Android Blogs 57 posts

Locking down your device

Posted by Darryn Campbell Apr 10, 2017

Overview

Any deployment of enterprise devices requires some control over which features of the device the user is able to access.
In general, you want to restrict an enterprise user in the following ways:

- Only allow a specific set of named applications to be run.
- Prevent installation or uninstallation of applications
- Prevent access to device settings or prevent changing existing settings e.g. wireless

There may be additional considerations for your enterprise deployment:

- The device may be designed for direct customer interaction (for example a check-in kiosk or audio tour guide)
- You may have a variety of Android versions in your deployment and the solution needs to run across all target devices.
- You may or may not be using a third-party Enterprise Mobility Manager (EMM) to facilitate device provisioning.

Existing solutions

There are a number of solutions to lock down your device, all of which are supported on Zebra devices. This blog will contrast these solutions to enable you to choose the solution that best fits with your deployment:

	Android's APIs and workflows aimed at "Single-Purposes Devices" or "corporate-owned, single use (COSU)" devices. Android has a set of supported APIs and modalities introduced in 5.0 which receive updates with each new Android release. These features can be considered enterprise ready from 6.0 onwards. https://developer.android.com/work/cosu.html
	Enterprise Home Screen (EHS). An application provided free of charge by Zebra; this is a replacement home screen for your device and provides extensive configuration to restrict what the end user is able to do on the device. http://techdocs.zebra.com/ehs/2-5/guide/about/
	Mobility eXtensions (MX). Available on all Zebra devices, MX provides an extensive feature set which includes the ability to control which applications on the device can be run or installed as well as locking down access to device settings. http://techdocs.zebra.com/mx/overview/ http://techdocs.zebra.com/emdk-for-android/6-3/mx/accessmgr/

Clearly there will be some overlap between these offerings. The differences are summarized in the below table:

			
Restrict applications running	Yes, via lock task mode	Yes, by only displaying enabled applications	Yes, by application whitelisting in the AccessManager
Restrict application installation	No, can still install via adb or programmatically through DPC but user ability is restricted as they cannot escape the current app	Not explicitely. Can disable USB to prevent sideloading. Can prevent access to Play store and web to prevent download.	Yes, by application whitelisting
Restrict access to device settings.	Yes, via lock task mode.	Yes, via a configuration setting	Yes, via the AccessManager
Restrict access to settings via the notification shade.	Yes, via lock task mode.	Not on all devices, recommendation is to use MX UiManager instead	Yes, via the UiManager

Kiosk mode support (prevent back, home and recent buttons)	Yes, via lock task mode.	Yes, via a configuration setting	No
GMS / Non-GMS?	GMS Only	Both GMS and non-GMS	Both GMS and non-GMS
Zebra / Non-Zebra?	No restriction	Zebra devcies only	Zebra devcies only
Earliest Android version supported	6.0 for Enterprise use cases	Early releases of EHS supported JellyBean and up	Features supported depend on installed framework version(s). See matrix .
Enhancements in later Android releases	Every version from 5.0 to present (O) includes enterprise enhancements	Latest release (at time of writing) supports KK and up	Additional features are continually being added.
Staging (getting the functionality on device)	Device needs to be configured to have a device owner and associated policy controller. Achieved through NFC bump or QR code	EHS is just an APK. Can be installed via EMM, MX (AppManager), ADB sideload or manually via 'unknown sources'	MX is pre-installed on all recent Zebra mobile computers. Older devices required installation via APK.
Provisioning / Integration with EMM (MDM)	Integrates with all major EMMs.	Can be achieved automatically with code, StageNow or EMMs that support Zebra's MX.	Zebra has a number of EMM partners who support MX and the additional functionality MX provides. See Zebra's device management partner page .
Security of solution	Can only escape lock task mode via an API call in the app that initiated the mode.	Password to escape user mode is encrypted	Whitelisted packages can be associated with a signature to ensure a 'bad app' could not spoof the package name.
Additional benefits	Recommended way forward by Google so benefits from Google support, development and partner ecosystem.	Flexible solution, can be used in conjunction with other lock down offerings e.g. to enhance lockTask mode by disabling USB.	Supported on all Zebra devices going back to JellyBean. Provides a consistent experience without the need for special considerations on each device.
Drawbacks	Only supports GMS devices with Android 6.0+, though applications are still compatible with earlier versions of Android	Locked to Zebra hardware. Does not support widgets	Locked to Zebra hardware, alternatives need to be found if the customer solution contains a mixutre of Zebra and non-Zebra devices.
API access	The DevicePolicyAdmin APIs provide management capabilities to device administrator applications (typically provided by the EMM)	None	Complete API access is provided in Java and C# via the Profile Manager class.
When best to use	New deployments exclusively targeting GMS devices, Marshmallow or greater.	Whenever you need a light weight, set and forget solution with a shallow learning curve.	When you have an exclusively Zebra deployment or need to support Zebra devices going back to JellyBean in a consistent manner.

Android's APIs aimed at "Single-Purposes Devices"

Although Zebra devices have a wide variety of features, for the purposes of Android enterprise they fall under the category of "Single-Purpose Devices" or "corporate-owned, single-use (COSU)" since compared with a consumer smart phone, Zebra devices are generally designed with a task specific purpose in mind.

It is highly recommended to read Android's developer guide for setting up single-purpose devices and the differences between

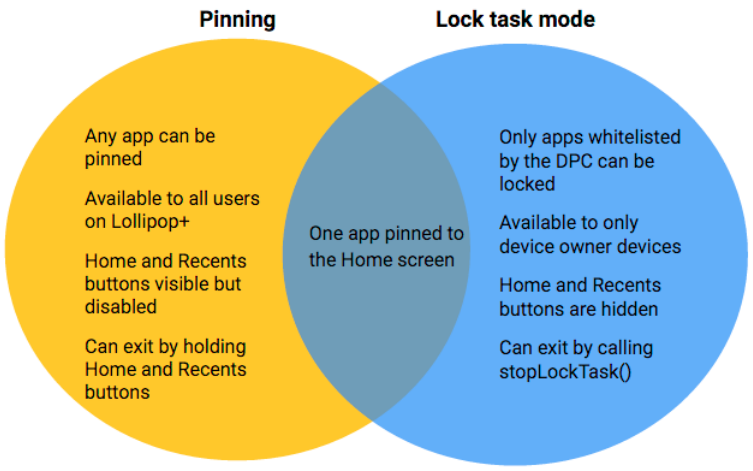
app pinning and lock task mode: https://developer.android.com/work/cosu.html

First, some prerequisites:

- Devices must support Google services. Non-GMS devices (i.e. AOSP only devices) do not support lock task mode.
- The device in question must be configured to have a device owner (DO). Device owners are a type of device policy controller (DPC) as explained here: https://developers.google.com/android/work/overview .
  - GMS Zebra devices running Marshmallow and up support device owner mode.
  - Typically you would NFC bump or scan a QR code to specify a device owner but for testing purposes (including the screenshots & videos in this blog) you can also configure an emulator with an adb command:

```
adb shell dpm set-device-owner "com.afwsamples.testdpc/.DeviceAdminReceiver"
```

Screen pinning vs. Lock task mode:



The Android developer guide does go into more detail but essentially, app pinning targets consumer use cases such as handing a device to a friend to show them a YouTube video without allowing them to see your messages. Lock task mode targets enterprise use cases, is manageable through the DPC and provides a stronger form of device lock down.

The concept of white listing applications is integral to lock task mode as only white listed applications can be shown in and control this mode. White listing is accomplished through an API in the Device Policy Manager which in a real deployment would be controlled through an EMM cloud based interface.

Benefits of lock task mode

- User cannot exit out of your application because the home and ‘recents’ buttons are hidden.
- If there are multiple whitelisted applications, you can switch between them using either the back button or Intents (where one whitelisted application would be acting as an ‘app chooser’)
- The status and menu bar are made invisible and inaccessible, this prevents the user from changing device settings.
- Screen lock and sleep functions are potentially disabled. You may see documentation stating that screen lock and sleep functions are disabled during lock task mode but this depends on how the device owner is managing the device and is configurable from the DevicePolicyManager APIs. It is important to note Google’s documentation for Lock task mode may concentrate on kiosk use cases rather than handheld mobile computers as the use case is more familiar to a wider audience.

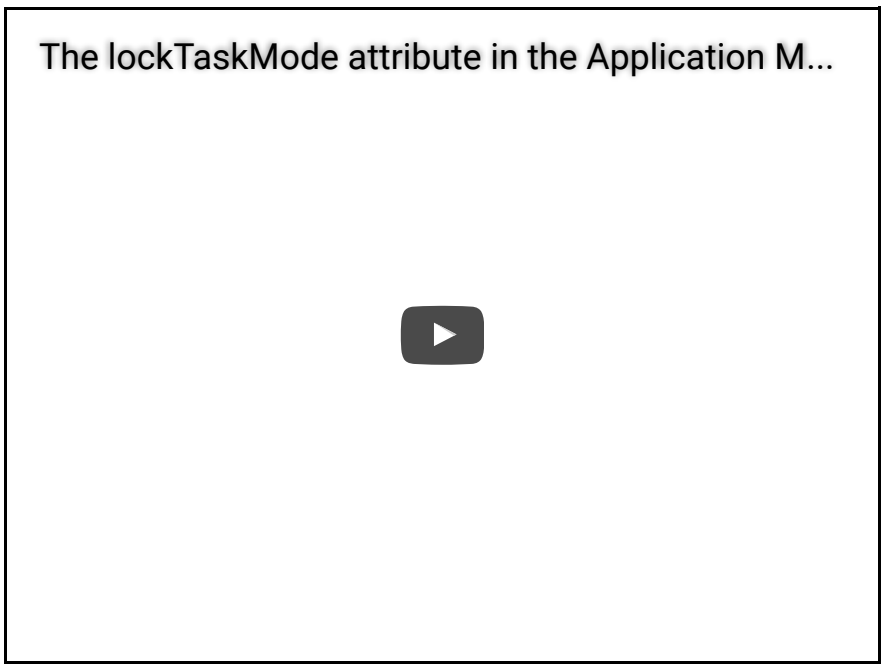
How to enter lock task mode

To demonstrate lock task mode our set up is as follows:

- An emulated device configured with Google services and running Marshmallow.
- The emulated device has been configured to have a device owner, the test dpc provided by Google: https://github.com/googlesamples/android-testdpc

An app that will stand in for the user facing application. Since the application can call startLockTask() and stopLockTask() our test application exposes those as buttons on the interface: https://github.com/darryncampbell/LockTaskMode-Exerciser .

Calling startLockTask() on an application that is n... 	**Base case: Screen pinning** Even running on an Android Marshmallow device with GMS, calling startLockTask() does not automatically enter lock task mode. The application must first be added to the lock task list, controlled by the DPC to whitelist the application. If an application is not whitelisted then calling startLockTask() will just result in application pinning.
Invoking lock task mode from an application 	**Invoking lock task mode from an application** If an application is whitelisted by the DPC, calling the startLockTask() API will cause the device to enter lock task mode. From this mode, the user cannot leave the displayed application. Note: the 'Screen pinned' toasts do not come from the test application, the Android framework is providing misleading information
Entering lock task from the dpc 	**Entering lock task from the DPC.** Calling startLockTask() from the DPC will cause the device to enter lock task mode. From this mode, you can also enter additional whitelisted applications by sending an intent (for the video, this is done via adb, in a real app you would call startActivity()) resulting in a stack of whitelisted applications. Pressing back will cause the stack to unwind and the original (DPC) application to show. Finally, calling stopLockTask() from an application that did not initiate lock task mode will result in a security error; if you call start / stop lockTask in that order from the launched app the stack will again unwind and the DPC app be shown.
Checking the state of lock task mode 	**Checking the state of lock task mode.** There are a couple of APIs which can be used to determine lock task mode. Firstly the DevicePolicyManager has an API to determine whether your application is whitelisted but that must be called from the DPC. The activity also has access to getLockTaskModeState() which returns whether or not the application is currently locked or pinned



The lockTaskMode attribute in the Application Manifest

Although the documentation for lockTaskMode will list 'normal', 'never', 'always' and 'if_whitelisted'; note that 'never' and 'always' are only available to system apps and privileged apps so a typical end user application will only be able to choose between 'normal' and 'if_whitelisted'.

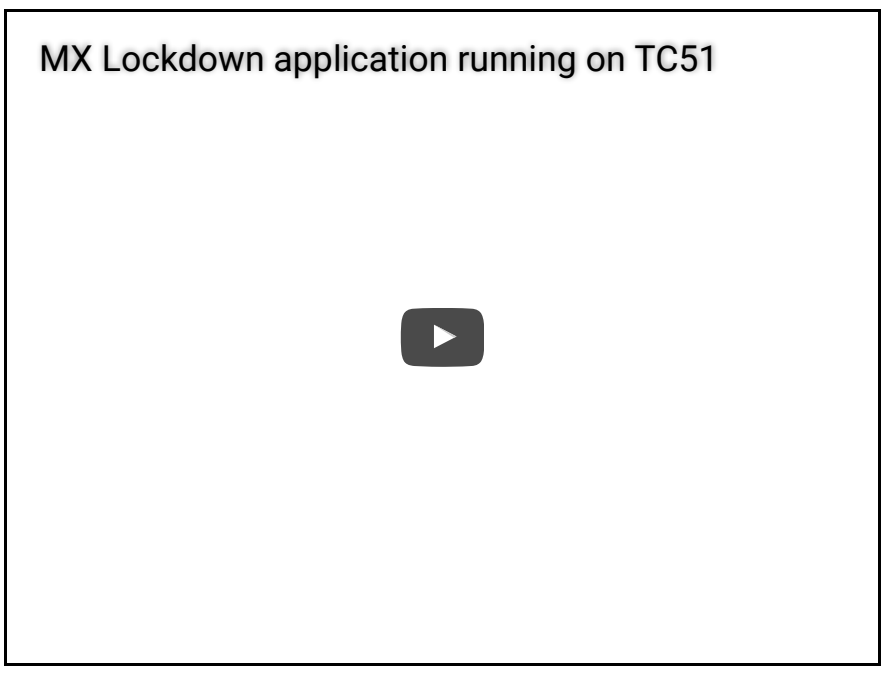
The examples listed so far have shown the effect of 'normal' which is also the default behaviour. If you specify 'if_whitelisted' then the application will launch straight into lock task mode. The video demonstrates an application configured with 'if_whitelisted ' being launched twice, the first time is it not white-listed and the second time it is white-listed.

Mobility eXtensions

Mobile eXtensions (MX) are Zebra's layer of additional functionality added on top of Android to provide an enterprise-class experience for our customers. MX has a wide variety of features designed to facilitate device staging and configuration as well as some application management and data capture capabilities; these features are exposed as 'Profiles' and can be accessed in a variety of ways:

- Through code, supported in both the EMDK for Android and EMDK for Xamarin
- Through StageNow , Zebra's own staging tool which offers integration with several major EMMs.
- Through a supported EMM agent. Several of our EMM partners support MX to enhance their ability to manage Zebra devices.

For the purposes of this blog we are only concerned with profile access only through code or through StageNow since these offer the greatest flexibility in how we control profiles. The examples shown here will use EMDK for Android since it is easier to demonstrate the techniques available but in practice a typical user would most likely be applying these profiles through StageNow.



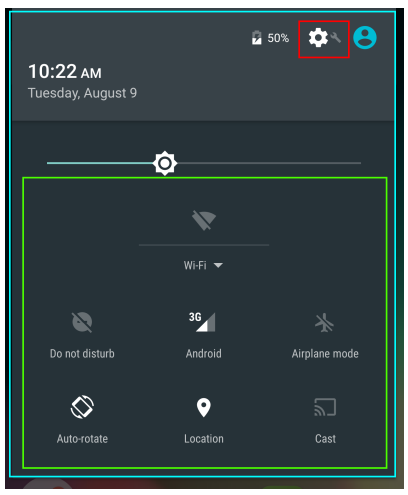
The MX lockdown application, available on GitHub can be used to quickly test the various profiles described here. A video is also available at https://youtu.be/C9OCWsOt4W4 which shows the application running on a Zebra TC51 device

Profiles to lock down your device

The MX profiles offer multiple, often overlapping ways to lock down your device. The most restrictive (in terms of a user's ability to circumvent the lock down) is through the Access Manager with other profiles offering less restrictive but more fine grained options to give the most possible control. You may wish to use some of the less restrictive options in conjunction with Enterprise Home Screen to provide a fully locked down solution.

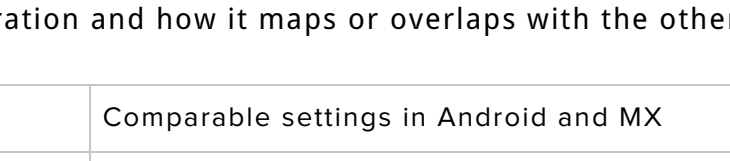
The table below summarizes the lock-down functionality available from the different MX profiles, it is important to note that the profiles themselves are far more functional but I have just pulled out the relevant bits related to locking down your device.

Profile & Functionality	Description

Profile & Functionality	Description
Access Manager • Application whitelisting • Prevent installation of non-whitelisted apps • Prevent running of non-whitelisted apps • Secure which apps can run through use of app signatures • Prevent ability of non-whitelisted apps to invoke other profiles (AKA submit XML) • Prevent access to system settings	This offers device administrators the most control but confusingly shares some terminology with Android's lock task mode. Whitelisting in this context also means specifying which applications can and cannot be run on the device but the Access Manager defines this list through the Add Package Name(s) parameter. Whitelisting applications has no bearing on how applications are launched in contrast with LockTask mode where the user was prevented from exiting back to the launcher, AppManager will however only allow whitelisted apps to be run from the launcher with non-whitelisted apps displaying an error through a Toast. The ability to restrict whether a whitelisted application can or cannot make changes to the system (through profiles, AKA 'Allow XML') offers greater control and fidelity. No special mode is required to use the Access Manager (unlike lock task mode which requires a Device Owner configured on the device) and whitelisting applications via Package signatures, not just the package ID, avoids security concerns on unmanaged devices. Finally, the parameter to prevent access to system settings provides the most comprehensive way to lock down access to device settings on Zebra device short of hiding the settings icon through EHS.
UI Manager • Prevent access to the notification shade • Prevent settings access from the notification shade • Prevent access to the tiles which enable / disable wireless etc. • Prevent Clipboard access	The UI manager offers the most control over which areas of the notification shade are displayed to the user. The following diagram shows the three different areas (blue, red & green) which can be prevented from being shown to the user allowing for example the user to adjust the screen brightness without giving them access to WAN settings, a common use case where employees are working outdoors.  Clipboard access can also be disabled via the UI manager to accommodate use cases where users need to run multiple applications but you do not want them to be able to copy data between apps, for example between a company app and a web browser.

Application Manager
• Disable applications from running
• Prevent access to application *info* settings via the Settings UI
	Beyond disabling specific system applications with known package names, you can also disable certain system services. In the MXLockDown application I use this to disable access to com.google.android.googlequicksearchbox which disables the Google assist function present on GMS devices. This is just an example and you can see all packages installed on your Android device by running the following command:
	`adb shell pm list packages`
	Or see full information about each package by adding the -f flag
	`adb shell pm list packages -f`
	Obviously disabling system packages may have unintended side effects but it is nonetheless a useful tool to have in your arsenal and can be used to restrict which applications the user can "share" content with.
	
	Disabling the settings application entirely can be done through the application manager with the package "com.android.settings". This is a very heavy handed approach however and can lead to side effects such as common actions not working and providing no feedback to the user as to why (e.g. long pressing an application and dragging onto 'App Info') or presenting confusing warnings to the user (e.g. long pressing an application notification gives a 'System UI stopped' warning).

Settings Manager • Prevent access to application settings via the Settings UI	To completely lock down access to settings it is recommended to use the Access Manager. The Settings Manager only prevents the user from navigating to the 'Apps' menu (Settings --> Apps) but the user can still access sub menus directly, for example pressing and holding an application and then dragging over the 'App Info' option at the top of the screen will still display that application's information.

Profile & Functionality	Description
	Taking this a step further, it is possible to lock down the 'App info' screen with the Application Manager's AccessToAppInfo parameter but again, this can be circumvented by accessing the sub menus directly. You can access an application's notification settings directly by pressing and holding on any notification from that app (Android 6.0+) and then pressing the icon (Android 6.0).  Android 7.0 changed this interface so you now click on 'More Settings', but the principle is the same.

MX Version support

Over time, additional features have been added to MX so whilst Zebra does not break backwards compatibility, newer MX features will only be available on newer devices.
For a full list of which profiles are available on which devices consult the compatibility table. You will need to be able to decipher which version of OSX and MX you are running and instructions for that are here .

In general, the major features discussed above will be available on JellyBean devices and up, meaning MX is an ideal solution for any deployment containing a mixture of JellyBean, KitKat, Lollipop or Marshmallow devices where you want a consistent approach to how you lock down your devices.

Enterprise Home Screen

Enterprise Home Screen (EHS) is a free application offered by Zebra to replace your device's default launcher application with an Enterprise-first alternative.
EHS resources including setup and configuration guidance can be found on techdocs and the application itself downloaded from the support page .

There is a lot of overlap between the EHS settings, Android's lock task mode and MX capabilities. The following table lists the available EHS configuration and how it maps or overlaps with the other lock-down options:

EHS Setting	Comparable settings in Android and MX
<kiosk> <preferences> <kiosk_mode_enabled>	Kiosk mode is directly comparable with LockTask mode, in both cases the Home and recent options are disabled and the user is locked out of the rest of the OS. Differences include: • EHS Kiosk mode is designed to run a single application whereas LockTask mode can use multiple applications, therefore the back key is disabled only in EHS's kiosk mode. • Since EHS does not offer a programmatic interface there is no way for a running application to enable or disable Kiosk mode (as you can with Android's stopLockTask() API). EHS allows you to choose the specific activity of an application to be invoked when the app starts whereas LockTask mode is limited to the launch activity as defined in the application manifest.
<applications> <application>	This setting allows you to specify which applications can be run on the device. There are obvious comparisons with the MX ApplicationManager to disable applications, MX AccessManager to whitelist only certain apps or LockTask mode's ability to only whitelist certain apps to run. The difference with EHS's setting is the applications are just being hidden on the launcher so the user is not able to select them, applications can still be started by an Intent even if they are not listed under <applications>. Another difference is in the use of wildcards to specify the package name: no equivalent exists for this in MX or LockTask whitelists but this is by design since the use cases are different.
<applications> <link>	This functionality is specific to EHS as it is a specific use case for a launcher app.
<tools> <application>	This functionality is specific to EHS as it is a value add of that product.
<user_options> <app_icon_size>	This functionality is specific to EHS, it is a value add unrelated to locking down access to the OS.

EHS Setting	Comparable settings in Android and MX
<preferences> <title>	This functionality is specific to EHS, it is a value add unrelated to locking down access to the OS.
<preferences> <title_bar_icon_X>	This functionality is specific to EHS, it is a value add unrelated to locking down access to the OS.
<preferences> <icon_label_background_color>	This functionality is specific to EHS, it is a value add unrelated to locking down access to the OS.
<preferences> <icon_label_text_color>	This functionality is specific to EHS, it is a value add unrelated to locking down access to the OS.
<preferences> <orientation>	You would expect native applications wishing to control the screen orientation to specify a screenOrientation property in their manifest.
<preferences> <auto_launch_enable> <preferences> <service_auto_launch_enable> <auto_launch> <application> <service_auto_launch> <service>	There is no equivalent feature in MX or the Android APIs to launch a set of applications at predefined intervals after application reboot. The closest feature is probably an application's ability to listen for a BOOT_COMPLETED broadcast intent and on receiving that, start an application or service in their broadcast receiver. There is overlap with Kiosk mode, as previously stated EHS Kiosk mode can only lock the user to a single application whereas the auto_launch settings will allow the user to toggle between apps quickly without going back to the launcher. Auto_launch will not lock the user out of the launcher like the EHS kiosk_mode does.
<preferences> <wallpaper>	This functionality is specific to EHS, it is a value add unrelated to locking down access to the OS.
<preferences> <fullscreen>	Counter-intuitively this setting only affects Enterprise Home Screen itself and does not have any effect on apps other than EHS
<preferences> <disable_status_bar_settings_icon>	On JellyBean and KitKat devices this prevents the settings icon being displayed in the Android Status bar, locking down the ability to adjust settings such as Wifi from the pull down. In Android L the notification drop down as reworked and the official EHS documentation recommends using the MX UI Manager to block access to 'quick settings' also known as the 'tiles' you see when swiping down twice. LockTask Mode also prevents the user accessing the notification pull down, duplicating this capability.
<preferences> <disable_status_bar_pulldown>	Similar to disable_status_bar_settings_icon, since the notification pull down received an overhaul in Android between KitKat and Lollipop this setting only works with pre-L devices. EHS documentation for L and above recommends the MX UI Manager to prevent the user pulling down the notification shade. LockTask Mode also prevents the user accessing the notification pull down, duplicating this capability.
<preferences> <install_shortcuts>	This functionality is specific to EHS, it is a value add unrelated to locking down access to the OS.
<preferences> <exit_instead_of_reboot>	This functionality is specific to EHS, it is a value add unrelated to locking down access to the OS.
<preferences> <reboot_on_install_enabled>	This functionality is specific to EHS, it is a value add unrelated to locking down access to the OS.
<preferences> <airplane_option_disabled>	The MX Settings Manager has a parameter which can be set to disable access to the airplane mode selector in the settings UI, quick settings and power key menu. In LockTask mode there is no way for a user to access the settings UI or pull down the notification shade so the lack of a specific setting is a moot point.
<preferences> <bypass_keyguard>	The Android DevicePolicyManager has an API to disable the Keyguard, this API is available to the device owner of a managed device. No equivalent exists in MX.
<preferences> <keyguard_camera_disabled>	The Android DevicePolicyManager has an API to disable features of the keyguard, including the camera. No equivalent exists in MX.

EHS Setting	Comparable settings in Android and MX
<preferences> <keyguard_search_disabled>	The Android DevicePolicyManager has an API to disable features of the keyguard. No equivalent exists in MX.
<preferences> <usb_debugging_disabled>	The MX USB Manager offers a setting to disable USB adb mode. No equivalent exists in Android enterprise APIs unless the device is rooted.
<preferences> <system_settings_restricted>	System_settings_restricted is directly comparable with the MX AccessManager parameter to specify “Reduced Access” to system settings. No equivalent exists in Android enterprise APIs but the user would typically not be able to access the system settings at all in LockTask mode.
<preferences> <apps_disabled><application>	Prevents the application being displayed in both admin and user mode, with some special considerations for settings and search. There are obvious comparisons with the MX ApplicationManager to disable applications, MX AccessManager to whitelist only certain apps or LockTask mode’s ability to only whitelist certain apps to run. Disabling an application in this list truly prevents the app from being run, even via Intent which maps closer to WhiteListing and the MX ApplicationManager than the previously described <applications><application> section.
<preferences> <apps_enabled><application>	This is the opposite of the <apps_disabled> setting. The implementation is a bit more confusing than the single whitelist + special lock mode as seen in both LockTask Mode and MX AccessManager but the end result is the same, you can explicitly enable or disable specific applications. Unlike MX AccessManager where you can provide an app signature or package name, EHS only supports the package name.
<preferences> <admin_max_attempts>	This functionality is specific to EHS, it is a value add unrelated to locking down access to the OS.
<preferences> <admin_inactivity_timeout>	This functionality is specific to EHS, it is a value add unrelated to locking down access to the OS.
Secure Mode	This functionality is specific to EHS and provides an extra layer of security to prevent tampering with the EHS configuration file. This is only required in EHS because it is a post-loaded application configured via an XML file so there are no equivalents in MX or the Android enterprise APIs.

Overlap with MX

It is worth calling out the <disable_status_bar_settings_icon> and <disable_status_bar_pulldown> settings which explicitly recommend to use MX as an alternative on L+ devices. There is also an additional section in the EHS help detailing how MX can be used to enhance EHS (e.g. to prevent screenshots); this illustrates how EHS is designed to work well in conjunction with MX where required.

Conclusion

Controlling what the end user is able to do on your device remains a fundamental use case for enterprise Android deployments and is an area where functionality is continually being added. With Google focussing on this market from Android 6.0 onwards it is likely this area will continue to improve and converge but there will always be the need for Zebra to offer customized capabilities on our devices. Where we do offer customized functionality, Zebra will endeavour to work well with alternative solutions so administrators can pick and choose the desired aspects from each (hopefully that can already be seen above where Android APIs, MX and EHS can all be used together).

184 Views Comments: 0 Permalink

Tags: emdk, enterprise, mx, enterprisehomescreen, lock, locking, zebra_news, device owner, device policy controller

Android Studio 2.3 & EMDK For Android Beta Fix

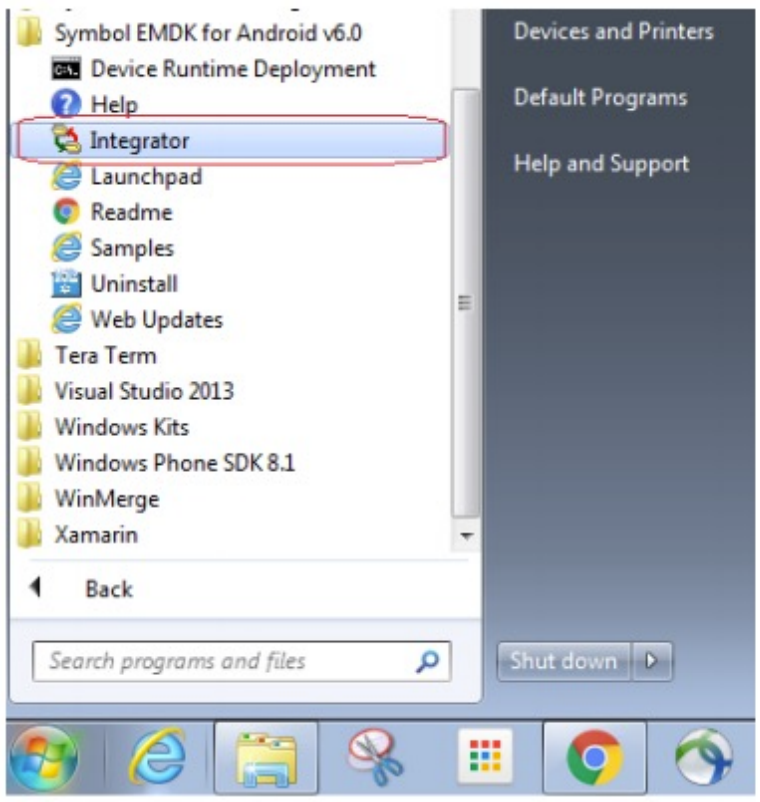
Posted by Rob Galvin Mar 15, 2017

The current version of EMDK For Android 6.x will not work if you have upgraded to the latest Android Studio release 2.3. You may not have noticed this unless you tried to use Profile Manager from the EMDK menu. While we plan on incorporating this into the next official release, we felt it was important to put out an immediate beta release patch to address this now prior to the official release.

PC Installation

1. Install or upgrade the Android Studio v2.3

2. Exit the Android Studio if it is running
3. Install EMDK for Android 6.x. Refer the installation instructions for more details. Remember the EMDK Installation location for later use.
4. Download the EMDK-A-6.x-ProfileManager-Update-For-AndroidStudio2.3-BETA.zip attached to this post
5. Unzip EMDK-A-6.x-ProfileManager-Update-For-AndroidStudio2.3-BETA.zip and copy the com.symbol.emdk.wizard.core.jar & com.symbol.emdk.wizard.intellijIdea.jar to EMDK Installed location i.e., *C:\Program Files (x86)\Symbol EMDK for Android\v6.0 \Integrator\plugins\IntelliJ IDEA\com.symbol.emdk.wizard.intellijIdea\lib*
6. Run the EMDK Integrator from Start Menu as shown below.



7. Restart the Android Studio for your development.

Mac Installation
1. Install or upgrade to Android Studio v2.3
2. Exit Android Studio if it is running
3. Re-Install EMDK for Android 6.x. Refer the installation instructions for more details.
4. Download the EMDK-A-6.x-ProfileManager-Update-For-AndroidStudio2.3-BETA.zip attached to this post.
5. Launch Finder on MAC and go to the Applications folder.
6. Locate Android Studio App, right click and select Show Package Contents.
7. Locate the *Contents\plugins\com.symbol.emdk.wizard.intellijIdea\lib* directory
8. Unzip EMDK-A-6.x-ProfileManager-Update-For-AndroidStudio2.3-BETA.zip and copy the *com.symbol.emdk.wizard.core.jar* & *com.symbol.emdk.wizard.intellijIdea.jar* files into Contents\plugins\com.symbol.emdk.wizard.intellijIdea\lib directory
9. Restart the Android Studio for your development.

337 Views Comments: 0 Permalink Tags: zebra_news

Deploying an application to Zebra Android devices ranging from Jellybean to Marshmallow and beyond

Posted by Darryn Campbell Feb 8, 2017

Overview

Native Android applications can be developed for Zebra mobile Android devices such as the TC51, TC55, TC8000 etc. in a number of ways. The primary API for accessing device hardware (e.g. scanner) and Zebra value-adds (known as 'MX') is the EMDK .The EMDK is being continually being updated to support new devices and features (download link). One drawback of these continual EMDK updates is that support for older devices can be dropped from newer releases, for example at the time of writing EMDK 6.0 is the latest version but can only support devices as far back as KitKat with Jellybean devices requiring development with EMDK 5.0 or below.

The Problem

A developer targeting a deployment with a mixture of old and new devices will not want to develop separate applications for each device OS.Before discussing an EMDK-based solution it is worth noting that all Zebra devices ship with a zero-code data capture solution known as DataWedge . Using DataWedge as your data capture agent and delivering barcode data etc. via Android intents may be sufficient for some applications and negates the problems with a lack of EMDK support for older devices. DataWedge may not be appropriate for your application however if you require fine grained control of the capture hardware or require access to the MX profile layer .The remainder of this guide assumes that your application requires the EMDK and needs to run across a range of

devices from Jellybean to Marshmallow and beyond. This guide supplements the official EMDK documentation and guides for creating enterprise applications, e.g. Creating a project & Creating a profile .

Android Product Flavours

Android build variants provides a solution to the problem as follows:

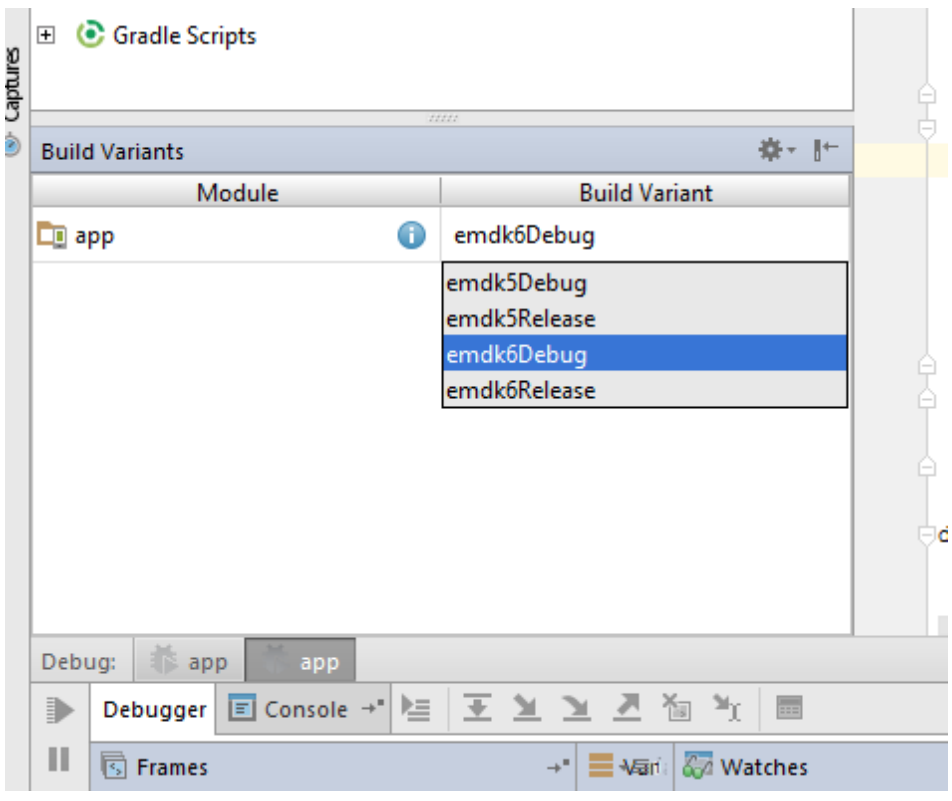
- In the application build.gradle a flavour is defined for each version of the EMDK we require.
  - In this worked example we will be using EMDK 5 to support JellyBean and EMDK 6 to support Marshmallow, Lollipop and KitKat devices. This gives us support for the full range of Android devices but the details may change in the future e.g. we may need to add a new flavour for EMDK 7 or above for Android O or above.
  - EMDK 5 release notes: https://www.zebra.com/content/dam/zebra_new_ia/en-us/software/developer-tools/emdk-for-android/EMDK%20for%20Android%20Version%205.0%20Release%20Notes.pdf
  - EMDK 6 release notes: https://www.zebra.com/content/dam/zebra_new_ia/en-us/software/developer-tools/emdk-for-android/EMDK-for-android-version-6.0-release-notes.pdf
- By defining different source sets we can ensure the correct EMDKConfig.xml (defining our application's MX profiles) will be used
- The output from the build process will be multiple apks but critically the source code remains unchanged between variants, we are just ensuring the correct EMDK library is being used.

Define the product flavours

In the app build.gradle file define product flavours as follows:

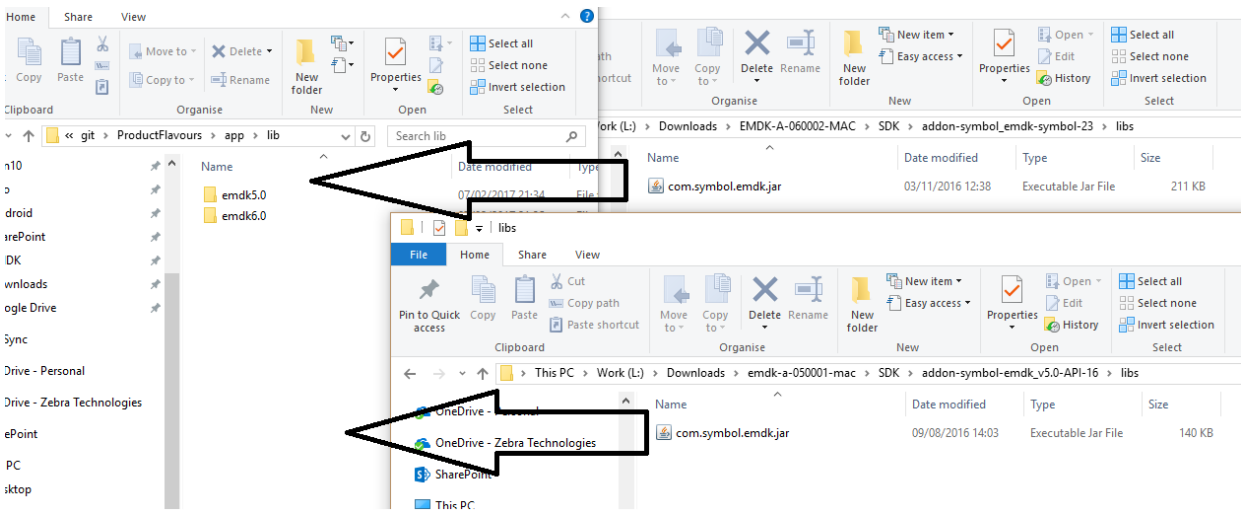
```
android {
  ...
productFlavors {
  emdk6 {
  targetSdkVersion 23
  minSdkVersion 19 // EMDK 6.0 supports as far back as KitKat (19)
  versionCode 30000 + android.defaultConfig.versionCode
  versionNameSuffix "-emdk6"
  }
  emdk5 {
  // Not compatible with Marshmallow devices
  minSdkVersion 16
  targetSdkVersion 16 // Though EMDK 5 actually supported up to Lollipop, we are only using it for JB dep
  maxSdkVersion 22 // Note: Google would still allow installation on 23+ devices (https://developer.andro
  versionCode 75000 + android.defaultConfig.versionCode
  versionNameSuffix "-emdk5"
  }
}
}
```

And sync your gradle project.You can now observe that your Build Variants pane in Android Studio (Build > Select Build Variant) contains entries for each of the defined flavours:



EMDK Installation

You will need to obtain the appropriate EMDK libraries , which in the case of this worked example are EMDK 5.0 and EMDK 6.0. Although the EMDK is currently delivered as a series of EMDK add-ons, you can extract the jar file from the add-on and use the jar directly in your project as a *provided library* and this is required by the next step. I personally find it easier to copy the jar files into my project to avoid installation dependencies. You may find it easier to extract the jar file from the EMDK Mac distribution as this is provided as a zip file.



Referencing the Jar from Gradle

We will be specifying a runtime dependency for our EMDK library similar to what is done in the official documentation when creating an application to run on non-Zebra devices, as explained here .Gradle enables you to specify dependencies as {flavour}Compile, as explained in the gradle docs . The simplest setup that works for us will be to specify the application dependencies as follows, using Provided instead of Compile:

```
dependencies {
// NOTE: This gradle depends on the EMDK Jars associated with EMDK 5 & 6 being in the following directori
  emdk5Provided fileTree(include: ['com.symbol.emdk.jar'], dir: 'lib\\emdk5.0')
  emdk6Provided fileTree(include: ['com.symbol.emdk.jar'], dir: 'lib\\emdk6.0')
  compile fileTree(exclude: ['com.symbol.emdk.jar'], dir: 'libs')
  compile 'com.android.support:appcompat-v7:23.1.0'
}
```

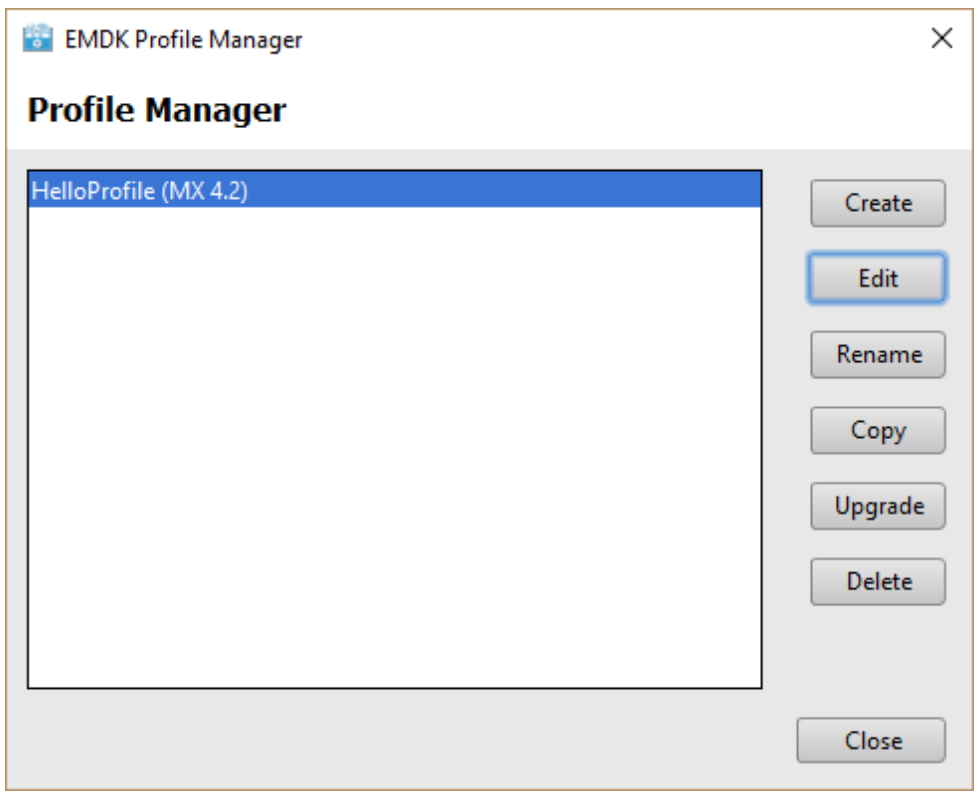
It is also possible to specify separate dependencies for Debug and Release (e.g. emdk5DebugProvided) but in this case that is not required. **Note that the above setup does depend on you having previously copied the EMDK lib jars to your project's 'lib' folder**.

Choosing a build variant

In general, you should choose the API level that matches your deployment target, just as you do when selecting an Android targetSdkVersion. There is nothing technically wrong with running an API 22 application on a Marshmallow (23) device, you just won't get access to Marshmallow features.Our example setup has two build flavours, emdk 5 and emdk6. This was done to enable us to support JellyBean, KitKat, Lollipop and Marshmallow. There is an overlap of supported variants with EMDK 5 supporting JB, KK and L and EMDK 6 supporting KK, L and M. When building for JB we must choose the emdk 5 build flavour, when building for M we must choose the emdk 6 build flavour.

Using Source Sets to define the assets to use

Application interaction with MX Profiles is achieved through the EMDK Profile manager and visually through a plugin to the Android Studio IDE (EMDK > Profile Manager)



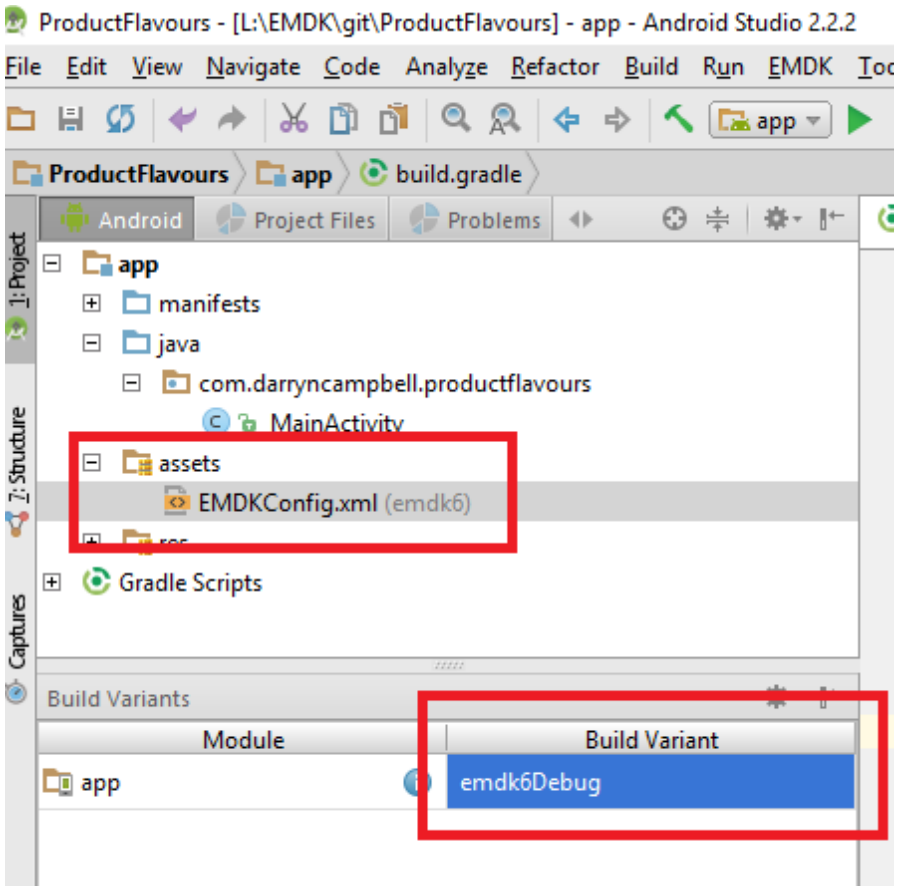
Profiles are defined as XML and stored under <Your app>\src\main\assets\EMDKConfig.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!--This is an auto generated document.
Changes to this document may cause incorrect behavior.-->
<wap-provisioningdoc>
  <characteristic type="ProfileInfo">
  <parm name="created_wizard_version" value="6.0.4"/>
  </characteristic>
```

```
    <characteristic type="Profile">
    <parm name="ProfileName" value="HelloProfile"/>
    <parm name="ModifiedDate" value="2017-01-16 10:00:18"/>
    <parm name="TargetSystemVersion" value="4.2"/>
    <characteristic type="Wi-Fi" version="2.7">
    <parm name="emdk_name" value="DisableWifi"/>
    <characteristic type="System">
    <parm name="WiFiAction" value="disable"/>
    </characteristic>
    <parm name="UseRegulatory" value="0"/>
    <parm name="UseAdvancedOptions" value="0"/>
    </characteristic>
    </characteristic>
</wap-provisioningdoc>
```

The version of MX resident on the device will have a number of capabilities with additional capabilities being added in each MX release. Newer capabilities will not be available to older devices. A full compatibility table is available from this link but for the purposes of this document we will assume that the desired profiles could differ between Jellybean, KitKat, Lollipop and Marshmallow devices. In practice since each release of EMDK supports multiple Android dessert letters you would not have separate XML files for each Android letter.Android uses the concept of Source Sets to define which source files are used for each of the product flavours. Whilst this principle can apply to java files, manifest files, aidl, jni etc we are most interested for this purpose in specifying different assets for each of the product flavours.Modified from the Android docs :

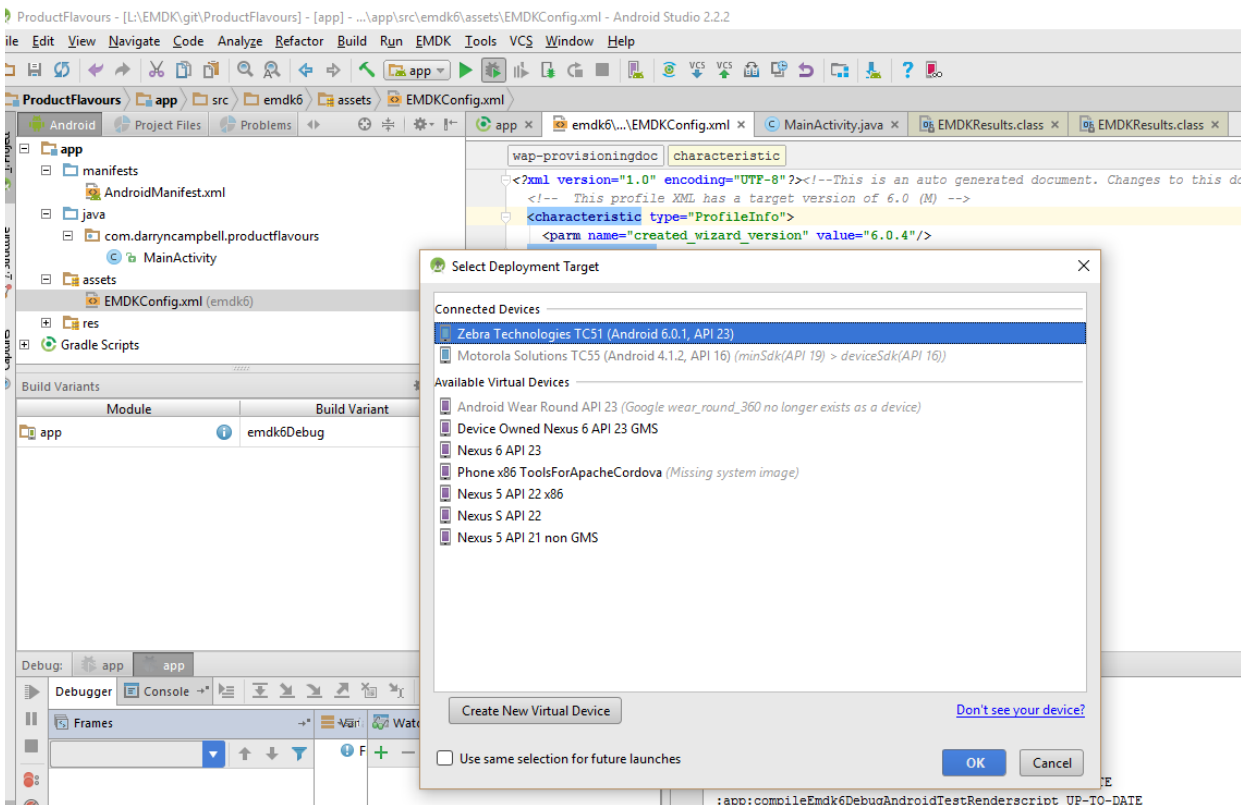
- Open the **Project** pane and select the **Project** view from the drop-down menu at the top of the pane.
- Navigate to MyProject/app/src/.
- Right-click the src directory and select **New** > **Folder** > **Assets Folder**. Note: You may need to untick the setting 'compact empty middle packages'
- From the drop-down menu next to **Target Source Set**, select **emdk6**
- Click **Finish**
- Add the required EMDKConfig.xml file to the newly created folder under <Your application>\app\src\emdk6\assets

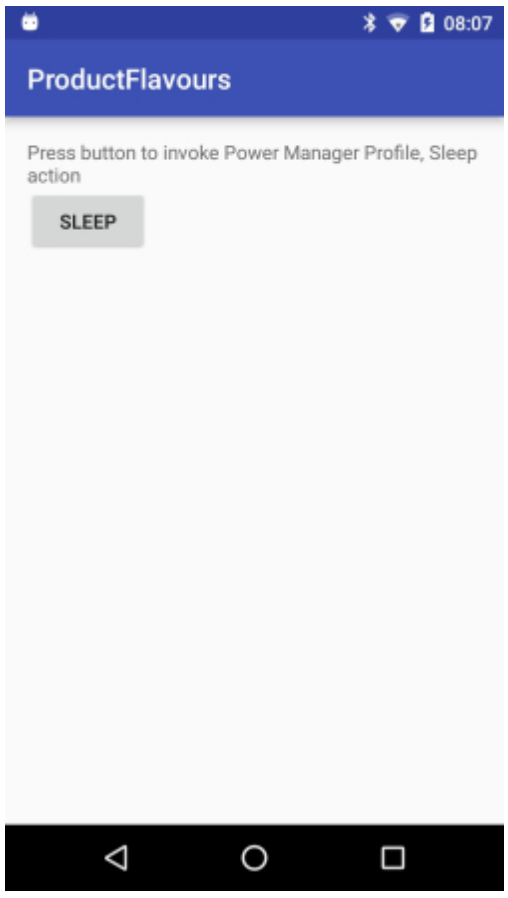


The EMDKConfig.xml under the emdk6\assets folder will be used whenever a build of that flavour is created. Repeat these steps for emdk5. **Note** A complication here is that the EMDK Profile Manager from Android Studio (EMDK > Profile Manager) will always look for the EMDKConfig.xml file under \main\assets\, care should be taken to ensure the correct EMDKConfig.xml file is located in the appropriate <product flavour>\assets folder prior to building (by manually copying or editing the file). Similarly, to edit the existing application profiles using the Profile Manager GUI, manually copy the EMDKConfig.xml file to \main\assets\ prior to launching the GUI.

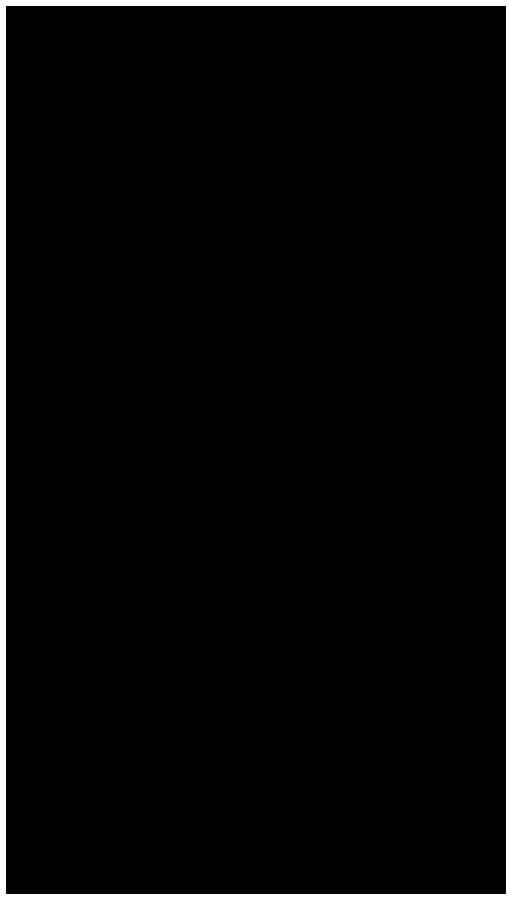
Worked Example

There is a worked example available from https://github.com/darryncampbell/ProductFlavours which shows the principles discussed in this blog. Note that as explained earlier, it is necessary to copy the appropriate EMDK jar files into the project 'lib' directory. The example shows a simple PowerManager profile that will turn the screen off when invoked, by selecting the appropriate build variant the same application can run on both JB and M devices





Power Manager example on M device



Power Manager example after turning screen off (ha ha)

288 Views Comments: 0 Permalink Tags: emdk

What's New for Android 'M' and the impact on Zebra developers

Posted by Darryn Campbell Jan 20, 2017

This document details the developer impact of moving to Zebra devices running Android Marshmallow (API level 23).

Audience for this document

Zebra's new TC51 and TC75x devices ship with Android Marshmallow installed. Any developer targeting those devices must be mindful of Marshmallow considerations. Of Zebra's existing portfolio of Android devices (ET1, MC40, TC55, ET50/55, TC70/75, MC67), there are currently no plans for a Marshmallow upgrade. But for deployments featuring or with the potential to feature a mixture of devices, companies should be familiar with Marshmallow changes.

Overview

The release of Android 'M' (Marshmallow, API level 23) introduced a slew of new features from Google, most notably a new 'Runtime Permissions' model, plus power-saving optimizations for idle devices and applications. Google published an overview of these changes on its developer portal, https://developer.android.com/about/versions/marshmallow/android-6.0-changes.html . This document does not replace recommendations from Google, but is designed to supplement those recommendations for enterprise use-cases.

Dynamic runtime permission model

Android Marshmallow introduces a new permission model whereby users are asked at runtime to grant access to restricted features and hardware such as contacts and cameras.

Consumer device impact

The new permission model is explained in detail in Google's official documentation https://developer.android.com/training/permissions/requesting.html , and Android will utilize this permission model for any application targeting API level 23 or higher. The new permission model necessitates an application change since the user will not be prompted automatically to grant permissions. The application must prompt the user and handle the response appropriately, which means being tolerant of only a subset of required permissions being granted by the user or justifying why certain permissions are required. Applications targeting API level 22 and below will still use the 'old' permission model, whereby all required permissions are accepted by the user during installation. Regardless of the target SDK API level, the user is empowered on Marshmallow devices and above to revoke any application's specific permissions, which necessitates very defensive programming on the part of the application developer as an application cannot rely on any permissions being granted.

Enterprise impact

Expecting device users to grant permissions at runtime is unacceptable to many enterprise customers. At best, users could accidentally deny a permission (e.g. using gloved hands trying to interact with the 'grant permission' dialog's small buttons) or require additional training to allow permissions. Such denial could lead to an inability of a user to perform a required task or at worst, a re-provisioning of the device.

The new permission model is separate from the security features offered as part of Zebra's Mobility eXtensions (MX). For example, the MX Camera manager (http://techdocs.zebra.com/emdk-for-android/6-0/mx/cameramgr/) introduced in MX4.3 can be used to deny access to the device camera through StageNow, MDM or EMDK:

```
<characteristic type="CameraMgr" version="4.3">
<parm name="UseAllCameras" value="2"/> <!-- 2 is deny all
cameras -->
</characteristic>
```

The MX security features will always take precedence over runtime permissions. In the example, even an application requesting access to the camera will not be able to use the hardware if it has been disabled through MX, and doing so will cause an exception

to be raised to the calling application.

Developers targeting Zebra Marshmallow devices have two options:

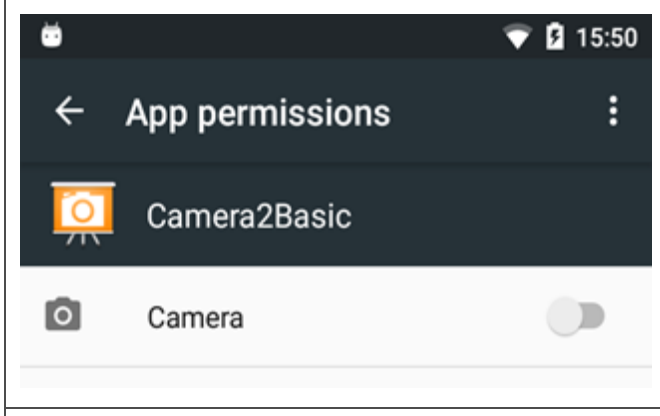
1. Continue to target API level 22 or lower with your application.

Applications targeting API level 22 will not use the new permission model and so will function identically to KitKat or Lollipop devices. You will not have access to Android APIs introduced in Marshmallow, but this may be the quickest option for many developers. The user's ability to revoke specific permissions from the application is controlled via the settings menu (Settings --> Apps --> (Application in question) --> Permissions). The Settings manager parameter 'AccessAppsSection' (http://techdocs.zebra.com/emdk-for-android/6-0/mx/settingsmgr/#ability-to-access-apps-section-in-settings-ui) can be used to ensure the user does not have access to the apps settings section and therefore will not be able to manually revoke permissions.

2. Update your application to API level 23 or higher

If you wish to take advantage of Android Marshmallow features in your application, you will need to re-compile your application to target API level 23 or higher. Applications installed via the AppManager will have their required permissions silently granted and will therefore not present the user with a dialog asking to grant permissions. Calls to checkSelfPermission() (https://developer.android.com/reference/android/content/ContextWrapper.html#checkSelfPermission(java.lang.String)) will return PackageManager.PERMISSION_GRANTED for any permission that is defined in the application's manifest with a <uses-permission /> tag. The AppManager is used by StageNow, EMDK and potentially MDMs with official Zebra device clients to silently install applications where required (http://techdocs.zebra.com/emdk-for-android/6-0/mx/appmgr/).

Applications that have had their permissions silently granted still can have those permissions revoked by the user through the App permissions UI, but as previously explained, the 'AccessAppsSection' parameter can be used to prevent the user from accessing that UI. Revoked permissions will return PackageManager.PERMISSION_DENIED from checkSelfPermission().



A revoked app permission

Applications not installed via AppManager, for example those installed through the Android Debug Bridge (adb) or during debugging in Android Studio will not have their permissions automatically granted. Zebra recommends granting these permissions manually during development through the standard Android dialog presented by using the requestPermissions (https://developer.android.com/reference/android/app/Activity.html#requestPermissions(java.lang.String[] , int)) method call.

In summary, the lifecycle of an enterprise application targeting API level 23 or higher should appear similar to a consumer application:
checkSelfPermission() should be called for the required permission, if this does not return PERMISSION_GRANTED, then requestPermissions() should be called. Doing so will facilitate easier debugging, but applications should be deployed through the AppManager to avoid the user seeing the dialog at any time.

During deployment, the device should be configured to ensure that the user cannot revoke permissions on previously installed applications (using the SettingManager's AccessAppsSection parameter). However, the best practice is still to call checkSelfPermission() to ensure the permission has been granted. If the lifecycle includes a call to requestPermissions(), the application can be debugged, installed

through adb prior to deployment, and be tolerant of misconfigured devices on which the user has managed to revoke the permissions.

Doze mode and App standby

Doze mode and App Standby are related features introduced by Google on Android Marshmallow 6.0 and higher, and are designed to extend battery life by managing how apps behave when under battery power. Google's own overview is comprehensive (https://developer.android.com/training/monitoring-device-state/doze-standby.html), with the key points being:

- All applications are affected regardless of their target API level
- Doze mode & App standby only affects GMS devices. AOSP devices are unaffected by any changes for Doze mode and App standby.
- Doze mode affects an entire device with the device entering doze mode after a period of time with the screen off as long as the device remains stationary.
  - Under doze mode, all apps' access to network and CPU-insensitive services are restricted to a 'maintenance window' which is the only time an application can sync (https://developer.android.com/reference/android/content/AbstractThreadedSyncAdapter.html https://developer.android.com/reference/android/app/job/JobScheduler.html) and trigger alarms (https://developer.android.com/reference/android/app/AlarmManager.html) or hold partial wake locks.
  - App Standby affects a single application, with the system identifying any app the user is not actively using according to a set of pre-defined criteria. Any app identified as not being actively used will be restricted in the same manner described under Doze mode. For example, network access through syncing and pending jobs is restricted along with the ability to trigger alarms.
  - Developers should test their applications under both of these modes to ensure maximum reliability on Marshmallow.
    - It is possible to force your device into both doze mode and app standby using an `adb shell`

Enterprise impact

By and large, the improvements in Android Marshmallow to improve a device's battery life are very positive. Zebra devices are designed with industry-leading battery capacities and address the need for a device to work through the user's whole shift. The primary cause of battery drain on any device (consumer or enterprise) will frequently be post-loaded applications. With this in mind, it is best to **work with the changes** introduced by Google rather than against them wherever possible. Zebra does not currently plan to offer ways for developers to circumvent Android Doze mode or App Standby beyond what is publicly available through Google's APIs.

Push messages

If the enterprise is running Google services (GMS), Google's recommendation to use Firebase cloud messaging (FCM), the successor to Google cloud messaging (GCM) should be followed whenever possible https://developer.android.com/training/monitoring-device-state/doze-standby.html#using_gcm , in particular the use of high-priority GCM messages.

Enterprise developers targeting non-GMS devices (such as those without Maps, Locationing and Google's other value-add services) will not have access to FCM (previously GCM). Although Zebra does not offer an alternative GCM technology, others are available from third parties such as Pushy (https://pushy.me/), PubNub (https://www.pubnub.com/) or Firebase (https://www.firebase.com/), though the 2016 acquisition of Firebase by Google works against advising its long-term use on non-GMS devices. More discussion on this topic is available as a developer blog (https://developer.zebra.com/community/android/android-forums/android-blogs/blog/2016/03/16/pushy-a-gcm-alternative-for-aosp-devices).
Although Doze mode and App standby apply only to GMS devices, any deployment with a mixture of GMS or non-GMS devices will need to be cognisant of the impacts.

Pushy (https://pushy.me/) WILL be adversely affected by both Doze mode and App standby on GMS devices. Any application leveraging pushy or any other MQTT-based messaging client will not receive push messages while the device is in Doze mode; messages will be delayed until the next "maintenance window". Applications leveraging any non-GMS-based push technology, including Pushy, must be whitelisted on GMS devices in order to receive push messages during Doze mode and App standby. This falls under Google's acceptable use cases for whitelisting since GMS is not available on AOSP devices. Please see the subsequent section on Whitelisting applications.

Alternatively, different push solutions can be used in deployments that cover both GMS and AOSP devices though in most cases that is likely to be a sub-optimal solution.

Interaction with Mobile eXtensions
Zebra offers numerous value-adds branded as 'Mobility eXtensions' (MX) that allow additional levels of secure access and configuration on Zebra devices through StageNow, EMDK and approved MDMs.

Power Manager (http://techdocs.zebra.com/emdk-for-android/latest/mx/powermgr/) exposes the ability to put the device to sleep. Instructing the device to go to sleep through the Power Manager will expedite the device entering Doze mode.

DevAdmin (http://techdocs.zebra.com/emdk-for-android/latest/mx/devadmin/) exposes the ability to change the screen-lock timeout interval. The screen-lock interval is distinct from the display-screen timeout, with the former timer only starting after the screen has turned off. Since Doze mode is concerned only with whether the screen is on or off, the screen-lock interval has no effect on Doze mode or App standby.

DisplayManager (http://techdocs.zebra.com/emdk-for-android/latest/mx/displaymgr/) configures the display-screen timeout, which is the time between the last user activity and the device screen turning off. The device screen being off is a prerequisite of the device entering Doze mode.

Cellular Manager (http://techdocs.zebra.com/emdk-for-android/latest/mx/cellularmgr/) settings include various options concerning the use of background data over a cellular connection. Doze mode will take priority over any of the cellular manager options. For example, if background data is set to 'enabled' in the cellular manager but the device is in Doze mode, an application will not be able to use the cellular connection for sync work outside of the OS-defined maintenance windows.

WifiSleepPolicy (http://techdocs.zebra.com/emdk-for-android/6-0/mx/wifi/#sleep-policy) controls whether the Wi-Fi radio will remain enabled when the device is sleeping. This does not override Doze mode; applications will lose network connectivity during Doze mode regardless of this setting.

Whitelisting applications
Google's developer docs (https://developer.android.com/training/monitoring-device-state/doze-standby.html#support_for_other_use_cases) describe a technique for whitelisting applications that makes the apps partially exempt from the limitations of Doze mode and App Standby. It is important to note that the exemption covers only network activity and the holding of partial wake locks (for example the ability to perform processing with the screen off) and that whitelisting does not allow the application to trigger alarms outside of the OS defined maintenance windows.
For Play store applications, Google has applied restrictions limiting the use cases of applications that can request whitelisting (https://developer.android.com/training/monitoring-device-state/doze-standby.html#whitelisting-cases), but the only way for a developer of consumer applications to whitelist an app is to prompt the user to do so through the battery optimization UI (Settings --> Battery --> Battery Optimization). Enterprise developers may wish to whitelist their applications without interaction from the user, but there are no current plans for Zebra to expose this feature during staging. An application can be whitelisted using an adb command:

```
adb shell dumpsys deviceidle whitelist +com.yourcompany.yourapp
```

An application can be removed from the whitelist with the following command:

```
adb shell dumpsys deviceidle whitelist -com.yourcompany.yourapp
```

Zatar client

Zebra's Internet of Things Cloud service, known as Zatar (http://www.zatar.com/), offers a client for Android devices. The Zatar solution is built on ARM's mBed client, which uses Lightweight M2M on CoAP as the protocol for 'things' (in this case our Android device) to communicate with the managing server. As Zebra recommended earlier for Pushy, if you wish to run the Zatar client (or any Android client using an SDK from either mBed or Zatar) it is required to whitelist your application in order for it to run successfully during Doze mode. The Zatar Android application will be incommunicado during Doze mode or App Standby unless explicitly whitelisted.

Android 'N' changes

Though the changes introduced by Google in Android Nougat are by definition outside the scope of this document, it is worth noting that Doze mode has received several enhancements in Android N. The newer version of Doze mode has two phases of system activity restrictions, and the requirement for a device to be stationary to enter Doze mode has been removed. Since this is an area Google is continuing to enhance, Zebra recommends following Google's best practises for application development for Doze mode wherever possible to help ensure future-proofing.

Android for the Enterprise

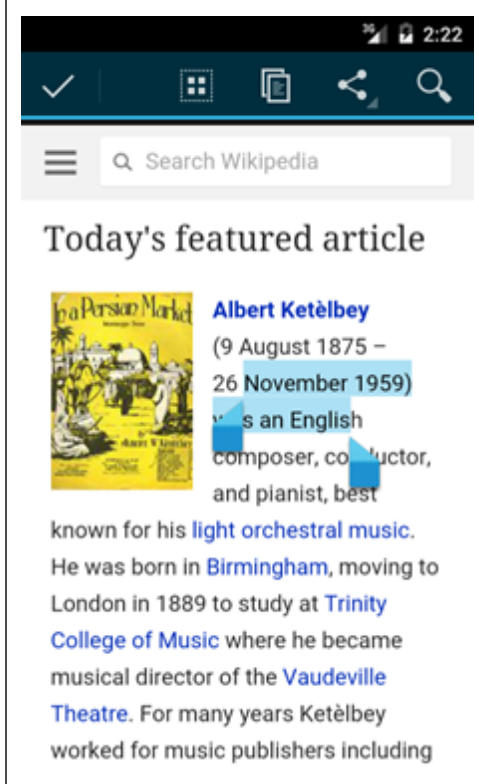
Google's drive toward Android adoption in the Enterprise, formally known as "Android for Work" but now simply incorporated under the Android umbrella, is a suite of solutions from Google to help manage devices and applications, augment overall security, and support an increasing number of enterprise-focused use cases. These include BYOD scenarios, Kiosk mode and others. Google started including such functionality in Android L, and with each new desert flavour has added new features, this is true of Marshmallow and will continue to be true for Android Nougat and beyond.

Although the specifics of what Android for the Enterprise means for your organization is outside the scope of this document, administrators are free to leverage Google tools to either replace or complement Zebra's tools, which share some of the same functionality. For example, Zebra's Enterprise Home Screen shares functionality with Android task locking. It is important to note that Zebra devices do not currently support multiple user profiles on the same device, so enterprise workflows that depend on this (e.g. COPE) will be unavailable, limiting administrators to COSU use cases (https://developer.android.com/work/cosu.html).

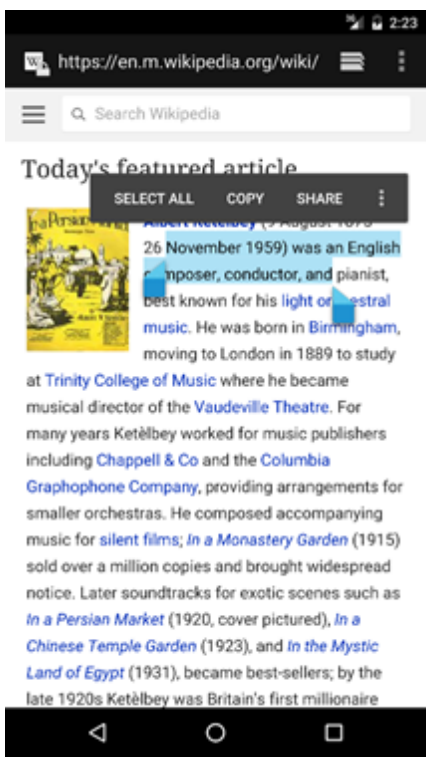
For more information on Android in the Enterprise, please refer to Google's landing page (https://enterprise.google.com/android/) or Zebra's future literature on integrating Android enterprise features with your solution.

New Copy / Paste and sharing features

The Android copy / paste experience has gone through a re-design between Android L and M.



Copy / paste experience on Android L



Copy / paste experience on Android M

On KitKat devices, the copy / paste and sharing experience can be fully controlled by the UI Manager Clipboard feature. Disabling the ClipBoard through the UI Manager will entirely prevent user access to the copy / paste / share menu.
On Lollipop devices, the behavior differs depending on the type of text being selected. A user long-pressing on a text field marked as textIsSelectable (https://developer.android.com/reference/android/widget/TextView.html#setTextIsSelectable(boolean) (for example a phone number in the contacts app) will not be able to access the clipboard. A user long-pressing in a more complex renderer (e.g. the Webview or mail client) will be shown the copy / paste /

share toolbar. And though the copy and paste buttons are non-functional, the user is still able to share and ‘web search’ for any highlighted term.
On Android Marshmallow devices, the look and feel of copy / paste has changed to a floating toolbar with some additional functionality for language translation and integration with Google Now for GMS devices. **No new functions have been added to MX to restrict these features.**
Administrators wishing to restrict access to these additional toolbar features can make use of the MX AppManager to prevent the specific Google packages from providing this functionality. For example, disabling com.google.android.googlequicksearchbox will remove the ‘Search’ and ‘Assist’ options but has the side effect of blocking all Google Now integration on the device.

Access to hardware identifiers

As described in Google’s official changelist for Android M (https://developer.android.com/about/versions/marshmallow/android-6.0-changes.html), calls to get the Wi-Fi (https://developer.android.com/reference/android/net/wifi/WifiInfo.html#getMacAddress()) or Bluetooth MAC address (https://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html#getAddress()) will return the hardcoded value “02:00:00:00:00:00” to protect device security. This is true when running on all Android devices with API level 23 or above, regardless of the target SDK of the application.
Zebra has no plans to allow application developers to circumvent this restriction since a negligible proportion of applications would be impacted by this change.
Developers requiring a way to uniquely identify devices may consider using the ANDROID_ID (https://developer.android.com/reference/android/provider/Settings.Secure.html#ANDROID_ID) or implementing a software solution, the latter protecting against future Android changes in this area.
One possible reason developers could require the Bluetooth MAC address would be if they are targeting a ‘scan-and-pair’ use case with the Zebra RS507 or RS6000 scanners and creating their own pairing app; these peripherals connect as a BT slave by scanning a barcode presented on the mobile device screen. Zebra already recommends using the built-in application ‘Bluetooth pairing utility’ to achieve this rather than trying to develop this capability within your own application. In the case of the RS6000, it is further recommended that tap-and-pair be the preferred method of connection, where possible.

NFC

Prior to Android M, whether NFC was enabled out of the box would have depended on the specific Zebra device you were deploying. It was therefore a best practice during device staging to specify whether you wanted NFC on or off. This can be configured in the Wireless Usage portion of StageNow or with the MX Wireless manager (http://techdocs.zebra.com/emdk-for-android/6-0/mx/wirelessmgr/#turn-onoff-nfc).
Starting in Android M, Google has mandated that NFC-capable devices ship with NFC enabled. Though strictly mandated only for GMS devices, Zebra will take a common approach to NFC on both our GMS and non-GMS device variants wherever NFC hardware is present.
However, it remains a best practice to avoid reliance on the availability of NFC in your application, even though an NFC-capable device will have it enabled out of the box. Zebra recommends that you should continue to specify the desired NFC state during staging and check that the NFC adapter is enabled prior to use:

```
NfcManager manager = (NfcManager)
context.getSystemService(Context.NFC_SERVICE);
NfcAdapter adapter = manager.getDefaultAdapter();
if (adapter != null && adapter.isEnabled()) {
  // adapter exists and is enabled.
}
```

Adoptable storage

Starting with Android M, Google has introduced “adoptable storage,” whereby external storage devices can be ‘adopted’ and behave like internal storage to store applications, application data and media. This opens up a slew of enhanced use cases for the enterprise. For example:

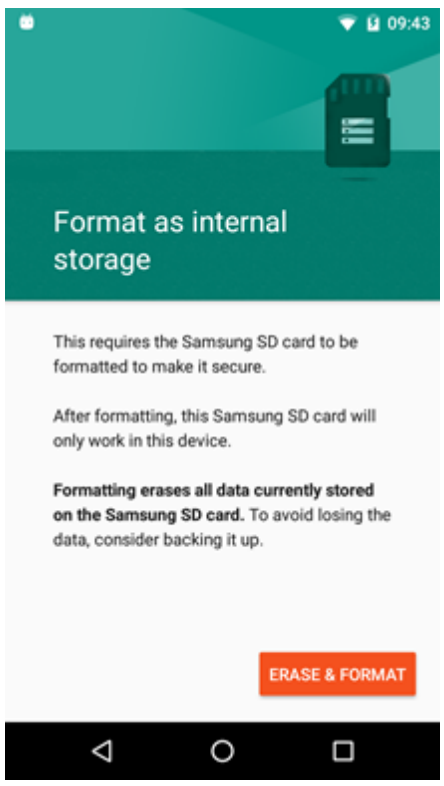
- Without having to separately encrypt the SD card:
  - Secure offline access to video tutorials for field mobility workers
  - Secure offline access to large product databases or additional application content

Adoptable storage is not suitable if you have a class 2, 4 or 6 card, or plan to swap your card between devices, since swapping necessitates re-formatting the card therefore loss of data.

Please note that there are a number of overlaps between adoptable storage and the Encrypt Manager (http://techdocs.zebra.com/emdk-for-android/6-0/mx/encryptmgr/), discussed in the Encryption section.

Several consumer manufacturers do not currently support adoptable storage on their Marshmallow devices, citing potential confusion. But this feature is fully supported on Zebra Marshmallow devices.

To enable adoptable storage on your device, go to Settings, Storage & USB, Portable Storage (Volume), More Options, Settings, Format as internal.



UI to format SD card as adoptable storage

Once the storage is formatted, you are given the option to migrate data to the new storage, potentially freeing up space on the original internal storage. Unfortunately, there is currently no way to adopt an installed SD card as part of your provisioning process through StageNow or EMDK.
Adopted storage will be encrypted and formatted to appear identical to internal storage but as a result is not transferrable to another device. Although the use of adopted storage obviously depends on the presence of an SD card in the mobile device, both Zebra Android M launch devices (the TC51 and TC75x) have this feature.

It is always a best practice when developing applications to not hardcode file paths. Adoptable storage makes this particularly important because applications can be moved between internal and external storage, causing paths to change dynamically. Zebra therefore recommends that developers avoid hard-coded file paths and the storage of fully qualified file paths that were previously built into code. For a full list of APIs to use when building file paths, please refer to the Google development documentation for this feature (https://developer.android.com/about/versions/marshmallow/android-6.0.html#adoptable-storage).

Encryption

Starting with Android M, Google has made full disk encryption mandatory on all new devices. As a Google partner, all Zebra devices running Android M will ship with full disk encryption enabled.
The typical developer targeting an Android M device will see very few differences, though applications implementing cryptography may be impacted by the move from OpenSSL to BoringSSL (https://developer.android.com/about/versions/marshmallow/android-6.0-changes.html#behavior-apache-http-client) or the change to the default Javax cypto cipher provider (http://stackoverflow.com/questions/34286798/javax-

crypto-cipher-working-differently-since-android-6-marshmallow/34307500#34307500
).

Interaction between full disk encryption and Encrypt Manager

If the application makes use of the MX Encrypt Manager (http://techdocs.zebra.com/emdk-for-android/6-0/mx/encryptmgr/), there are additional considerations:

- There is an overlap between the Encrypt Manager's full storage card encryption functionality (http://techdocs.zebra.com/emdk-for-android/6-0/mx/encryptmgr/#sdcard-operation) and adoptable storage. Adopted storage cards will not require encrypting separately since they are encrypted by default during adoption. SD cards that are used as portable storage, (i.e. not adopted) are still subject to separate encryption by the Encrypt Manager
  - Attempting to encrypt an adopted SD card with the Encrypt manager will result in an error, "Cannot Encrypt SD : Mount SD to Encrypt"
- Encrypt Manager's folder encryption mode allows any number of Encrypted File Systems (EFSs) to be created, but isrestricted to non-encrypted storage only. Since internal storage is encrypted, it is only possible to create EFSs on non-adopted SD cards that are not themselves encrypted by the Encryption Manager.
- Calls to retrieve the SD card encryption state through Encryption Manager will return true only if the SD card was encrypted through the Encryption Manager. If an SD card is encrypted by the Android OS during adoption, then SDCardOperation will return 0 (false).

Interaction between full disk encryption and Power Manager

The MX Power Manager (http://techdocs.zebra.com/emdk-for-android/6-0/mx/powermgr/) gives applications and administrators the ability to update the device and to perform several different types of reset. The EMDK documentation goes into detail, but in summary the types of reset are shown in the table below. Please note that adopted storage cards will be made "unadopted" during a full device wipe, factory reset or enterprise reset.

Reset action	Description
Reboot	Normal reboot; same as holding the power key on your phone and choosing reboot.
Full Device Wipe	Wipes everything from the device including all types of storage cards (emulated, physical and adopted), the /data partition and the /enterprise partition. GMS devices will display the setup wizard after reset. Confusingly, the message 'Performing Factory Reset' might appear prior to shutting down your device; this is due to a naming inconsistency between the Power Manager and the Android OS that will be addressed in the future.
Factory Reset	Wipes the /data and /enterprise partitions. Since emulated storage cards are stored on the /data partition in Zebra Android M devices, a Factory reset also will erase the emulated storage card along with any adopted storage card, but will not erase physical storage cards. Confusingly, the message 'Performing Data Reset' might appear prior to shutting down.
Enterprise Reset	Retains all data on the /enterprise partition, allowing configuration to be retained using the Persist Manager, (http://techdocs.zebra.com/emdk-for-android/6-0/mx/persistence/). DataWedge (http://techdocs.zebra.com/datawedge/6-0/guide/advanced/#enterprisefolder), or any other application whose configuration relies on the /enterprise partition. The /data partition is wiped during an Enterprise reset, and since emulated storage cards are stored in the /data partition on Zebra Android M devices, these internal storage cards also will be erased along with any adopted storage card. External storage cards are not affected.

Another feature of the MX Power Manager is the ability to perform Operating System updates.
To perform an OS Update through the power manager on Android Lollipop or below, the associated zip file must be placed in a location that is unencrypted. With Android M and full disk encryption, this limits the locations from which the OSUpdate.zip can be read. At the time of writing, there were no OSUpdate packages available for

Marshmallow devices. When such an update becomes available, please consult the Power Manager documentation (http://techdocs.zebra.com/emdk-for-android/6-0/mx/powermgr/#os-update-zip-file) for details on how to perform an OS Update programmatically on a Marshmallow device.

To perform an OS update without relying on the EMDK or MX Power manager (i.e. manually) you can no longer just copy the update zip file to the emulated storage card as the bootloader will not be able to read the encrypted storage. You have two options:

- Select the option to load the update from adb. Then, assuming you have all the correct device drivers and Android tools installed, use the `adb sideload <osUpdateFile.zip>`
- Download the OS update zip file to an unencrypted, unadopted (portable) physical SD card inserted to the device and apply the update through the boot loader.

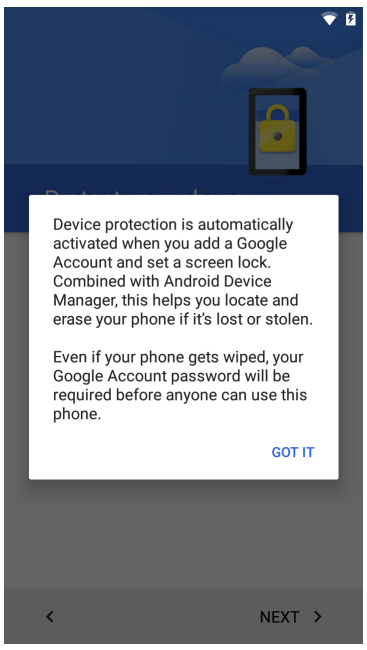
This technique also applies when flashing a new OS to your device, and is explained in the device quick start guide.

Performing OSUpdates via EMM or using StageNow's OSUpdate service has the additional option of using a fixed location to apply the update from, /data/tmp/public, but this location cannot be used with the MX Power Manager's update method.

Interaction between the Power Manager and Google's Device Protection Features

"Device protection" is a security feature available on GMS devices that are associated with a Google account and have a screen lock enabled (e.g. PIN), it was introduced in Lollipop MR1 and is intended primarily to reduce the value of stolen devices by preventing the use of a device unless authorized by the known user.

Device protection is enabled automatically if a Google account & screen lock are added to the device during the set-up wizard but will not be enabled if only one or the other is applied. After running the set-up wizard, device protection can be applied by assigning both a Google account and screen lock to the device.



Set-up wizard confirming device protection

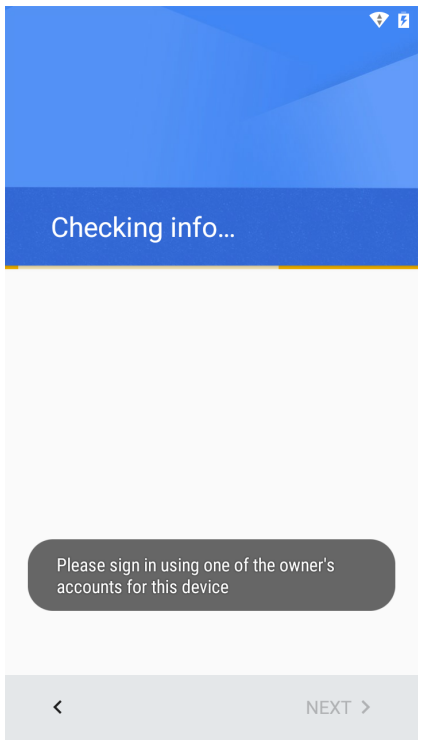
With device protection enabled, Android's device manager can be used to remotely locate, lock and reset the device. Whilst intended primarily for consumer devices there are obviously applications to enterprises looking for a free way to achieve wipe functionality with unmanaged deployments. Android's device manager can also assign a screen lock to a previously unprotected device as long as that device had a Google account (therefore causing device protection to be enabled).

There are different types of reset for protected devices:

Untrusted

- Untrusted factory resets mandate that the user re-enter the Google account and password previously associated with the device during a reset. This ensures that the person resetting the device owns it and will help block stolen devices. If multiple Google accounts were previously associated with the device then the credentials from any account are sufficient.
  - Examples of untrusted factory resets are resets from the Android device manager as well as any of the reset types invoked from the MX Power manager, as described above: Full Device Wipe, Factory Reset and Enterprise Reset.
    - This means that a protected device cannot be disassociated from a Google account through the Power Manager

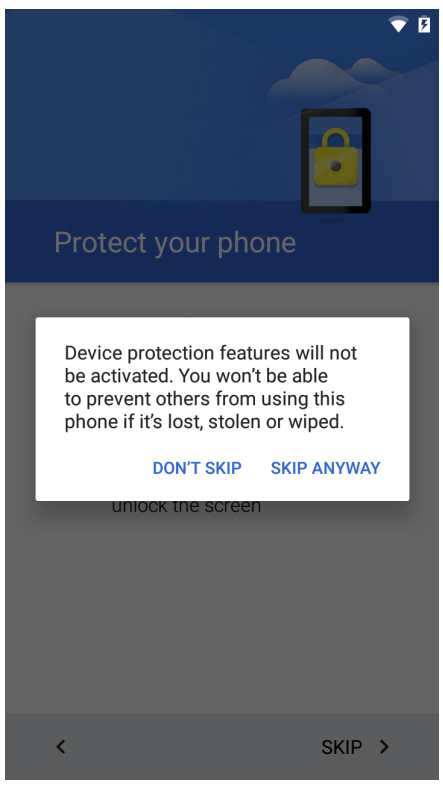
- The factory and enterprise reset packages available from Zebra support are also examples of untrusted resets. If you need to clear Google's device protection features on your device and do not have access to the associated Google account then contact Zebra support..
- Signing in with a Google account not previously associated with the device will result in an error and you will be asked to try again:



Error toast given for using an incorrect Google account

Trusted

- Trusted factory resets do not mandate the user re-enter any previously associated Google account information
  - Examples of trusted factory resets are those invoked from the device settings UI --> Backup & reset e.g. 'Enterprise Data Reset'.
  - During a trusted factory reset the set-up wizard will give you the option of enabling or disabling device protection (by specifying a Google account and screen lock). Choosing not to apply either of these will result in the device no longer being protected:



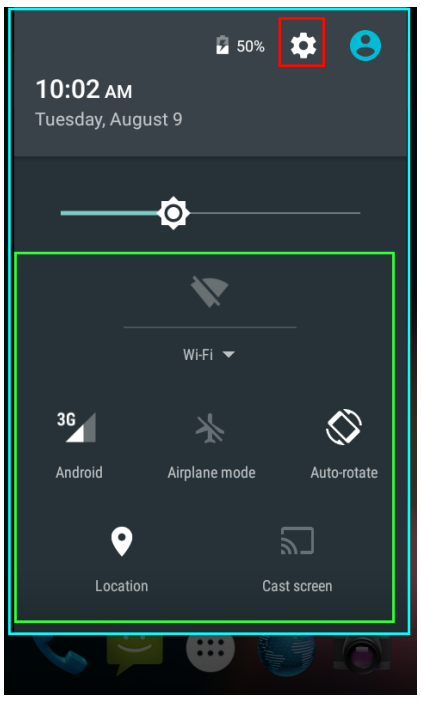
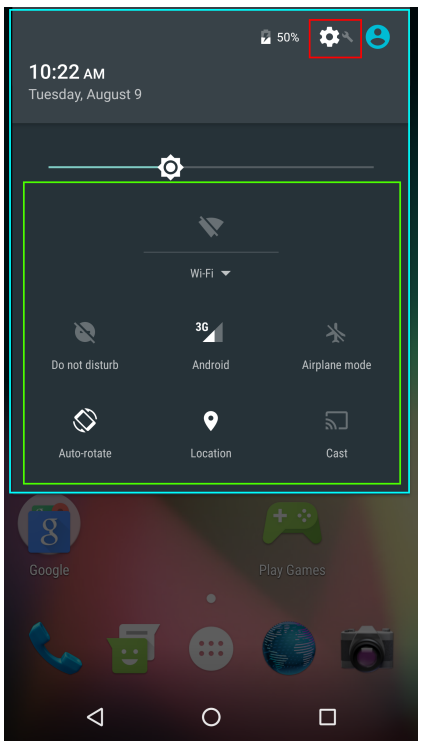
Set-up wizard warning that you are not protecting the device.

New notification shade and quick settings menu

Every iteration of Android has made either major or incremental updates to the pulldown notification shade and associated options. Android M introduces incremental updates to the major changes that were brought in for Lollipop and several options are available to Zebra developers to lock down access to this notification shade.

For Lollipop devices, Zebra introduced changes to the UI Manager related to the notification pull down that can be programmatically accessed through StageNow or the

EMDK:

	
Configurable sections of notification pull-down on Android L	Very similar configurable sections of notification pull-down on Android M

Notification Pull Down (http://techdocs.zebra.com/emdk-for-android/6-0/mx/uimgr/#notification-pulldown-enabledisable)
When disabled it is not possible to drag the notification pull down at all. This section is colored blue in the diagrams above.
Notification (Icon) Settings (http://techdocs.zebra.com/emdk-for-android/6-0/mx/uimgr/#gear-icon-in-notifications-panel-showhide)
When disabled, the gear icon is not shown in the notification pull down, highlighted in red in the diagrams above.
(**Notification) Quick Settings** (http://techdocs.zebra.com/emdk-for-android/6-0/mx/uimgr/#two-finger-quick-settings-enabledisable)
When disabled, the 'quick settings' used to modify Wi-Fi, Airplane mode, location etc., are not shown to the user. This section is colored green in the diagrams above. Note that the quick settings option does not include the brightness control.
Similar functionality also is exposed through the Enterprise Home Screen application configuration parameter disable_statusbar_pulldown (which introduced support for Android M in version 2.5)

User account icon
On consumer versions of Android, the user icon shown in the top right-hand corner  of the notification shade provides quick access to the different user accounts configured on the device. User accounts were officially added to consumer phones starting with Android 5.0 (Lollipop) but also were accessible on consumer Android back in 4.2. The focus of user accounts was on consumer use cases, such as sharing a tablet with multiple family members or a guest mode for a friend to borrow the device.
Android device "users" should not be confused with Enterprise Android profiles, and "users" are not available on Zebra Lollipop or KitKat devices. User accounts continue to be unavailable on Zebra Android devices running Marshmallow, and for this reason you will not see the account icon on your Zebra Marshmallow device. Multi-user mode cannot be enabled on Zebra devices without using unsupported tools or mechanisms, and there are no plans to implement multiple user accounts for Android Marshmallow at this time.

System UI tuner
Long-pressing on the Settings gear in Android M will unlock the 'System UI Tuner,' which presents an additional settings menu. This allows greater configuration of the Notification UI, such as configuring which Quick Settings icons are shown and modifying the UI that appears when the volume buttons are pressed.
There are two approaches to locking down this functionality:

1. Disable the ability to long-press on the Settings gear in the first place by hiding it using the UI Manager's

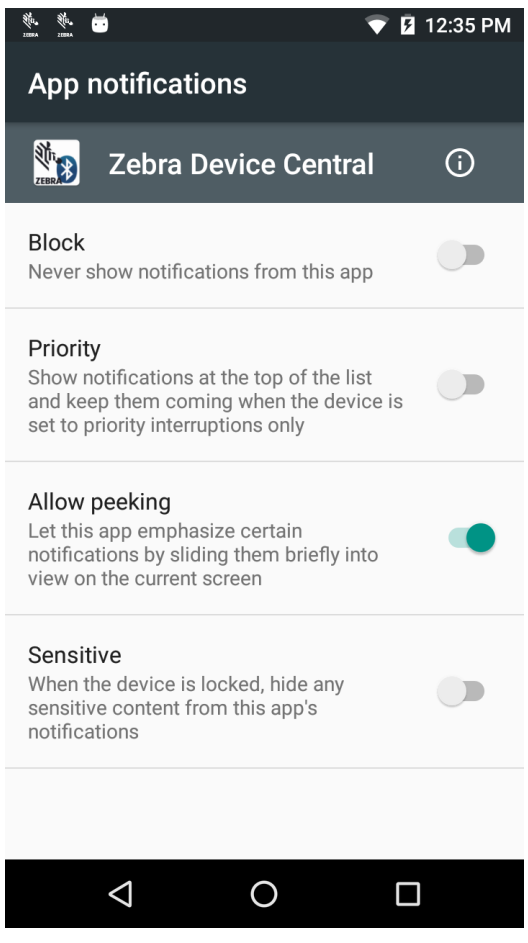
Notification (Icon) Settings feature (http://techdocs.zebra.com/emdk-for-android/6-0/mx/uimgr/#gear-icon-in-notifications-panel-showhide).

2. A less elegant approach would be to restrict access to the device settings entirely using Enterprise Home Screen or MX AccessManager (http://techdocs.zebra.com/emdk-for-android/6-0/mx/accessmgr/#system-settings-access). Note that it is not possible to specifically control the visibility of the “System UI Tuner” option.

Long press on notifications

On Marshmallow, if the user long-presses on an application notification they are presented with an information button  . Clicking the information button will by default give access to the ‘App notifications,’ offering options to block or prioritize the notification.

Long-pressing on an application notification to bring up additional information has been present in Android since KitKat, and the MX AppManager introduced a property to “disable access to application info for all applications,” (http://techdocs.zebra.com/emdk-for-android/6-0/mx/appmgr/#access-to-app-info-action) which locks down this and other related features. Unfortunately, this parameter of AppManager is not supported beyond KitKat. A related MX feature is the Settings Manager parameter ‘AccessAppsSection’ (http://techdocs.zebra.com/emdk-for-android/6-0/mx/settingsmgr/#ability-to-access-apps-section-in-settings-ui). While this blocks ‘Apps’ access from the Settings UI and prevents the user from escaping the ‘App notification’ screen, long-pressing a notification will still bring up the ‘App notifications’ settings. The only way to reliably block access to the ‘App notifications’ settings screen with MX is by using the Access Manager’s ‘system settings access’ parameter (http://techdocs.zebra.com/emdk-for-android/6-0/mx/accessmgr/#system-settings-access) to prevent access to all but a small subset of settings. You can disable the com.android.settings package using the Application Manager but doing so will give an ambiguous error to the user if they long press on the notification icon, "System UI has stopped" as opposed to the more informative error you get when using the recommended access manager, "OSX Admin has disabled app notifications".



App notification settings

Interaction with built-in and post-install applications

There are four types of applications that can run on your Zebra Marshmallow device:

1. Applications from Google such as the Web browser, calculator, etc. These will have their Android Marshmallow variants present out of the box as you would expect, and the choice of applications will differ between the GMS and non-GMS device variants. For example, Google Maps will be available only on GMS devices.

2. Applications from Zebra that are pre-installed from the factory. These include DataWedge, AppGallery, and depending on the specific device, perhaps an MDM agent or printing utility.
Less visible than the application updates will be an updated version of Zebra's Mobility etensions Management Framework (MXMF) and OSX, Zebra's eXtensions to the Android OS (completely unrelated to Apple's OSX). You can find out your device's versions of MXMF and OSX by going to Settings à About Phone/Device à SW components. Developers can access MX through EMDK and see which features are available for which MX configuration in the MX compatibility matrix (http://techdocs.zebra.com/mx/compatibility/).

3. Applications from Zebra that are post-loaded onto the device. These applications are downloaded from Zebra's support portal, and might require a license. These include Enterprise Browser and Enterprise Home Screen. The important differentiation here is that a user of a post-loaded application will need to ensure the version in use supports the version of Android being targeted. For example, Enterprise Home Screen (EHS) 2.5 is the most recent release at the time of writing and supports a single Marshmallow device, the TC51 (http://techdocs.zebra.com/ehs/2-5/guide/about/). Please note that in order to ensure maximum reliability and compatibility, these post-loaded applications are supported per device and not per Android version. For example, EHS 2.4 supported the WT6000 running Lollipop, but not the TC75 running Lollipop, which requires at least EHS 2.5 or later. Post-loaded applications will be updated to support new devices as close to that device's launch date as possible. Post-installed applications are treated as separate products and may document specific considerations for Android M development. For example, Enterprise Browser might choose to expose new features that are supported only on Android M, but such features are outside the scope of this document. Please consult individual product documentation for more information.

4. Applications built by customers and partners: These are the applications that will be written by the likely audience of this technical note. Applications making programmatic use of enterprise value-add hardware such as the barcode scanner, notification beeper or payment capabilities will depend on Zebra's EMDK, which is certified only for a specific list of devices for each release. For example, EMDK for Android 6.0 currently supports a single Marshmallow device (http://techdocs.zebra.com/emdk-for-android/6-0/guide/about/), the TC51. User applications targeting Marshmallow must utilize an EMDK variant that also supports Marshmallow. EMDK for Xamarin developers will have a similar experience, requiring at least EMDK for Xamarin 2.2, thefirst version with Marshmallow support (http://techdocs.zebra.com/emdk-for-xamarin/2-2/guide/about/).
Developers also need to take care that the features they include in their application are supported by the versions of OSX, MX and EMDK they are targeting as defined in the compatibility matrix (http://techdocs.zebra.com/mx/compatibility/). The versions of MX and OSX on the device can be seen at Settings à About Phone à SW Components

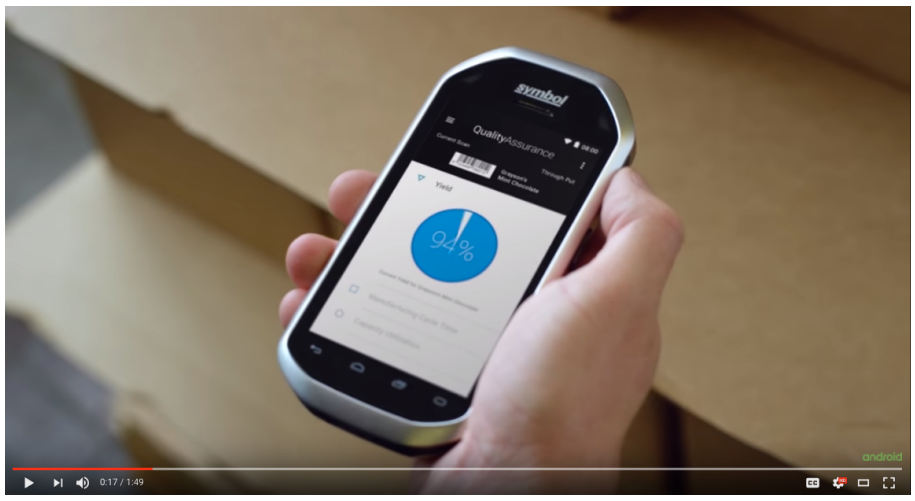
812 Views Comments: 0 Permalink Tags: emdk, marshmallow

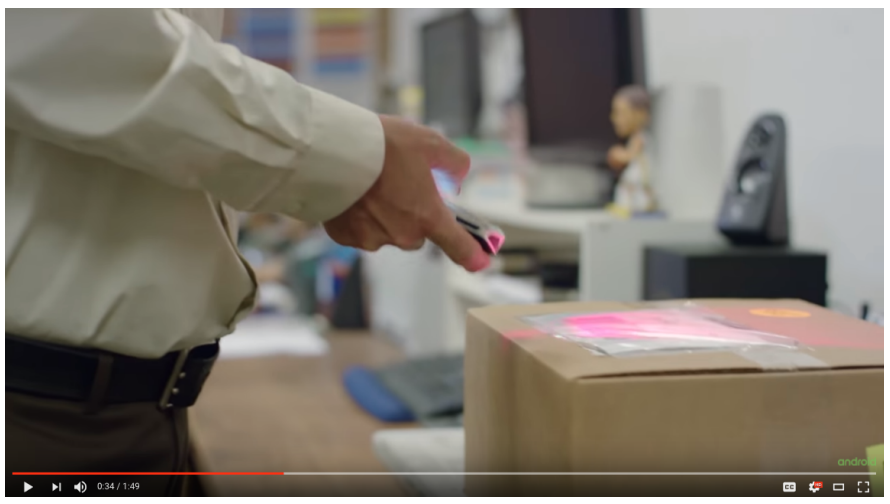
Zebra Featured in 'Powering the World' Video

Posted by Edward Correia Dec 20, 2016

Zebra devices played a prominent role in a video made recently by Google to promote Android as the dominant platform for devices and software for the workplace. It blends the stories of a manufacturer and shipping department, its grocer customer, and a father of twins during a trip to the doctor's office. In a span of less than two minutes, the story unfolds of a chocolate bar's journey from melty molding to ultimate end.

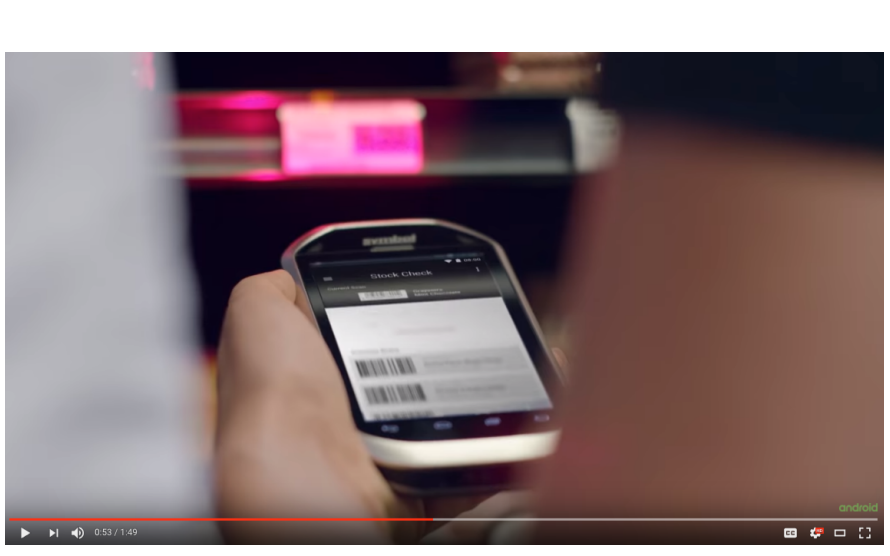
The candy bar's made, monitored and quality-checked using Android-based imaging and a Zebra MC40 Mobile Computer . Then an order is shipped to the grocer, again with the help of an MC40.

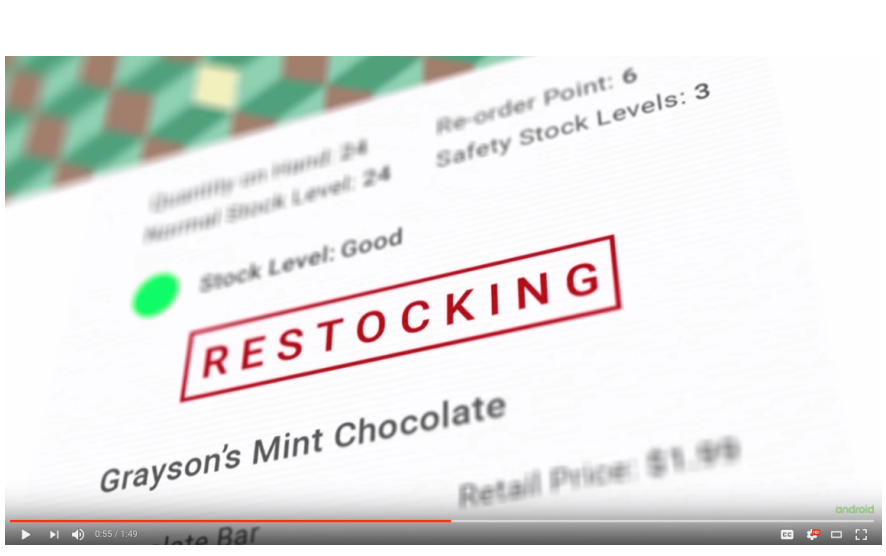




Meanwhile at the grocer, the dad's proximity to the sweets section is detected, prompting a wall-mounted Android tablet to alert him to the day's special offer.

Undeterred, he heads for the chocolate bars, only to find that the twins' favorite is out of stock. Fortunately, a nearby clerk uses her MC40 to kick-off the restocking process.





Zebra Technologies offers solutions for all of these enterprise scenarios, including its Enterprise Asset Intelligence solutions for the workplace.

Watch the whole video, then learn how Zebra Solutions can help your business.

Android: Powering the world's work devices - YouTube

157 Views Comments: 0 Permalink

Announcing The EMDK for Android 6.0 Release

Posted by Rob Galvin Nov 28, 2016

The latest EMDK For Android - version 6.0 has been released and is now available for download. In this release there has been several new devices supported, features added, samples updated and many bug fixes.

Here is a brief summary of what is new in this release. The full online documentation can be found at: techdocs.zebra.com

1. Added support for ET50, ET55 and TC8000 Lollipop (Android 5.1.1) devices.
2. Added support for MX v6.1 in Profile Manager:
   - Threat Manager – Added new feature to configure the detection of rooting and to apply a countermeasure when the device is detected as being rooted.
   - UI Manager – Added new feature to allow or disallow network monitored messages/notifications.
   - Bluetooth Manager – Added new feature to enable or disable the mobile device discoverability.
   - GPRS Manager – Added new capability to add APN and set certain parameters such as proxy, server, port.
   - Wi-Fi – Added new feature to enable or disable the password protected encryption.
   - Device Administrator – Added new feature to select the type of screen lock such as no password, password, pattern and swipe.
   - Wireless Manager – Added new feature to specify balance between accuracy and battery life when using GPS.
   - KeyMapping Manager – Added key mapping support for Rear Button and Grip Trigger 2.
3. Enhanced Barcode Manager APIs with the following features:
   - Added new decoder parameter in Code128, Code39, I2of5 and UpcEan decoders in ScannerConfig.DecoderParams:
     - reducedQuietZone - Flag to enable or disable the decoding of margin less barcodes.
   - Added new parameter to ScannerConfig.DecoderParams.Gs1DatabarLim:
     - securityLevel - Sets the four levels of decode security for GS1 Databar Lim barcodes.
   - Added new reader parameters in ScannerConfig.ReaderParams.ReaderSpecific.ImagerSpecific:
     - oneDQuietZoneLevel - This parameter sets the effort at which the decoder will attempt to decode margin-less barcodes.
     - poorQualityDecodeEffortLevel - This parameter permits selection of enhancement modes for decoding barcodes of poor or degraded quality.
     - Added new reader parameters in ScannerConfig.ReaderParams.ReaderSpecific. CameraSpecific:
     - viewfinderSize - Sets the View Finder window size in camera scanner as a percentage of full width and full height.
     - viewfinderOffsetX - Sets the X axis position of the top left corner of the view finder.
     - viewfinderOffsetY - Sets the Y axis position of the top left corner of the view finder.
     - oneDQuietZoneLevel - This parameter sets the effort at which the decoder will attempt to decode margin-less barcodes.
     - poorQualityDecodeEffortLevel - This parameter permits selection of enhancement modes for decoding barcodes of poor or degraded quality.
   - Added new reader parameters in ScannerConfig.ReaderParams.ReaderSpecific.LaserSpecific:

- adaptiveScanning - This parameter enables or disables the adaptive scanning.
- beamWidth - Controls the beam width of the laser scanner. Laser beam width can be shortened or widened using this parameter.
- oneDQuietZoneLevel - This parameter sets the effort at which the decoder will attempt to decode margin-less barcodes.
- poorQualityDecodeEffortLevel - This parameter permits selection of enhancement modes for decoding barcodes of poor or degraded quality.

- Added new enums in ScannerConfig:
  - OneDQuietZoneLevel - Describes the effort at which the decoder will attempt to decode margin-less barcodes.
  - PoorQualityDecodeEffortLevel - Describes the selection of enhancement modes for decoding barcodes of poor or degraded quality.
  - AdaptiveScanning - Enable or Disable Adaptive scanning.
  - BeamWidth - Controls the beam width of the laser scanner.
  - GS1LimitedSecurityLevel - Security level addition of GS1 DataBar lim decoder.

4. Enhanced DataCapture feature in the Profile Manager:
   - Added new reader parameters:
     - Aim Timer - Sets the duration (in ms) for timed aim modes.
     - Viewfinder Size - Sets the View Finder window size in camera scanner as a percentage of full width and full height.
     - Viewfinder X Offset - Sets the X axis position of the top left corner of the view finder.
     - Viewfinder Y Offset - Sets the Y axis position of the top left corner of the view finder.
     - Character Set Selection - Allows the user to convert the barcode data if different from default encoding type.
     - Added new values for Aim Type:
     - Timed Hold - Once trigger is pressed, an aiming session is started for a time specified by Aim Timer. When this time expires, a decode session is started and scan beam will be visible. The decode session will remain active until the Beam Timer expires, the trigger is released or a barcode is decoded.
     - Timed Release - Once the trigger is pressed, an aiming session is started and will continue until the trigger is released. If the Aim Timer is expired when the trigger is released, a decode session will be started with scan beam visible for a remaining time equal to Beam Timer or a barcode is decoded.
     - Press And Release - The scan beam starts when the trigger is pressed and released. The decode session will remain active until the Beam Timer expires or a barcode is decoded.
     - Added new values for Character Set Selection:
     - ISO-8859-1 - Allows the user to convert the barcode data using ISO-8859-1 character encoding type.
     - Shift_JIS - Allows the user to convert the barcode data using Shift_JIS character encoding type.
     - UTF-8 - Allows the user to convert the barcode data using UTF-8 character encoding type.
5. Enhanced the performance of using decodeAudioFeedbackUri in ScannerConfig.ScannerParams in Barcode Manager API.
6. The ProfileConfig class which can be used to access the profile data has been deprecated. It is recommended to use the name-value pair function of Profile Manager feature. The Profile XML can also be directly modified. Refer to the Clock Sample for information on modifying Profile XML data.
7. `Fixed:` Toggling Hard trigger and soft trigger sometimes results into cancel read exception.
8. `Fixed:` In earlier versions, selecting EMDK APIs as target Compile Sdk Version in the Android Studio project would result in the compilation error. This issue is now fixed.

256 Views Comments: 0 Permalink Tags: emdk, android;, zebra_news

Progressive Web Applications in the Enterprise

Posted by Darryn Campbell Nov 13, 2016

Introduction

If you are a developer, then you have no doubt heard of Progressive Web Apps (PWA). If you are a hybrid or web developer, then you may be considering developing your next application as a PWA. There are already some great resources online explaining exactly what PWAs are and how to get started writing your own so I won't cover any of that here; the top results you'll find will be Google's overview , Google's getting started guide and another good getting started blog from a Google engineer, all dating back to late 2015 just after Google announced the methodology at their IO conference.

So, after a year in the wild you would expect the technology to have matured a bit. In this blog I will look at PWAs from an enterprise perspective and see if they a sensible choice for an enterprise application.

What is a Progressive Web App (PWA)?

The important point about PWAs is that they are not a single technology, PWA is an umbrella term to describe a development

methodology that encompasses numerous characteristics. From Google's own definition these characteristics are:

Progressive, Responsive, Connectivity independent, App-like, Fresh, Safe, Discoverable, Re-engageable, Installable, Linkable.

At first glance the focus on consumer use cases is evident. Consumer applications are concerned with the discoverability of their applications in app stores [*Discoverable*], are seeking their users to engage frequently with their apps to maximize add revenue [*Re-engageable*] and are seeking to overcome the installation barrier where most users are dissuaded from installing an application in the first place as it involves multiple steps (visit app store, click install, wait for download, open etc) [*Installable*].

There is substantial overlap however between enterprise use cases and progressive web apps: As we see tablets proliferate in the enterprise space then [*Responsive*] apps increase in importance; keeping the application fully server-based avoids the need to coordinate application updates across remote devices [*Fresh*] and being [*Connectivity Independent*] might finally deliver on the promises made but never quite realised by Application Cache in HTML5

Are Progressive Web Apps a good fit for Enterprise Development?

As mentioned in the previous section there are clear benefits for enterprise applications to use a subset of Progressive Web app functionality.

Let us assume the following use cases when developing an enterprise progressive web app:

- We want to make use of service workers and cache so our application is available offline or during poor connectivity
- We want to use the Web manifest to remove the browser UI and brand our application with the appropriate theme colour, splash screen & icons.
- We do not want to rely on the end user installing the PWA themselves after being prompted by a mysterious Google heuristic.
- The connection to our server will be over HTTPS in line with best practice, PWA mandating use of HTTPS is therefore not a problem.
- We want our application to make use of push notifications

Critically here the enterprise requirement to not allow users to install progressive web apps themselves goes against the fundamental tenets of PWAs. A primary use case of progressive web apps is that the end user will visit a mobile web site and then add that site to their device home screen, either after receiving a prompt or manually doing so through the browser menu. Adding the PWA to the home screen through any other technique is not supported by Google and further, we cannot prevent Google offering the installation popup to the user so long as our application meets the requirements of a PWA (i.e. it runs over HTTPS, has a service worker and has an application manifest).

Note: Although manually adding an application to the home screen is not officially supported, see the end of this blog for a discussion of how that might be achieved.

In conclusion

Progressive web applications are not suitable for Enterprise Development because they put the role of application management solely in the hands of the end user, bypassing any IT management solution. By disallowing user-centric application management we also lose the branding benefits of being installable (icons on the home screen and a custom splash screen) so a manifest is also no longer required.

Being an umbrella term however we can choose to leverage those aspects of Progressive web applications which offer potential value to the Enterprise:

- Service workers
- Push notifications
- HTTPS
- Server-based application logic

Are Progressive Web Apps supported on Enterprise Devices?

As mentioned in the previous section, an end to end PWA solution is not applicable for Enterprise applications but we will leverage the best of what PWAs have to offer for our Enterprise use cases, specifically:

- Service Workers : Bringing together previous functionality offered by 'Web workers' and 'Application Cache', Service workers seek to make an app function in a partially connected environment.

- Push notifications : Making use of service workers' background processing, push notifications offer a platform agnostic notification mechanism independent of Google / Firebase cloud messaging.

- HTTPS and server-based application logic: Fundamental to the idea of a 'web app' is to ensure the latest version of your application is available on your device in a secure fashion.

Support for service workers and push notifications are delivered through Chromium and are therefore dependant on the webview being used.

A Chromium Blog from late 2014 states that Service Workers were introduced in Chromium version 40 however as with any software development I suspect the true level of support has been enhanced over time. This blog is a good resource to track Service Worker status across the different versions of Chromium.

Which webview is in use can become confusing so please see below:

Chrome for Android: GMS devices will have Chrome for Android pre-installed as part of Google services along with other Google value adds such as the Play Store, location services etc. Chrome for Android (com.android.chrome) can be updated via the play store and is built on the Chromium open source project. There are a number of variants present in the play store supporting different versions of Chromium e.g. Chrome Canary & Chrome Beta. Importantly, you will not find Chrome for Android on non-GMS devices or be able to install it in a supported manner.

Android Browser: This will be used as the rendering engine for natively built applications that expose a webview to the user, for example Enterprise Browser and by default with Ionic, Cordova and PhoneGap. The Android Browser also has a browser UI which will be the only browser present by default on non-GMS devices. For enterprise GMS devices, it is conceivable that you would have two browsers: Chrome for Android and the Android Browser. Prior to Android 5.0 (Lollipop) the Android Browser version was not independently updatable and required the whole Android OS to be updated. From Android 5.0 onwards the Android Browser is updatable through the Google Play Store under “Android System WebView” so a more recent Chromium version can be supported. Android Browser tends to lag slightly behind Chrome for Android in terms of the version of Chromium supported.

Crosswalk: This popular third party browser view is also built on Chromium so will support service workers and push notifications. Crosswalk is popular amongst Enterprise developers as it provides a consistent and known webview instead of having to worry about which versions of Android Browser or Chrome are present on the device. Developers of Cordova based applications may choose to use Crosswalk and there are tentative plans to add Crosswalk support to Enterprise Browser in a future version.

Opera: A third party browser built on Chromium. Opera does not ship by default on any Zebra device but is an option for app development as it claims support for progressive web applications. Opera could be a replacement for Chrome for Android with one possible use case being the application needs to run on a non-GMS device and the customer has side-loaded Opera onto devices as the default browser.

Is it supported on my device?
The best way to check if Service Workers or the Push API are supported on your target devices is to use the website ‘caniuse.com ’:

ServiceWorkers / Cache: http://caniuse.com/#feat=serviceworkers

Push API: http://caniuse.com/#search=push%20api

Bearing in mind you are interested in either ‘Android Browser’ or ‘Chrome for Android’, depending on how you are targeting your application.

One limitation of the site is it only shows results for the latest version of Chromium supported by the browser. For example you might have a non-GMS Lollipop device whose ‘Android Browser’ is not up to date with the latest ‘Android System WebView’. In this instance, it is helpful to use https://www.whatismybrowser.com which will tell you the version of Chromium being run.

Code solutions to check for the presence of the JavaScript features would be to test as follows:

```
01. if ('serviceWorker' in navigator) {  console.log(‘service workers supported’);}
01. if ('PushManager' in window) { console.log(‘push messaging supported’);}
```

Accessing Enterprise APIs

If you are writing an application to run on Enterprise devices then it is very likely you need to make use of enterprise hardware such as the barcode scanner, Bluetooth payment solution or NFC.
There are a number of ways to achieve this through a web app (though technically if you are using hardware APIs this could be classified as a hybrid application).

Enterprise Browser (EB)

Enterprise Browser is the Zebra recommended solution for developing web-based or hybrid applications. As previously mentioned the webview in which your EB application is running is based on the Android Browser and therefore on Lollipop devices or above you should expect to be able to make use of service workers but not push notifications. This is because at the time of writing Push Messaging is not supported in the Android Browser.

As well as many other features Enterprise Browser includes an “eb” JavaScript namespace offering JavaScript APIs for barcode

scanning, NFC and much more.

Note that there are tentative plans for Enterprise Browser to more thoroughly flesh out which PWA features are supported on which devices in their documentation and samples.

DataWedge

DataWedge is a powerful service running on every Zebra device which offers a zero-code method to integrate with the device hardware. To use DataWedge you define 'profiles' which describe how to control the hardware when specified applications come to the foreground, for example you can enable the barcode scanner when your application is launched. Data would be returned to the application in the form of keystrokes, so scanning a barcode with the cursor in a text box would result in the text box being populated with the decoded data.

One advantage of DataWedge is it is able to run with any application being in the foreground, including Chrome for Android. Since Chrome for Android supports the push API (unlike the 'Android Browser' at the time of writing) it would be possible to write a web application running in Chrome and utilising both push notifications and barcode scanning.

When defining the application to be associated with the Datawedge profile you would choose `com.android.chrome`. The Chrome activity to choose is less obvious, if you were developing a true Progressive Web Application then the activity would look something like `org.chromium.chrome.browser.webapps.WebappActivity0` but for a web-app just running in a Chrome tab it might be safest to specify * for the activity.

To view a list of current running activities you can run

```
01. $adb shell dumpsys activity
```

Custom proxy activity

Though a bit more work, it would be possible to interface with the device hardware by writing a custom android application which would act as a proxy between the web application and the native APIs.
The proxy application would expose some custom defined URI scheme which could then be invoked from the browser, similar to how clicking a `<a href="tel:xxxx">dial</a>` link will bring up the Android phone dialer. Rather than dial a number however the proxy application could use native APIs such as the EMDK for Android to perform any action e.g. integrate with a Bluetooth payment peripheral.

Note however that this communication would only be one way, the options for returning data back to the calling web-app are limited and would require some ingenuity.

The linked stack overflow question is relevant to this scenario.

An Aside: Provisioning a PWA home screen icon

As mentioned earlier the only officially supported techniques for adding a PWA icon to the device home screen are initiated by the device user, either the user selects the relevant menu item from the browser menu or they give a positive response to a popup presented to them by Chrome after some Google heuristic determines they have been interacting with the web app sufficiently.

It would be nice if it were possible to have greater control over whether or not those icons are placed on the home screen and ideally provision a device so the Progressive Web Apps are already present without the user having to interact with the application at all.

It is possible to create a shortcut on the home screen using the undocumented INSTALL_SHORTCUT action but whilst creating a shortcut to a URL is straight forward, the intent to launch a progressive web app is more complicated (containing the application icon amongst other details). The intent which we need to store in this shortcut and defines the PWA is undocumented but we can find out the details of shortcuts on the home screen using the technique detailed here .

Unfortunately, my attempts to reconstitute the Intent required to launch the PWA were unsuccessful. Invoking the intent directly from adb produced an unresolvable permission error and attempting to insert the Intent into a shortcut proved troublesome.

After a few hours of trying I did not pursue this avenue any further given the completely unsupported nature of adding shortcuts to the home screen on Android. Your typical developer would not want to rely on this functionality continuing to work moving forward, further, there are currently no plans by Zebra to offer shortcut installation as part of their MX suite or Enterprise Home Screen.

617 Views Comments: 0 Permalink Tags: zebra_news, progressive web apps

StageNow Troubleshooting

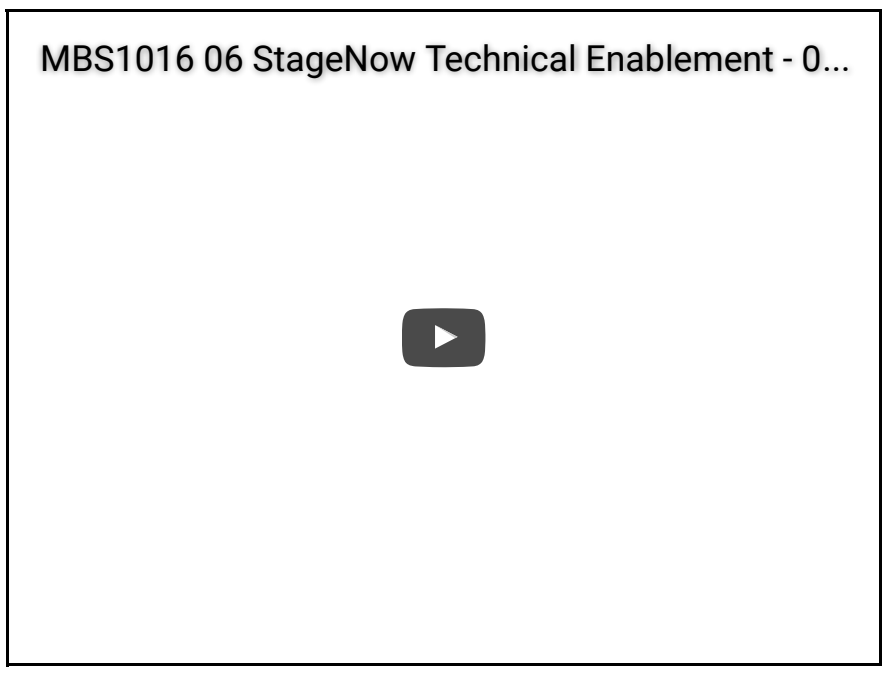
Posted by Arsen Bandurian Sep 6, 2016

Hi, All.

If you have issues when deploying devices with StageNow - check this educational video created by Zebra Knowledge Center
It explains typical issues they might face and how to address them as well as some workings of StageNow.

Please let us know what you think of it! Also, make sure you check out other videos in the series - there is a whole playlist!

https://youtu.be/HIHqXYVwFQ8



1357 Views Comments: 2 Permalink Tags: troubleshooting, stagenow, mbs1016

Zebra Phasing Out Support for Xamarin Studio with EMDK for Xamarin

Posted by Daniel Quagliana Aug 29, 2016

After the recent Xamarin acquisition, Microsoft has announced that Xamarin Studio is no longer supported on Windows. Microsoft now supports only Visual Studio for Xamarin development on Windows. In the next Zebra EMDK for Xamarin release [v2.2], Zebra will discontinue the support for Xamarin Studio for Windows. However, Zebra will continue to support Xamarin Studio on Mac.

1. Why is Zebra ending support for Xamarin Studio on Windows?
Since Microsoft has ended support for Xamarin Studio on Windows, Zebra is also ending support for all versions of Xamarin Studio on Windows.

2. Then what are my option for developing Xamarin apps on Windows?
Visual Studio (2015 or higher) as this is the only IDE Microsoft and Xamarin are supporting for Windows.

3. I use Xamarin Studio on Mac. Is there any change to Zebra's support?
No. You can continue to use Xamarin Studio on Mac for application development.

4. Will Zebra continue to support Xamarin Studio on Windows for the previous versions of the EMDK?
No. Zebra has ended support for all EMDK versions for all Xamarin Studio versions on Windows.

947 Views Comments: 0 Permalink Tags: emdk, android, c#, xamarin, zebra_news

Announcing EMDK for Xamarin v2.1

Posted by Bill Hecox Aug 25, 2016

EMDK for Xamarin v2.1 Now Available

For details on this release, see the EMDK for Xamarin 2.1 About Page

Installing EMDK for Xamarin 2.1:

If you are not upgrading from EMDK for Xamarin 2.0, follow this Guide to get setup.

Upgrade to EMDK for Xamarin 2.1:

1. Open your preferred IDE (Visual Studio or Xamarin Studio)
2. Open your IDE's Extention or Add-in Manager
    - Visual Studio - Tools > Extensions and Updates
    - Xamarin Studio (MAC) - Xamarin Studio(menu) > Add-ins
    - Xamarin Studio (Windows) - Tools > Add-in Manager
3. Look for Upgrade notification, and Install

NOTICE:

- After the recent Xamarin acquisition, Microsoft has announced that the Xamarin Studio is no longer supported on Windows. Microsoft supports only the Visual Studio for Xamarin development on Windows. In the next EMDK for Xamarin release [v2.2], Zebra will discontinue the support for Xamarin Studio for Windows. However, Zebra will continue to support Xamarin Studio on Mac.
- Xamarin Studio 6.0 is not currently supported.

Announcing EMDK for Android v5.0

Posted by Bill Hecox Aug 18, 2016

EMDK for Android v5.0 is now available. Find installers for this and previous releases on the EMDK Downloads page.

What's New

EMDK for Android v5.0

- The EMDK support for ADT and Eclipse is terminated. The supported development tool now is Android Studio. Therefore all the existing Eclipse and ADT projects must be migrated to Android Studio. Please refer to the Google documentation Migrating to Android Studio for an overview of the migration process.
- Enhanced the EMDKManager > ProfileManager to support simultaneous usage in multiple applications.
- Added support for the MX v6.0 in the Profile Manager:
  - Clock
    - Added new feature to manage Auto Time Zone - whether to automatically acquire time zone from the network.
    - Added new feature to manage Military Time - whether to use Military (24 hour) time format.
    - Updated to allow Manual Time Zone setting when Auto Time is On.
  - Camera Manager - Some of the latest devices can now be used to take pictures using Imager. The Camera manager is enhanced to block the Imager from taking pictures.
  - Analytics Manager - Added new capability to enable or disable features such as File Upload, ANR (Application Not Respond) Info Collection, Ruggedness Info Collection, Feature Usage Info Collection, Restrict SelfUpdate WiFi Only, Device Info Collection and custom feature.
- Enhanced the Notification Manager APIs with the following features:
  - Added support for using the pluggable External Vibrator with WT6000 device. The earlier version of EMDK Notification Manager API supported only RS6000.
  - Added new enum value "EXTERNAL_VIBRATOR1" to NotificationManager.DeviceIdentifier to specifically select External Vibrator.
  - Added new enum value "VIBRATOR" to DeviceType for External Vibrator.
  - Added new enum value "PLUGGABLE" to ConnectionType for External Vibrator. Enhanced the DeviceInfo class for determining the notification capability of RS6000 and External Vibrator.
  - Added new method isLEDSupported() to determine the support for Line Of Sight LED.
  - Added new method isBeepSupported() to determine the support for Beeping.
  - Added new method isVibrateSupported() to determine the support for Vibration.
  - Enhanced the NotificationDevice class for canceling the active notifications on the remote device.
  - Added new method cancelNotification().

For more details on this release, see the EMDK for Android 5.0 About Page

710 Views Comments: 0 Permalink Tags: emdk, android, java, zebra_news

Integrating DataWedge into your Cordova application

Posted by Darryn Campbell Aug 4, 2016

Update - April 2017

It has come to my attention that since this blog was first posted a year ago, there are several issues with the solution:

- The 3rd party plugin that I am using to send broadcast intents has been removed from GitHub
- The 3rd party plugin that I am using to receive barcode (startActivity) intents has been marked as no longer maintained
- The solution here does not take account of the new DataWedge APIs introduced in DataWedge 6.2.

I am working on these issues and hope to update this application in the near future.

DataWedge Intent Interface

DataWedge is a value-add of all Zebra Technologies devices (formally Symbol and Motorola Solutions) that allows barcode capture and configuration without the need to write any code. This application will demonstrate how to use Android intents to add DataWedge scanning functionality to your application

Quick Start - Getting this Demo running

```
>git clone https://github.com/darryncampbell/DataWedgeCordova.git
>cd DataWedgeCordova
>cordova platform add android
Plug in Zebra device
Follow instructions under "Configuring Datawedge" (below)
>cordova run android
Scan a barcode
```

Getting Started

This section walks through the steps to create a new Cordova application that utilises DataWedge for scanning.

Create a cordova application that will run on Android

```
cordova create DataWedgeCordova com.zebra.datawedgecordova DataWedgeCordova
cordova platform android
```

We will use a couple of 3rd party plugin to handle sending and receiving Intents to the DataWedge service. Any plugins capable of sending or receiving generic intents and interpreting the extra bundle into JSON will work:

For receiving intents:

```
cordova plugin add https://github.com/napolitano/cordova-plugin-intent#v0.1.31
```

For sending intents:

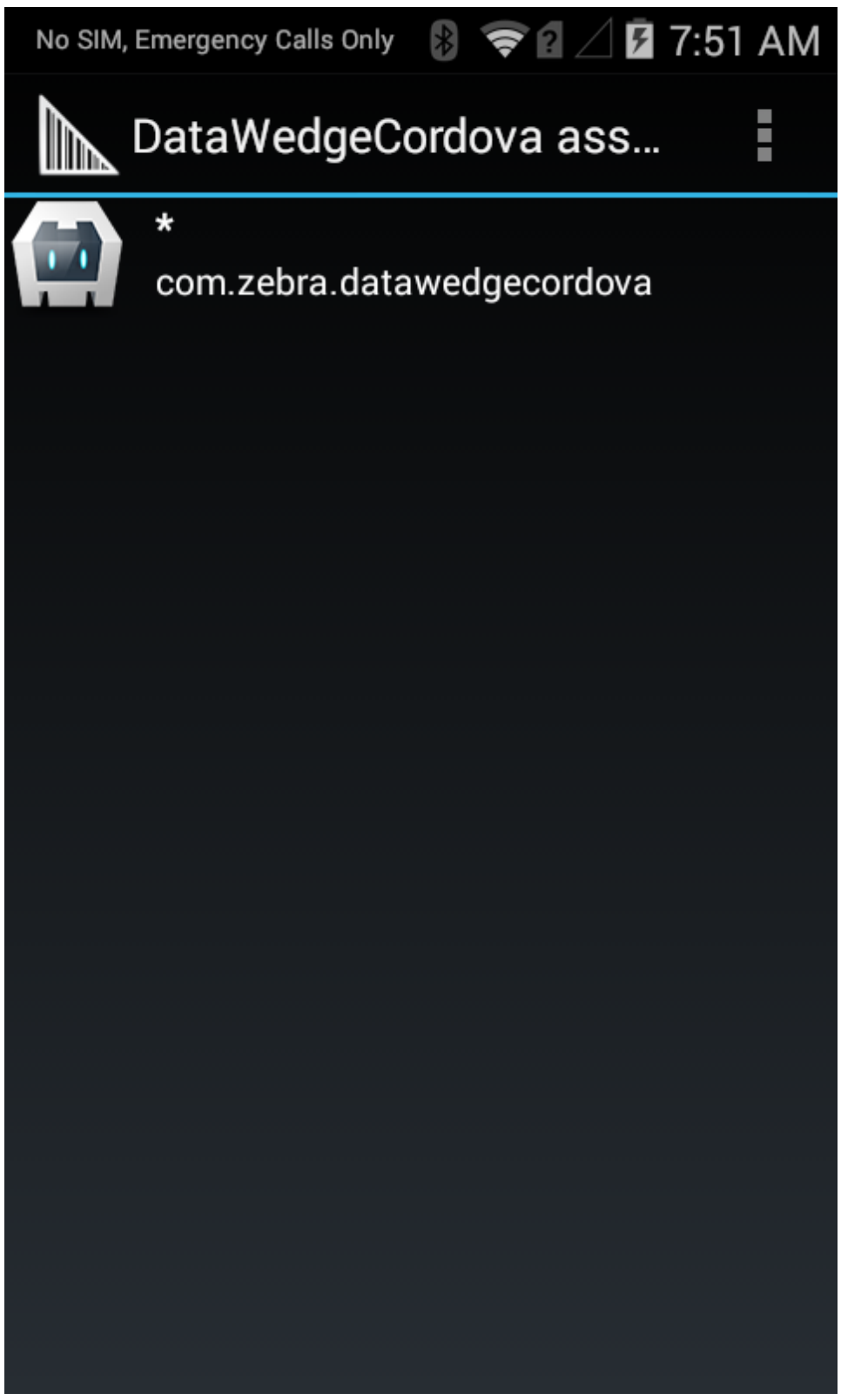
```
cordova plugin add https://github.com/Initsogar/cordova-webintent.git
```

The author of the plugin we'll use to receive intents very helpfully added parsing of intent extras in the 1.31 release so we'll explicitly grab that version.

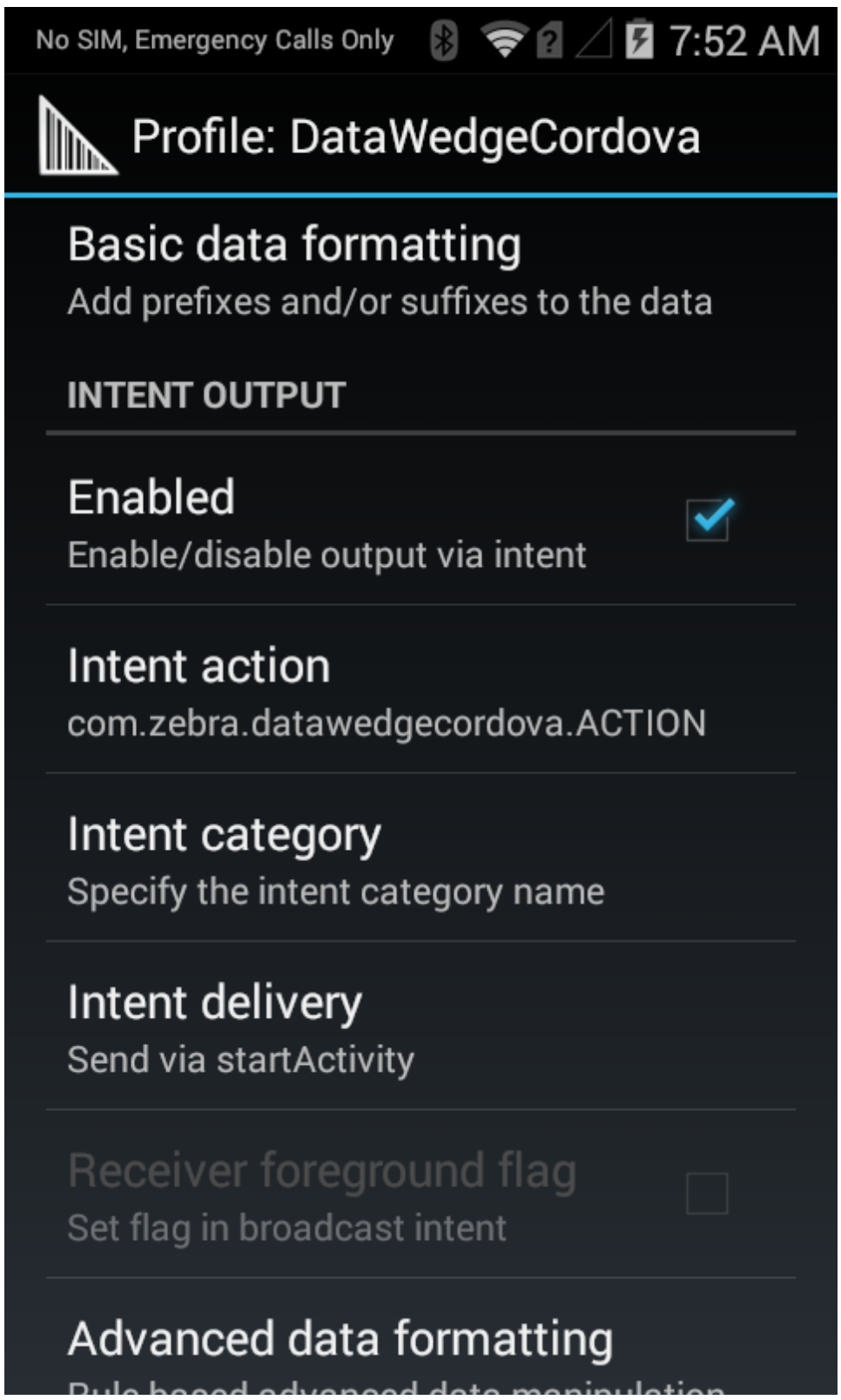
Configuring Datawedge

DataWedge can be configured to send intents whenever it scans barcodes. More detailed help is available at the official documentation but for the purposes of this demo I will include the pertinent steps. There are more sophisticated ways to configure DataWedge but this section will cover the basics.

- Create a new DataWedge profile (Applications --> DataWedge --> Menu --> New Profile). This will be the profile that will be active when our Cordova application is in the foreground. Give it a name e.g. DataWedgeCordova and click into it to configure.
- The next step requires our Cordova application to have previously run on the device so if you have not already done so, `cordova run android`
- Back in DataWedge configuration associate the Cordova application with our DW profile



- Scroll down to the Intent Output section of DW configuration and enable intents to start our activity:
  - Intent Action: com.zebra.datawedgecordova.ACTION
  - Intent Category: leave blank
  - Intent delivery: Start Activity



Add Intent Filter to the Cordova Application

By default our 3rd party plugin handling intents will not know to listen for intents carrying com.zebra.datawedgecordova.ACTION. There are multiple ways to configure our Cordova app to listen for this intent, we could create a custom plugin whose config.xml contains the intent-filter or we could get more complicated and dynamically register a broadcast listener in our plugin (assuming the DW intent delivery is modified to broadcast of course).

The simplest technique for the purposes of this demonstration is to just manually insert the listener into our AndroidManifest.xml. This is not ideal as the manifest will be overwritten if we remove and re-add our Android platform but for most purposes this is fine. To make this demo more reliable I have added a build hook to copy a predefined AndroidManifest.xml during the build process: DataWedgeCordova\platforms\android\AndroidManifest.xml:

```
        <activity ... android:launchMode="singleTop" ... >
            <intent-filter android:label="@string/launcher_name">
            ...
            </intent-filter>
  <intent-filter>
                <action android:name="com.zebra.datawedgecordova.ACTION" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>`
        </activity>
```

Add Code to the Cordova application

Now to hook up our logic to listen for and send intents.

Listening for intents

For sake of brevity modify the onDeviceReady function to call out to our 3rd party intent plugin and register a handler for new Intents. Assuming this is a DataWedge intent, process the data and display the barcode on the screen:

```
    onDeviceReady: function() {
        app.receivedEvent('deviceready');

        window.plugins.intent.setNewIntentHandler(function (intent) {
            console.log('Received Intent: ' + JSON.stringify(intent.extras));
            var decodedBarcode = intent.extras["com.symbol.datawedge.data_string"];
            var parentElement = document.getElementById('barcodeData');
            if (parentElement && decodedBarcode)
            {
                parentElement.innerHTML = "Barcode: " + decodedBarcode;
                parentElement.setAttribute('style', 'background-color:#0077A0;color:#FFFFFF;');
            }
        });
    },
```

And in index.html:

```
<div class="event" id="barcodeData"></div>
```

Sending intents

DataWedge supports an intent based API documented here . The API is somewhat limited but supports initiating the scanner via software (simulating a trigger press), disabling the scanner entirely and various DataWedge profile operations. We'll be adding features to our application to simulate a trigger press and disable / enable the scanner through that Intent interface:

To simulate a trigger press:

```
window.plugins.webintent.sendBroadcast({
    action: 'com.symbol.datawedge.api.ACTION_SOFTSCANTRIGGER',
    extras: {
        'com.symbol.datawedge.api.EXTRA_PARAMETER': 'START_SCANNING'
        }
    },
    function() {},
    function() {}
);
```

To disable the scanner:

```
window.plugins.webintent.sendBroadcast({
    action: 'com.symbol.datawedge.api.ACTION_SCANNERINPUTPLUGIN',
    extras: {
        'com.symbol.datawedge.api.EXTRA_PARAMETER': 'DISABLE_PLUGIN'
        }
    },
    function() {},
    function() {}
);
```

And hook up our logic to the UI:

```
document.getElementById("scanButton").addEventListener("click", startSoftTrigger);
document.getElementById("disableScanningButton").addEventListener("click", disableEnableScanning);
```

Now deploy & launch your app. You are able to scan barcodes and exercise the functionality:

Notes

The activity launchMode needs to be set to 'singleTop' in Cordova, this can be achieved by setting `<preference name="AndroidLaunchMode" value="singleTop" />` though in my experience the lauchMode is singleTop by default. This will ensure that each scan does not launch a new instance of the application.

Feedback

This technique for adding scanning capabilities to a Cordova application represents the most generic possible solution. We appreciate your developer feedback:

- Would you prefer a dedicated Cordova plugin for Javascript?
- Would you like to access EMDK profile functionality through Cordova?
- Would you rather a dedicated Javascript interface as opposed to relying on third party plugins?
- Are you interested in JavaScript development outside of Cordova e.g. ReactNative or NativeScript for your Enterprise application

Please feel free to raise github issues on the repository or post comments to this blog.

1103 Views Comments: 0 Permalink
Tags: dev, datawedge, phonegap, hybrid, dev_tools, data_wedge, cordova, hybrid mobile application - cordava js - phonegap, ionic

Mobile POS demo with NFC tap & pair and Bluetooth receipt printing

Posted by Ian Hatton Jul 13, 2016

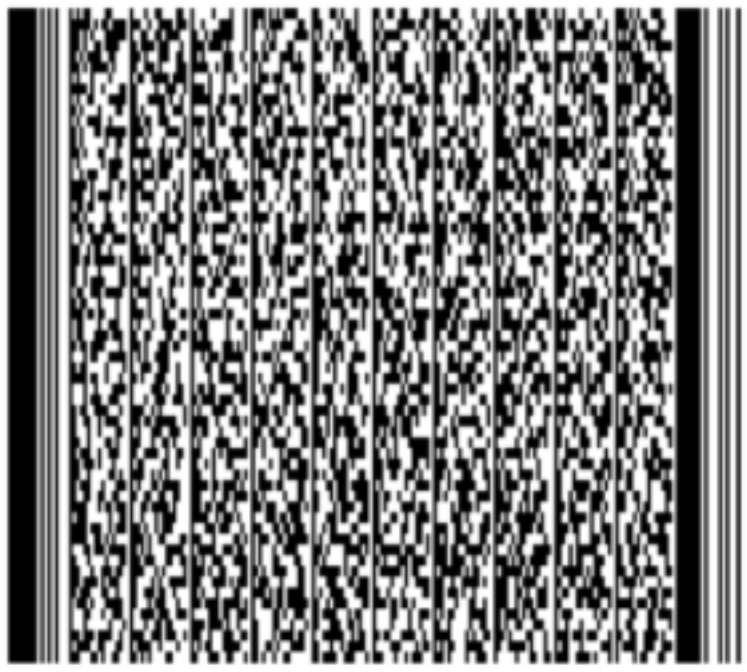
Zebra MPOS mobile POS demo with NFC tap & pair and Bluetooth receipt printing

Ian Hatton

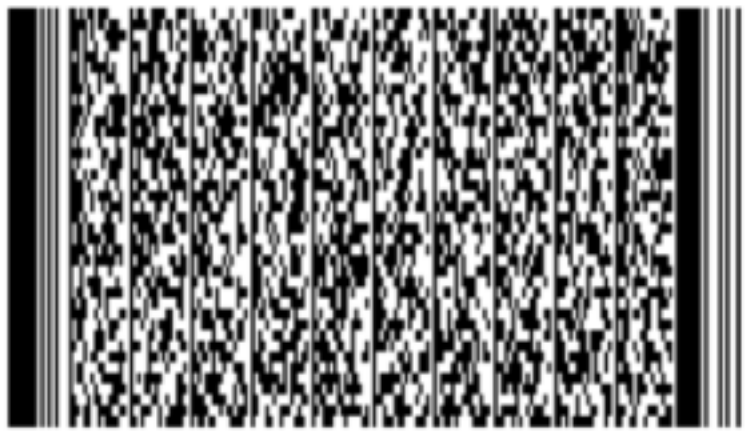
Zebra Technologies EMEA June16

Install required apps via StageNow barcodes below (note that device requires a pre-configured internet conn

1

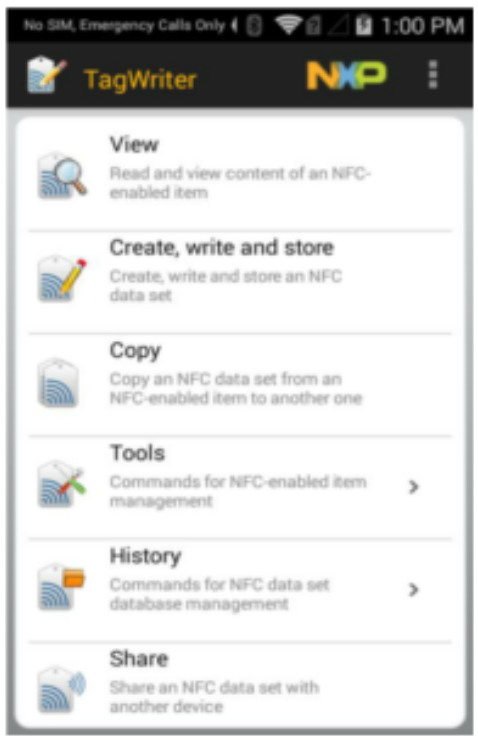


2

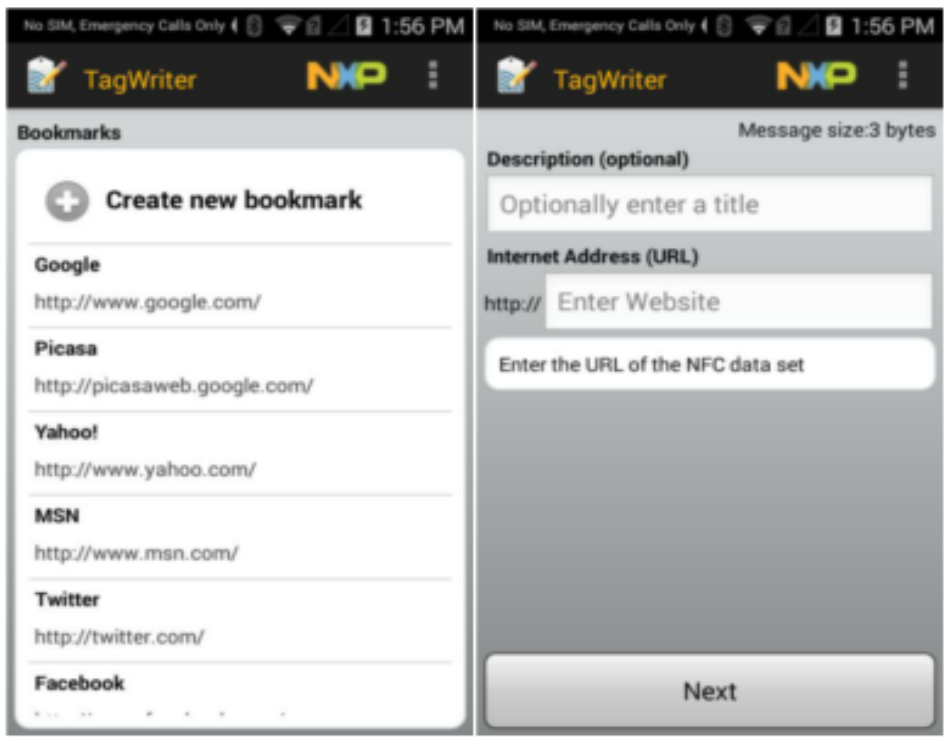


Create NFC pairing tag with NFC TagWriter (only required for Bluetooth printers without an embedded NFC tag)

1. Run NFCTagWriter. Select Create from the menu



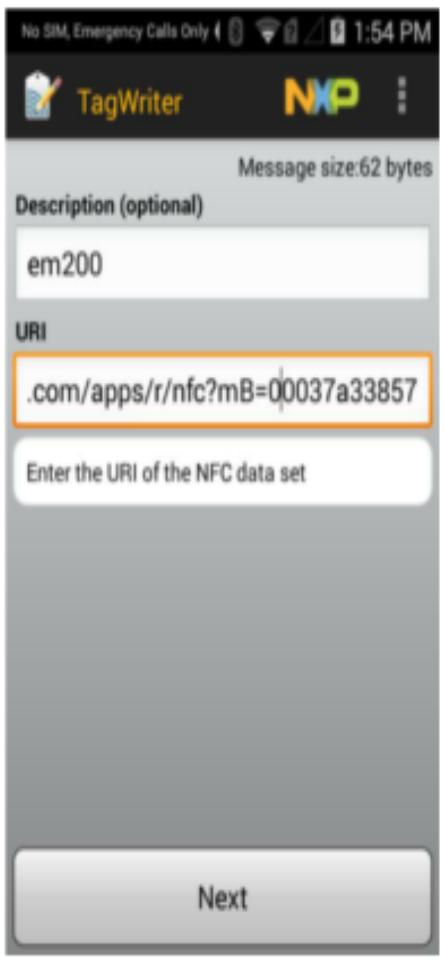
2. Select Bookmark from the content type list and then Create new bookmark



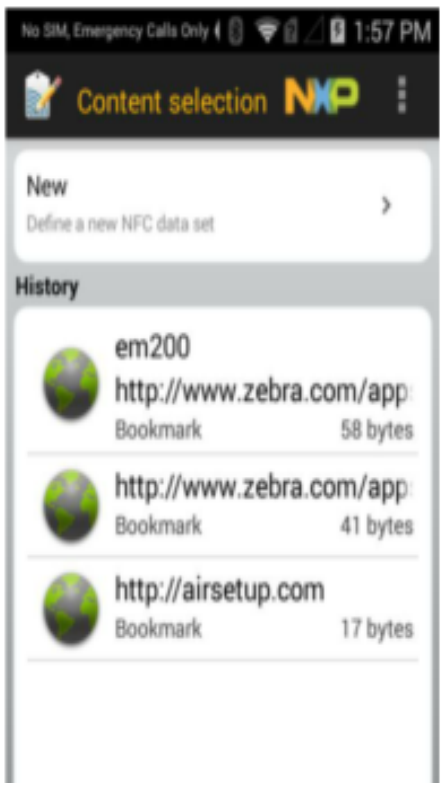
3. Enter a description for the profile and then the URL in the format

www.zebra.com/apps/r/nfc?mB=xxxxxxxxxxxx

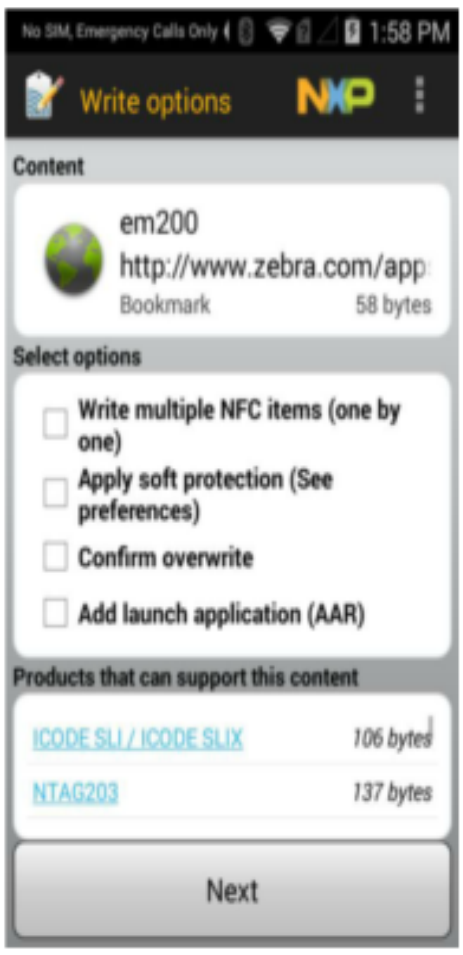
where xxxxxxxxxxxx is the Bluetooth MAC address of the printer



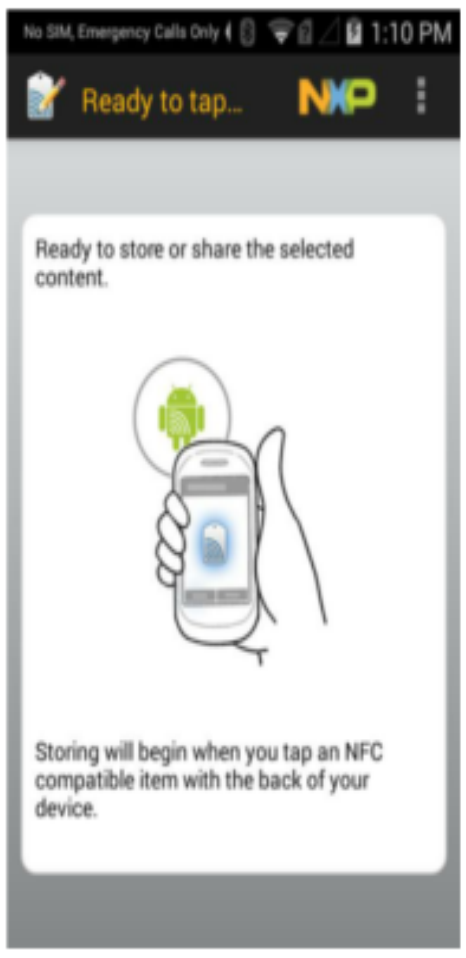
4. Press Save and then select the profile from the list



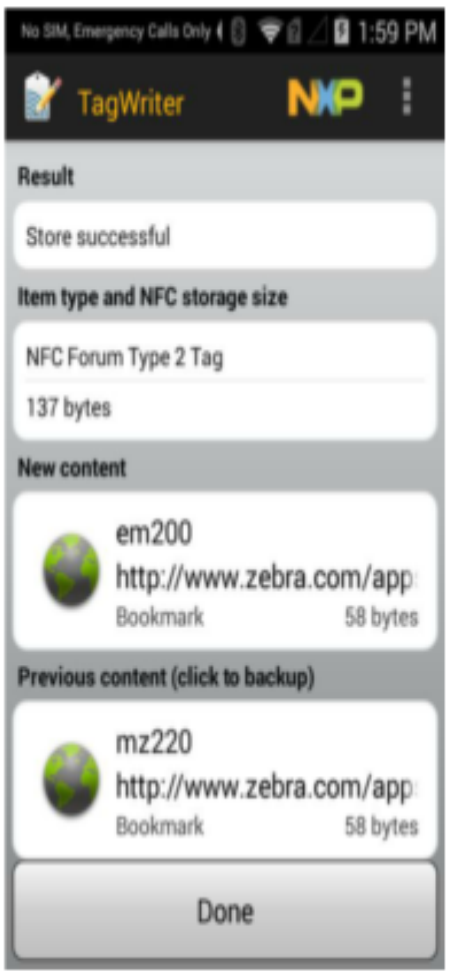
5. Leave write options to default and press Next



6. At Ready screen below, hold the device against the NFC tag to program the tage

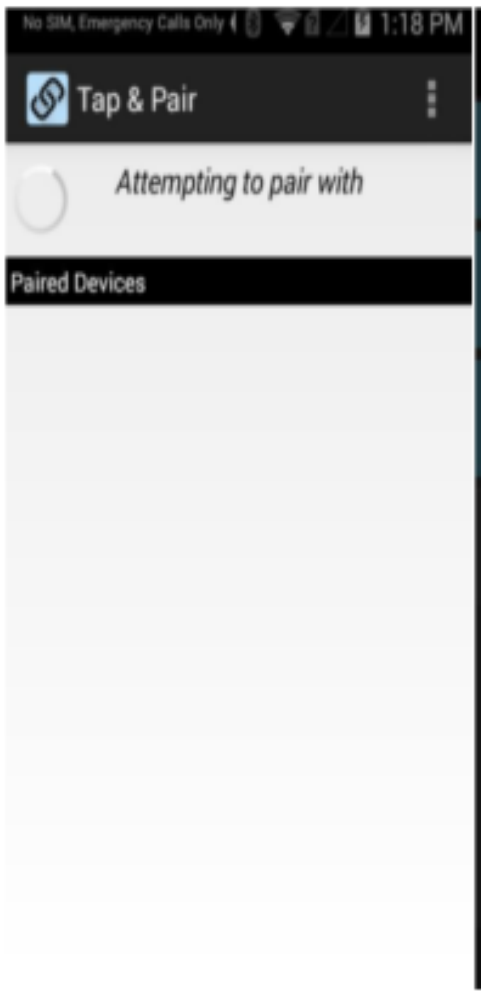


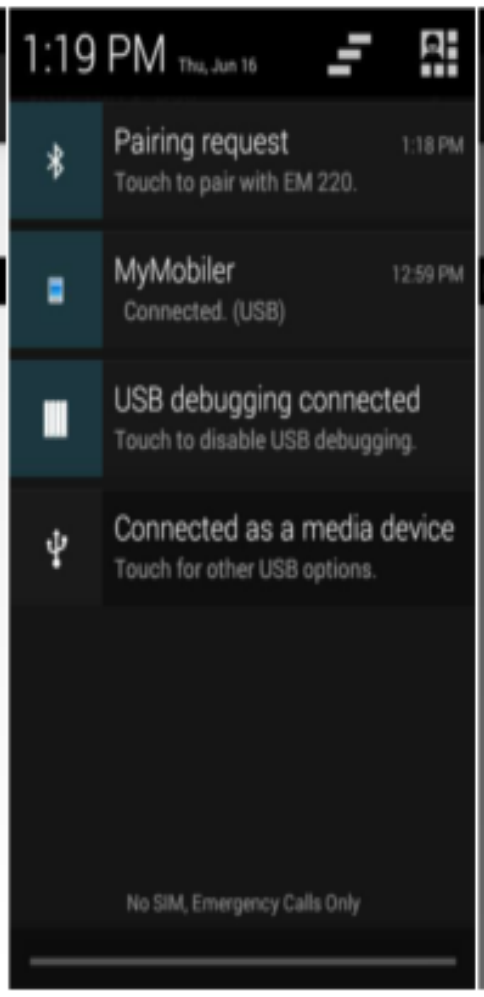
7. If tag is written correctly , store successful screen below will appear. Attach programmed tag to printer.

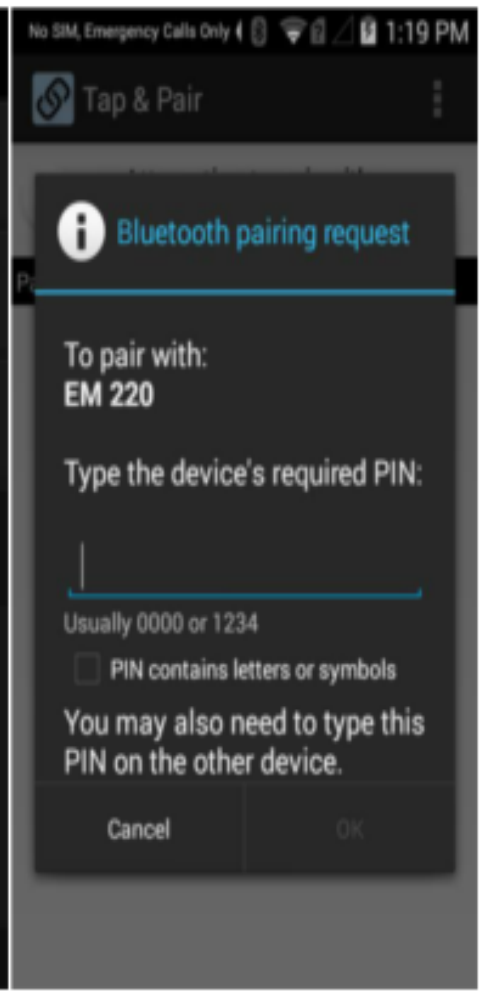


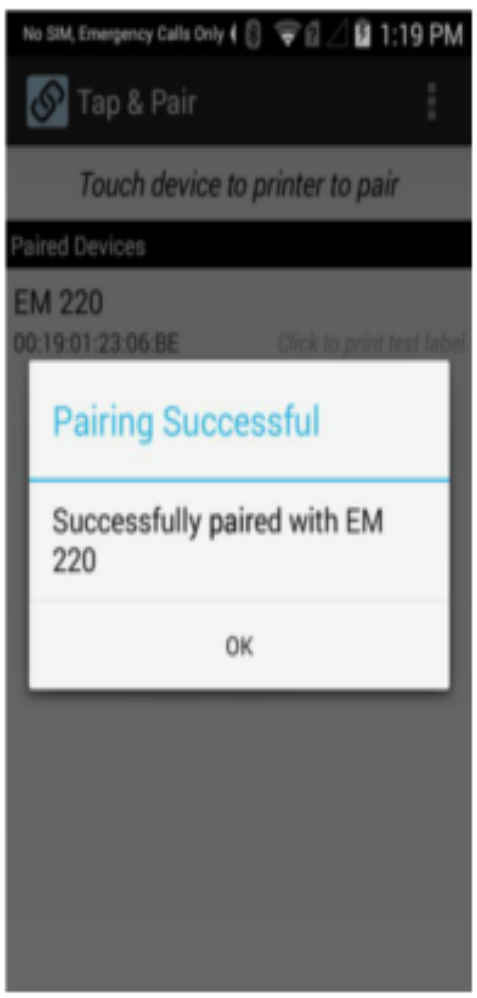
8. Run Tap & Pair application

9. Ensure printer is powered on, Bluetooth is enabled on device. Touch device to NFC tag on printer to pair



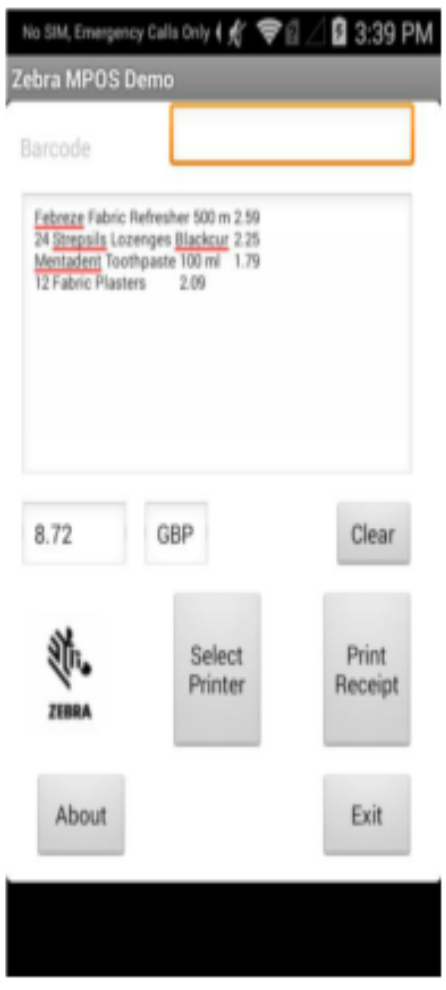




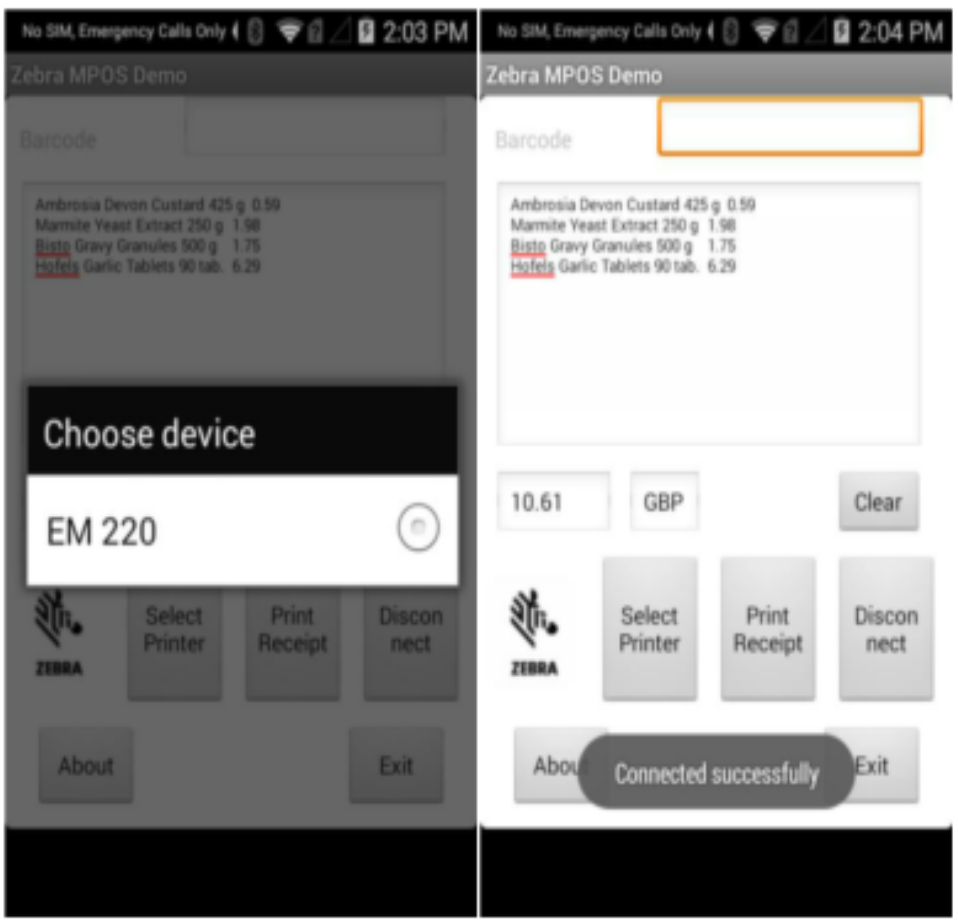


10. Run PrintScanDemo application

11. Scan barcodes from default barcode sheet. Note that the Datawedge profile used with this app (by default Profile0) should be manually modified to Send a Tab key after the barcode data (Keystroke Output/Basic Data Formatting/check Send TAB Key option) in order to accept the barcode in the app.

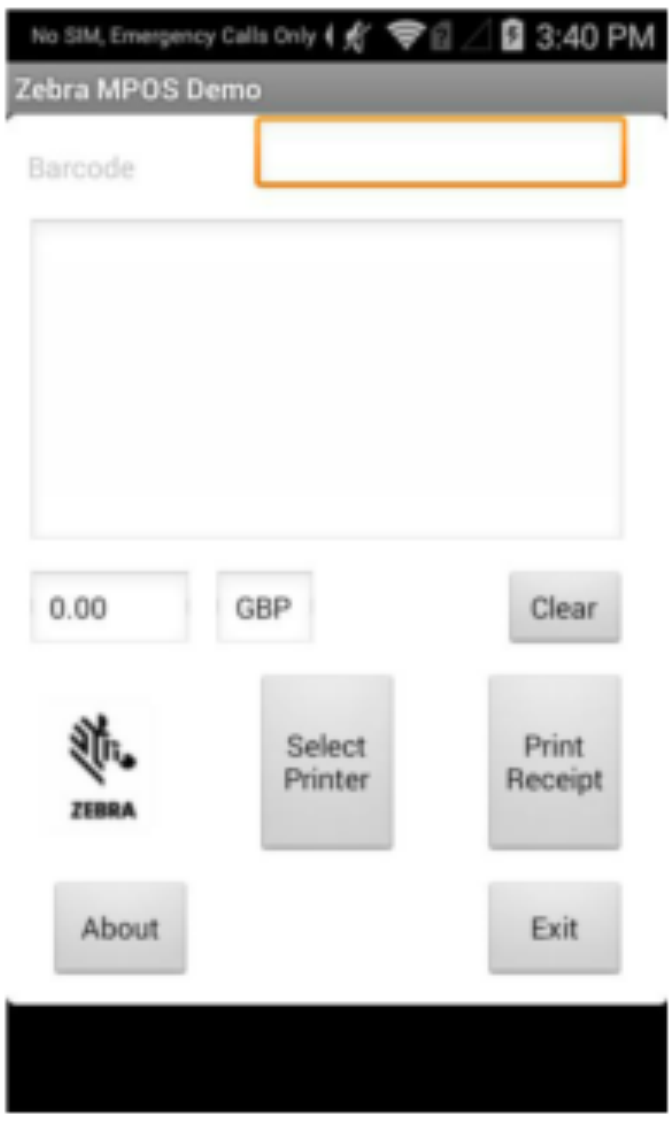


12. Ensure printer is powered on. Press Select Printer to show list of paired printers and select printer to use



13. Select Print Receipt and press OK to confirm. Receipt should print.

14. Press Clear to start new transaction



756 Views Comments: 0 Permalink

EMDK Now supports the WT6000

Posted by Bill Hecox Jul 1, 2016

Following yesterday's announcement of the release of the WT6000 . We would also like to announce that EMDK for Android 4.2 and EMDK for Xamarin 2.0 now support the WT6000 Industrial Wearable Computer!

711 Views Comments: 0 Permalink Tags: emdk, android, zebra_news

Announcing EMDK for Android v4.2

Posted by Bill Hecox Jun 28, 2016

EMDK for Android v4.2 is now available. Find installers for this and previous releases on the EMDK Downloads page.

Whats New in v4.2

Profile Manager:

- Added support for the MX v5.1

- UI Manager
  - Added new feature to set the language to localize the device to match the preferred language of the intended Device User.
  - Added new feature to enable or disable pulling down the Notification Panel.
  - Added new feature to display or hide the Settings icon in the Notification Panel.

- App Manager - Added new feature to launch an application by specifying the application name.

- Host Manager - Added new feature to set the device host name to identify device both locally and within any DNS-enabled IP-based network.
- Bluetooth Manager - Added new feature to allow or disallow the mobile device to pair with new remote devices.
- Analytics Manager - Added new capability to control the data captured by the analytics engine such as data transportation, type of information to collect and data collection scheduling.
- Updated the DataCapture feature:
  - Added support for additional scanner devices:
  - Bluetooth Imager 1 (please note this value is for use with RS507 only)
  - Added support for controlling the barcode decode notifications:
  - Aim Mode - Enable/Disable scanner aim during scanning.
  - Illumination Brightness - Controls illumination brightness of the imager.

API:

- Updated Barcode Manager APIs with the following features:
  - Enhanced the ScannerInfo class for the selecting the scanner:
  - Added new method getDeviceIdentifier() for selecting the scanner from the supported scanner devices. This method provides the information returned by the getConnectionType() and getDeviceType() methods in one call.

For more details on this release, see the EMDK for Android 4.2 About Page

456 Views Comments: 9 Permalink Tags: emdk, android