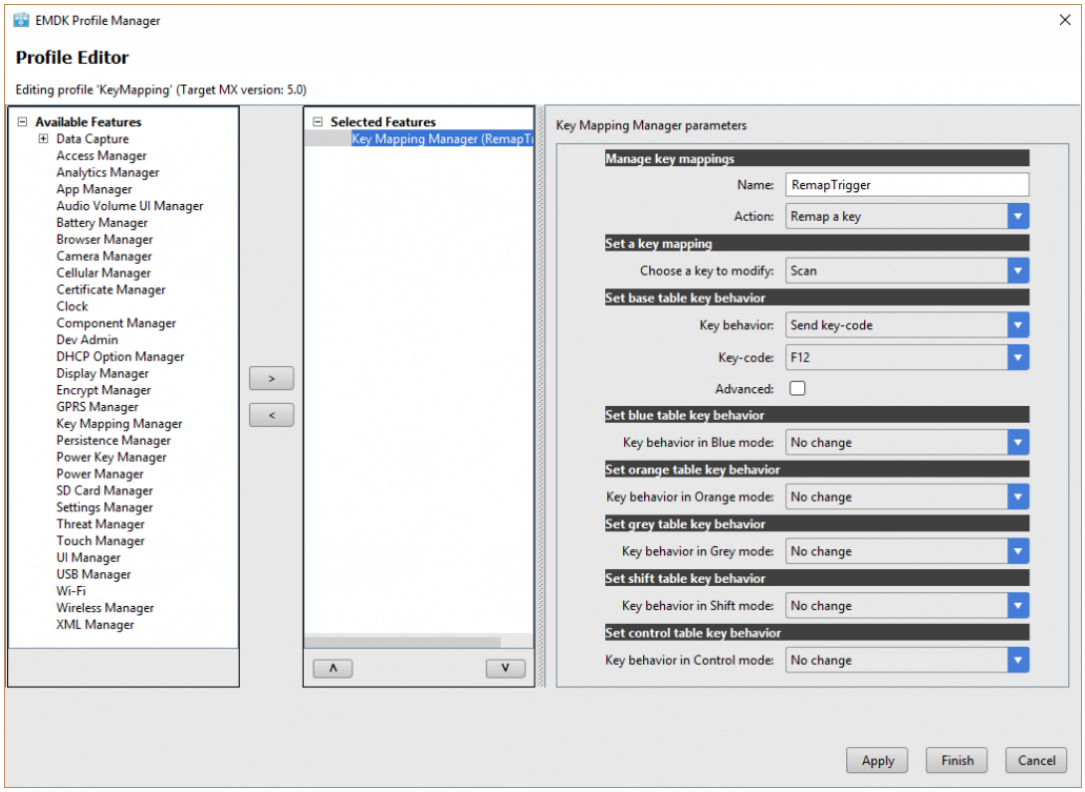
Remap the scan trigger key

It does not have to be 'F12' of course, that is just the example we are working with and on Zebra devices you can remap the scan key using the Key Mapping Manager. Bear in mind many devices will have

custom key identifiers for the key triggers so it is worth reading the documentation and understanding the behaviour for your device.

You can invoke the key mapping manager either via StageNow during staging or via an API through the EMDK.

If you are remapping the key via the EMDK you could define a profile as shown below which remaps the scan key to F12:



Example profile showing remapping the ‘Scan’ key to ‘F12’

This will produce the following XML (assuming MX5.0):

```
<?xml version="1.0" encoding="UTF-8"?><!--This is an auto generated documen
   <characteristic type="ProfileInfo">
     <parm name="created_wizard_version" value="6.3.0"/>
   </characteristic>
   <characteristic type="Profile">
     <parm name="ProfileName" value="KeyMapping"/>
     <parm name="ModifiedDate" value="2017-07-06 09:55:23"/>
     <parm name="TargetSystemVersion" value="5.0"/>
     <characteristic type="KeyMappingMgr" version="4.4">
       <parm name="emdk_name" value="RemapTrigger"/>
       <parm name="Action" value="1"/>
       <characteristic type="KeyMapping">
         <parm name="KeyIdentifier" value="SCAN"/>
         <characteristic type="BaseTable">
           <parm name="BaseBehavior" value="2"/>
```

```
            <parm name="BaseKeyCode" value="142"/>
          </characteristic>
        </characteristic>
      </characteristic>
    </characteristic>
  </wap-provisioningdoc>
```

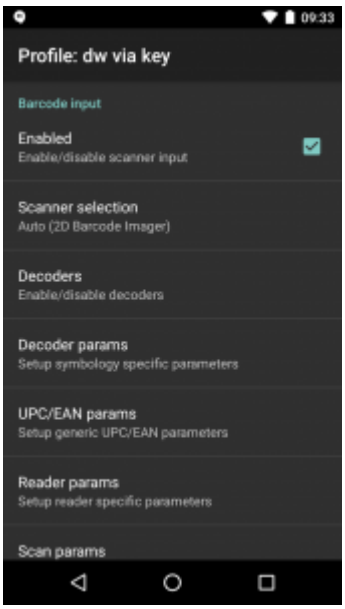
Which can be applied to the device with the processProfile API. That is the link to the Java SDK but you can also call it from Xamarin as well as create the KeyMapping profile with StageNow for mass deployment.

Define a DataWedge profile

We are not doing anything clever here, a DataWedge profile is created with the following specification:

- Associated with our application (so it will come into effect when the application comes to the foreground)
- Has barcode input enabled. Note that because we have remapped the trigger key the scan beam will not emit when we press the trigger by default, we must cause the scanner beam to emit ourselves using the DataWedge API. By enabling barcode input in the profile it allows DataWedge to respond to our API requests to initiate the laser or imager.
- Has some method of output enabled, either key strokes or Android intents. This is how the barcode data is returned to your application and will vary on a case by case basis whichever best fits your use case. If you are already receiving data as key strokes then you can continue doing so, if you have a callback handler in your code then you may wish to add intent processing which calls that handler.

You will end up with a profile like the below



Example DataWedge profile

When the application receives a key down, start the scanner

This will likely require a change to your application but depending on how your application is architected it may only require minimal changes, for example if you are using a plugin to control scanning it may be possible to just modify the plugin rather than the application itself.

When you receive a Key Down or Key Up you need to use the DataWedge API to start or stop the scanner.

The API just uses standard Android intents so send them however best suits your application, this might be through Java, Xamarin, Kotlin, some plugin to a JavaScript framework or some other technique etc.

The example below uses Java:

```
// On Key Down

String action = "com.symbol.datawedge.api.ACTION_SOFTSCANTRIGGER";

String extraKey = "com.symbol.datawedge.api.EXTRA_PARAMETER";

String extraValue = "START_SCANNING";

Intent dwIntent = new Intent();

dwIntent.setAction(action);

dwIntent.putExtra(extraKey, extraValue);

sendBroadcast(dwIntent);


// On Key Up

String action = "com.symbol.datawedge.api.ACTION_SOFTSCANTRIGGER";

String extraKey = "com.symbol.datawedge.api.EXTRA_PARAMETER";

String extraValue = "STOP_SCANNING";

Intent dwIntent = new Intent();

dwIntent.setAction(action);

dwIntent.putExtra(extraKey, extraValue);

sendBroadcast(dwIntent);
```

Conclusion

The technique described in this blog will only be useful in a minority of use cases, wherever possible you should let the scanner framework control the hardware trigger but as stated, occasionally it may be easier to treat the trigger as a standard key.

Related

Instrumented Testing and the Zebra EMDK Barcode API
1st March 2017
In "Zebra Technologies"

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

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

Category Zebra Technologies

Tags Android DataWedge Trigger

Leave a Reply

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

Comment

Name *

Email *