

2017

Android Blogs

January 2017



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:

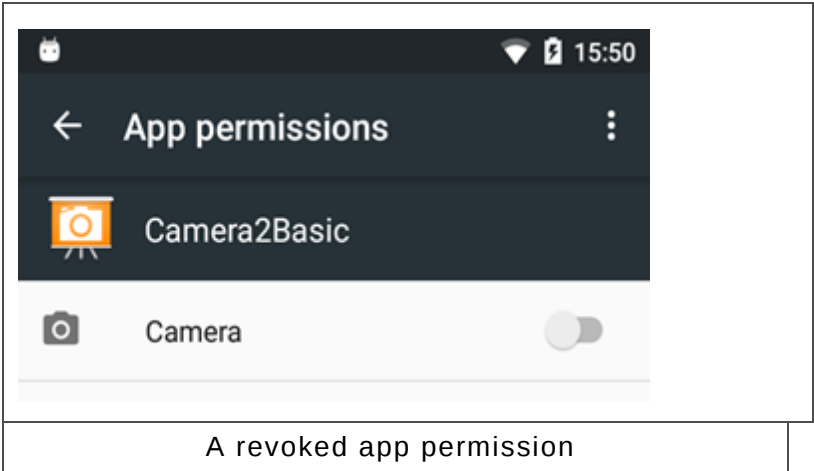
1. Continue to target API level 22 or lower with your application.

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\(java.lang.String\)](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().



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\)](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

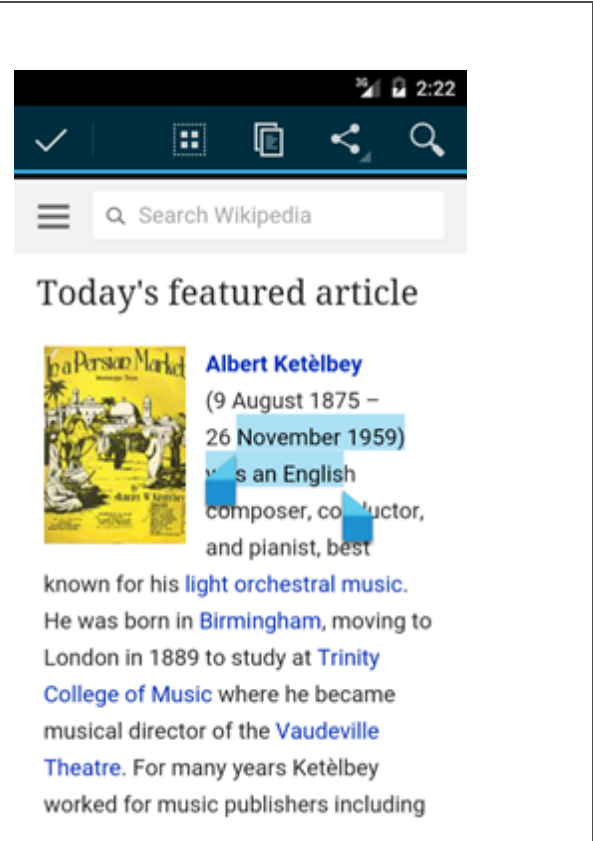
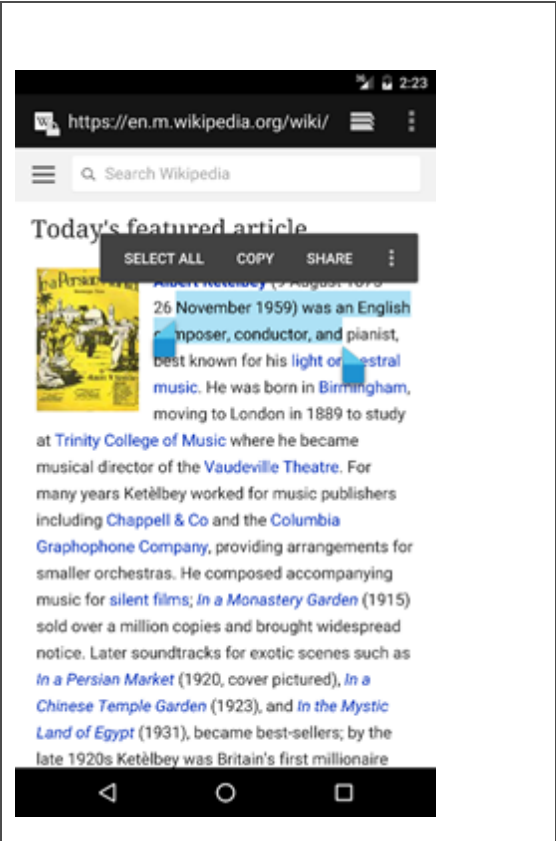
Google’s drive toward Android adoption in the Enterprise, formally known as “Android for Work” but now simply incorporated under the Android umbrella, is a suite of solutions from Google to help manage devices and applications, augment overall security, and support an increasing number of enterprise-focused 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\(boolean\)](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\(\)](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\(\)](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:

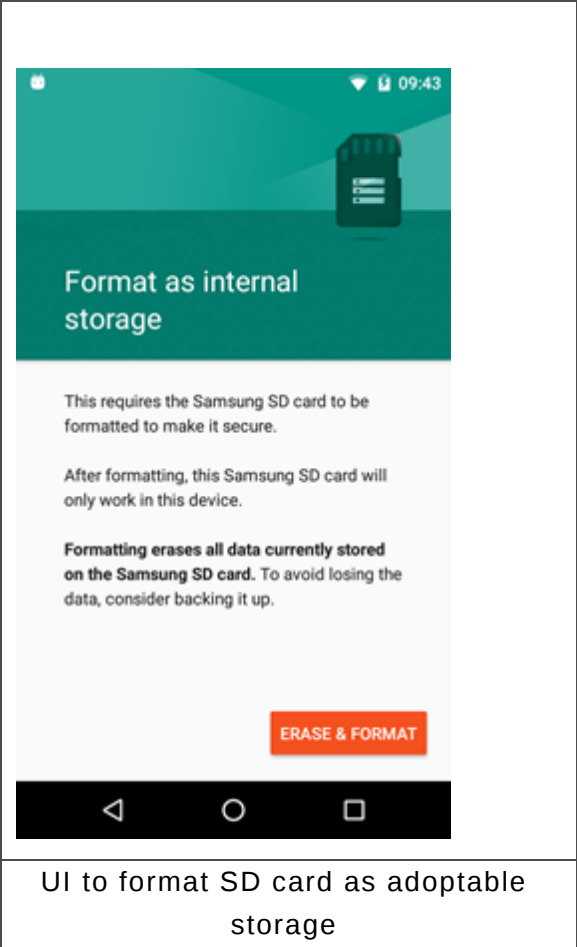
- Without having to separately encrypt the SD card:
 - Secure offline access to video tutorials for field mobility workers
 - Secure offline access to large product databases or additional application content

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.



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.

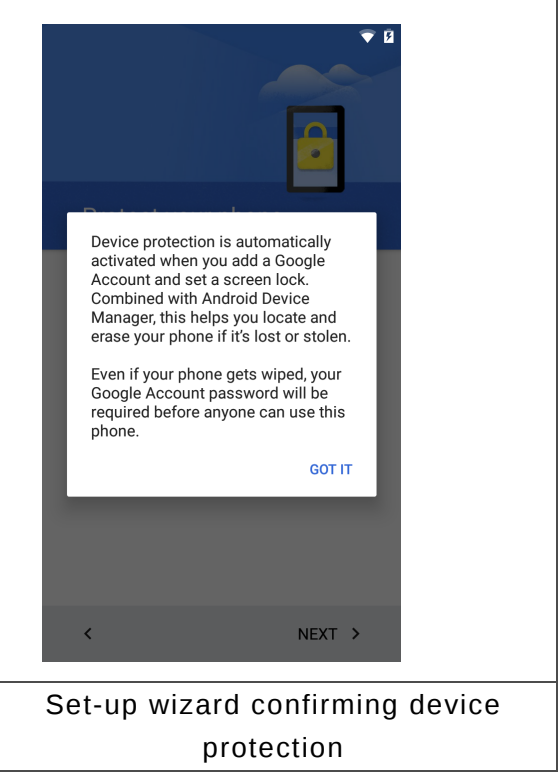
This technique also applies when flashing a new OS to your device, and is explained in the device quick start guide.

In addition to the manual method described above, performing OSUpdates via EMM or using StageNow's OSUpdate service has the option of using a fixed location to apply the update from, `/data/tmp/public`.

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.



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).

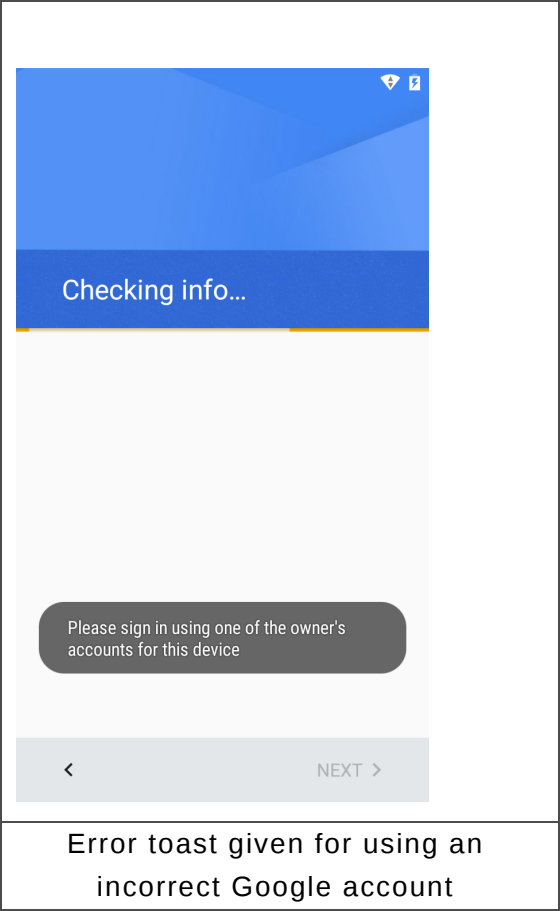
There are different types of reset for protected devices:

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

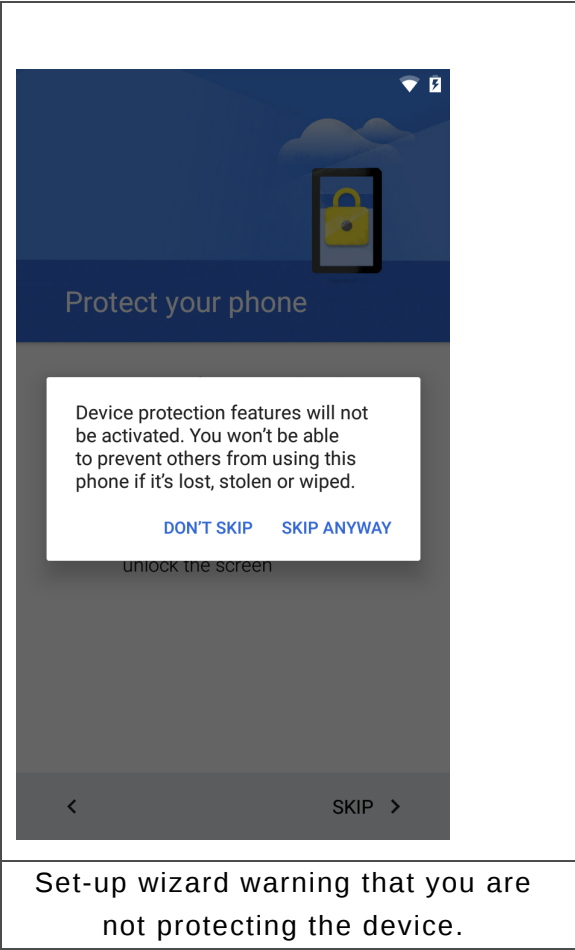
reset types invoked from the MX Power manager, as described above: Full Device Wipe, Factory Reset and Enterprise Reset.

- 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 your device will need to be returned to Zebra for repair, in which case 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:



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:

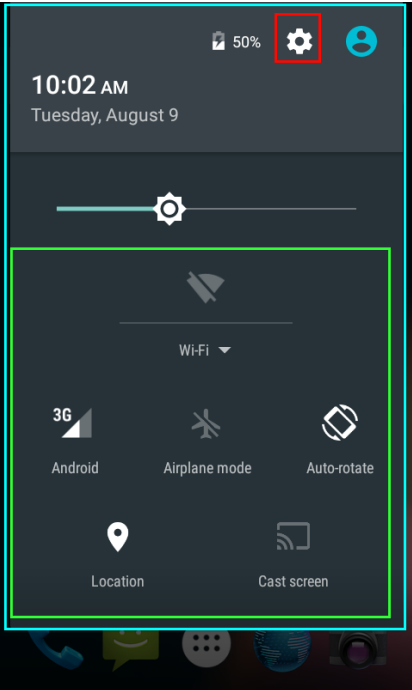
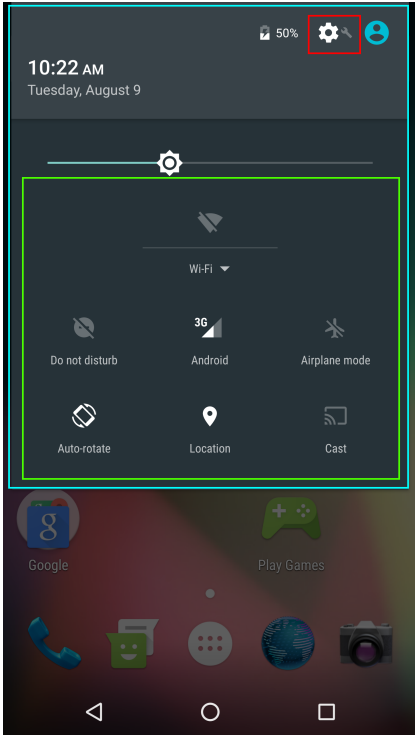


New notification shade and quick settings menu

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:

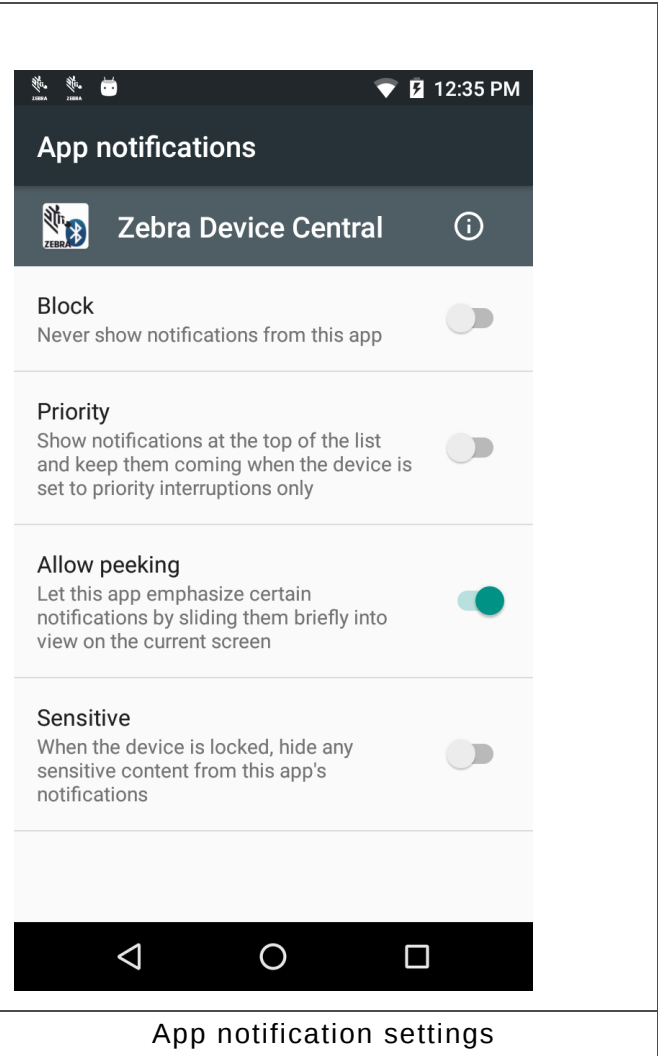
1. Disable the ability to long-press on the Settings gear in the first place by hiding it using the UI Manager's Notification (Icon) Settings feature (<http://techdocs.zebra.com/emdk-for-android/6-0/mx/uimgr/#gear-icon-in-notifications-panel-showhide>).
2. A less elegant approach would be to restrict 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".



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 extensions 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, the first 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

1685 Views [Comments: 0](#) [Permalink](#) Tags: [emdk](#), [marshmallow](#)