Porting a DA EMM Agent to a DO EMM Agent

This document is intended primarily for EMM Vendors that have an existing EMM Agent that currently augments its functionality using Zebra MX functions via XML submitted directly to MX using the mechanism documented in the Zebra EMM Tool Kit. While others might also find this material useful or interesting, this document provides theory and practical guidelines focused on and specifically targeted at enabling an EMM Vendor to port their existing "DA" solution to a new "DO" solution based on standard Android Enterprise Device Owner (AEDO) APIs augmented by Zebra OemConfig Managed Configurations.

The primary reason for an EMM embarking on such a porting is to move from the current situation, under which EMM Vendors are required to develop and maintain one or more OEM-proprietary EMM Agents to support devices from each OEM, to a more standards-based approach allowing a single generic EMM Agent to be used to manage devices from OEMs conforming to the same standards.

Background and Definitions

Before describing the theory and practice of porting from DA to DO, it is important to establish a baseline understanding of the starting state and the desired end state, and to define the specialized terminology involved.

- "Privileges"
  - Many of the functions traditionally required to implement an EMM are considered privileged by Android, and hence the APIs required to perform those functions cannot be called by an unprivileged application. Lack of access to such functions generally would result in a lack of critical functionality and a non-viable EMM solution.
- "Privilege Escalation"
  - A Privilege Escalation method is a means by which an application can be granted the functionality provided by privileged APIs to which it would otherwise be denied due to the privileged nature of those APIs.
  - Originally, the only Privilege Escalation method available in Android that could be used to grant applications the functionality of many critical APIs was granting the application direct access to those APIs by signing of the application by the device OEM using the platform key of the desired Android Device. While Zebra did sign some EMM Agents for some EMM Vendors, doing so created the potential for security risks since there is no practical way to revoke privileges from an EMM Agent once it is signed. This meant that Zebra had to carefully qualify an EMM Agent before agreeing to sign it and requalify it before resigning each time changes were made. This was a time consuming and costly process, for both Zebra and the EMM Vendors, and could not reasonably and safely scale beyond a small number of EMM Vendors.
  - Over time, Android added some additional Privilege Escalation methods, specifically Device Administrator (DA) Mode, then later Profile Owner (PO) Mode, and even later Device Owner (DO) Mode.
  - Zebra also added a Privilege Escalation method called MX, which was exposed and documented for use by EMM Vendors through the Zebra EMM Tool Kit (EMMTK).
- "DA" = Device Administrator Mode
  - Device Administration is a standard Android Privilege Escalation method that became available quite early, in Android 2.2.x (Froyo) API Level 8 and which has been expanded somewhat in later versions.
  - Being a DA allows an application to be granted the right to use selected methods of the Android Device Administration API, exposed primarily via methods of the Android DevicePolicyManager class.

  - Unfortunately, the methods of the Android Device Administration API that can be used by a DA provide minimal functionality and alone do not enable the development of a viable EMM solution.
- To be eligible to be a DA, an application must be developed to be a Device Policy Controller (DPC), which means it must implement a sub-class of the Android DeviceAdminReceiver class.
- To become a DA, an application must create an instance of its DeviceAdminReceiver sub-class and request that Administrator status be granted to that instance by the Device User.
- When DA status is granted by the Device User, the instance of the DeviceAdminReceiver sub-class in the application will be notified, after which the application will be able to use that instance to access the subset of the methods of the Android Device Administration API designated as available to a DA.
  - At any time, the Device User could elect to revoke DA status from an application to which it was previously granted, at which point the application would lose access to the methods of the Android Device Administration API until the Device User elected to again grant DA status to that application.
  - Via MX, Zebra also provided a mechanism by which an EMM Agent could grant itself DA status instead of having to request it from the Device User. Use of this MX mechanism did not prevent a Device User from revoking DA status, but an EMM Agent could—when detecting a loss of DA status—use MX to again grant DA status to itself.

❖ "PO" = Profile Owner
- Profile Owner is a standard Android Privilege Escalation method that became available in Android 5.0 (Lollipop) API level 21 and expands on the capabilities of DA. Like a DA, a PO must be a DPC. In fact, every PO is also a DA (but not every DA is a PO).
- An application that is eligible to become a DA (and that meets certain additional requirements) can be promoted to a PO through a special enrollment process that requires the Device User to opt in to having a Work Profile and having that Work Profile (but NOT the rest of the device) managed by that PO, which is usually under the control of an employer.
- Once an application becomes a PO, it gains access to an expanded set of Android Device Administration APIs not available to it as a DA.
- A PO is generally used to implement a Work Profile as part of a "Bring Your Own Device" (BYOD) deployment scenario. Under such scenarios, a Work Profile contains the "Work" part of the device, which is managed by a PO selected by the Enterprise, and is segregated from the "Personal" part of the device, which is kept private from and cannot be managed by the Enterprise.
- Through Android Device Administration APIs, a PO is granted broad control over the Work Profile on the device, but only a small amount of control over selected global aspects the device. This makes sense because the device is generally owned by the Device User and NOT by the Enterprise that is managing the device. An example of a Profile Owner controlling global aspects of a device would be to configure the device to require a PIN or password of some minimum strength, which would allow the Profile Owner to protect against access to the managed Work Profile by unauthorized users, but would also affect personal use of the device.
- While the set of Android Device Administration APIs to which a PO has access is well beyond those granted to a DA, they generally are insufficient to enable the development of a viable EMM solution except in the special case of a BYOD deployments.

❖ "DO" = Device Owner
- Like a DA, a DO is a DPC, and every DO is a DA. There can be only one DO on a device at any given time.
- An application that is eligible to be a DA (and that meets certain additional requirements) can be promoted to DO through a special enrollment process that requires the Device User to opt in and that must generally be performed early during the initialization process of the device. The Android

Compatibility Definition Document (CDD) requires that a DO be enrolled only when there is "no user data" present in the device. The primary reasons for this limitation are:
  - If a DO could be enrolled at any time, the chance of enrolling an unwanted DO (and thereby granting it unwarranted control over the device) would be significantly increased.
  - If a DO could be enrolled in conditions under which the device had been compromised via questionable configuration and/or the installation of questionable applications, then the DO might not be able to detect or correct such compromise, leaving the device vulnerable to misuse or attack.
  - By ensuring that a DO can be enrolled only under tightly controlled circumstances, a DO can be confident that the device was not compromised and can actively prevent subsequent attempts to compromise or attack the device.
- A DO is generally used to implement device management as part of a "Corporately Owned Special Use" (COSU) device deployment.
- Through the Android Device Administration APIs, a DO is granted broad control over many aspects of the device, because the device is generally owned by the Enterprise that is managing the device and NOT by the Device User.
- The set of Android Device Administration APIs to which a DO has access is well beyond those granted to a PO or DA; they generally are sufficient to enable the development of a viable EMM solution for many COSU deployments.
  - Nonetheless, there remain some functions that customers might deem highly important for a "complete" EMM solution to offer for Zebra Android devices, but that are beyond what a DO can do solely using Android Device Administration APIs.
    - Some of these might be standard Android functions that have been or will be added to the Android Device Administration APIs over time. But if the requisite functions are not available some Android version, they could be a barrier to customer acceptance of an EMM Solution that attempts to manage devices with that Android version based solely on an EMM Agent that is a DO.
    - Some of these can be OEM-proprietary functions that might never be added to the Android Device Administration APIs. If the requisite functions are not available to the EMM Agent on a given device, then they could be a barrier to customer acceptance of an EMM Solution that attempts to manage devices based solely on an EMM Agent that is a DO.

❖ "MX" = Zebra Management Extensions
- MX is a Zebra-proprietary Privilege Escalation method designed to enable an unprivileged EMM Agent to access all "missing" functions needed to enable an EMM vendor to implement a "complete" EMM Solution for managing Zebra Android devices.
- MX was developed concurrently with the availability of the DA model; it predates the availability of PO and DO. As such, MX was originally designed to augment and extend the capabilities of any EMM Agent. In most cases, EMM Agents that used MX fell into one of two categories:
  - The EMM Agent could have been a DA that used MX to augment the functionality provided by the Android Device Administration APIs to which it had access by becoming a DA.
    - The main advantage to an EMM Agent being a DA in addition to using MX was the ability to use the Android Device Administration APIs to control password policies, functionality that MX did not provide.
    - For DA EMM Agents, MX also provided a way to automate granting of DA status, which the EMM Vendor could use to simplify deployment by eliminating the need for manual action by the Device User to grant DA status to the EMM Agent.

- The EMM Agent could have been a non-DPC application that was not eligible to become a DA and that used MX to provide all privileged functionality it required to manage the device.
  - As noted above, such an EMM Agent would not be able to use the Android Device Administration APIs to control password policies, functionality that MX did not provide. Such an EMM Agent would thus lack the ability to provide such control, which wasn't always a critical customer requirement.

❖ "ZMC" = Zebra Managed Configurations are Managed Configurations exposed by the Zebra OEM Configuration Application (OemConfig).

- To facilitate porting to DO mode and the removal of all Zebra-proprietary logic from an EMM Agent, an alternative to MX was required. While MX accomplished the goal of providing EMM Agents with access to privileged and proprietary functionality, the proprietary nature of the MX interface is inherently incompatible with the goal of enabling an EMM Vendor to develop and maintain a single generic EMM Agent that could work on all Android devices from all OEMs, or at least all devices running a suitably recent version of Android.
- Luckily, the Android Device Administration APIs provide a generic extension mechanism called Managed Configurations (previously known as "Application Restrictions").
  - As the original name would suggest, this mechanism was originally conceived as a way for Android device applications to expose restrictions that could be imposed on them from outside, such as by an EMM. For example, the Chrome Web Browser for Android exposed the ability to change key behavioral aspects, such as whether to show the address bar and to store passwords.
  - Over time, Android recognized that applications might expose more than just restrictions, hence the name was changed to Managed Configurations, although the functionality was not altered.
- Zebra recognized that Managed Configurations represented an extension mechanism that could be used for more than just configuration of applications, specifically that configuration of a device could be accomplished indirectly through a configurable application.
  - Zebra defined a special application, called OemConfig that publishes Managed Configurations that control the behavior of the device rather than just the behavior of the application itself.
  - Zebra reviewed this approach with Google and with several key EMM Vendors and this approach has now been approved and is being promoted by Google as the recommended approach for use by OEMs to publish extended functionality and for use by EMMs to utilize extended functionality published by OEMs.

❖ "EMMTK" = Enterprise Mobility Management Tool Kit

- The EMMTK is a Zebra product designed to provide EMM Vendors with everything they need to develop an EMM Agent that can access all functionality to implement a "complete" EMM Solution for managing Zebra Android devices.
- Early versions of the EMMTK focused primarily on providing documentation, sample code, tools, and technologies required to leverage MX as a Privilege Escalation method to augment the functionality of an EMM Agent that was either totally unprivileged or that was using another Privilege Escalation, such as DA.
  - Multiple EMM Vendors successfully used the EMMTK to enable development of a Zebra-proprietary EMM Agent that provided the in-device support for a "complete" EMM Solution for managing Zebra Android devices.
  - Note that some EMM Vendors chose to augment a DA EMM Agent while others chose to augment a non-DPC unprivileged application.

- Upcoming versions of the EMMTK are de-emphasizing the use of MX and instead focusing on adapting to the Android Device Administration API (available to a DO), augmented with the Zebra implementation of OemConfig and its use of Managed Configurations.

❖ Full Support
- Full Support indicates that a feature is fully supported and generally recommended to be used without significant limitations or restrictions.

❖ Partial Support
- Partial Support indicates that a feature is supported and generally recommended to be used but can have some limitations or restrictions on when or on which devices it can be used, or in the functionality provided.

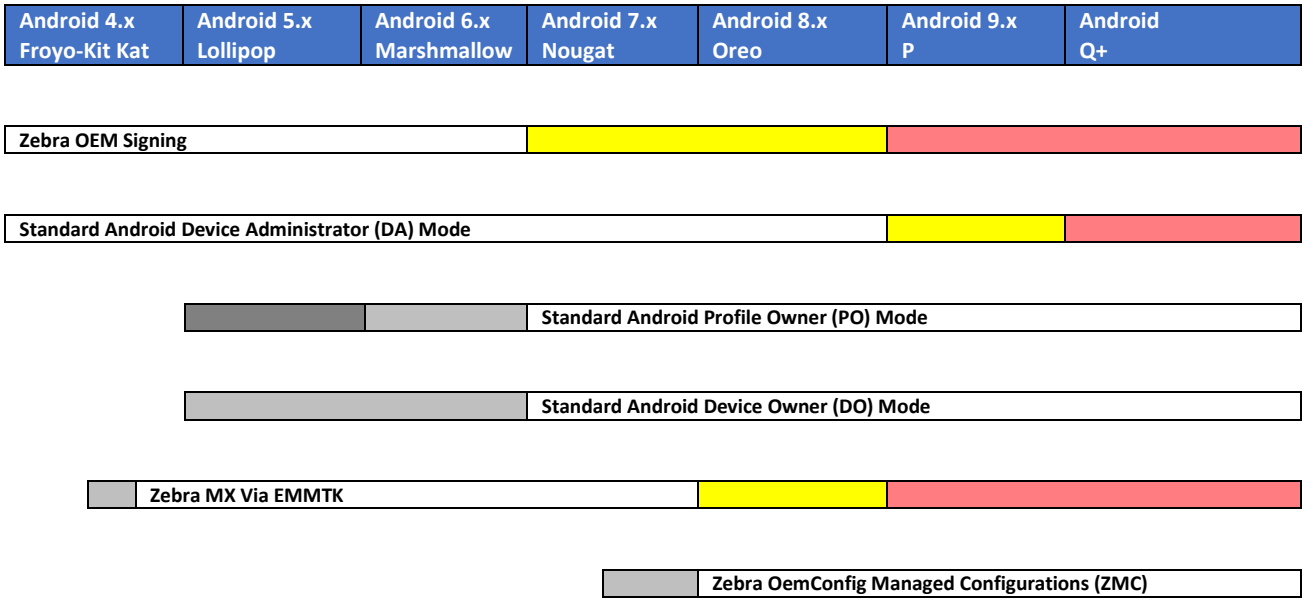
❖ Deprecated
- Deprecated indicates that a feature is being phased out and hence is still supported but is generally discouraged from being used in favor of a newer mechanism. Deprecation of a feature that previously had Full Support might also result in only Partial Support for that feature while it is being phased out.

❖ Discontinued
- Discontinued indicates that a previously deprecated feature is no longer available or supported and typically means that attempts to use that attempts to use that feature will fail gracefully. Continued use of the feature would generally require rebuilding to use the new mechanism that was previously provided when the feature was deprecated.

Summary of Privilege Escalation Methods

Method	Support	Notes
Zebra OEM Signed (ZS)	Limited to select Tier 1 EMMs Full Support from Android 2.3.x (Gingerbread) Full Support through Android 6.x (Marshmallow) Partial Support in Android 7.x (Nougat) Deprecated in Android 7.x (Nougat) Discontinued in Android 8.x (Oreo)	Only for use by approved Tier 1 EMMs • Allowed use of Standard Privileged Android APIs plus Zebra Proprietary OS Extension (OSx) APIs • Commonly used method to manage Zebra Android devices with Lollipop and older • Acceptable method to manage Zebra Android devices with Marshmallow • Discouraged method to manage Zebra Android devices with Nougat • Unsupported method to manage Zebra Android devices with Oreo or newer
Android Device Administrator (DA) Mode	Full Support from Android 2.2.x (Froyo) Full Support through Android 8.x (Nougat) Deprecated by Google in Android 9.x (Oreo) Discontinued by Google in Android Q	Can be used by all EMMs • Acceptable but insufficient (see ZMC) method to manage Zebra Android devices with Oreo and older
Android Profile Owner (PO) Mode	Full Support from Android 5.x (Lollipop) Full Support through Android 9.x (P) No current indications that this will be Deprecated or Discontinued	Can be used by all EMMs • Acceptable method to manage Zebra Android devices with Lollipop and later when used for BYOD deployments
Android Device Owner (DO) Mode	Partial Support from Android 5.x (Lollipop) Full Support from Android 7.x (Nougat) No current indications that this will be Deprecated or Discontinued	Can be used by all EMMs • Recommended method to manage Zebra Android devices with Nougat and newer when used for COSU deployments
Zebra MX Via EMMTK (MX)	Partial Support from Android 4.2.x (Jelly Bean) Full Support from Android 4.4.x (Kit Kat) Full Support through Android 8.x (Oreo) Deprecated by Zebra in Android 8.x (Oreo) Discontinued by Zebra in Android 9.x (P)	Can be used by all EMMs • Recommended method to use alone or to augment other methods to manage Zebra Android devices with Marshmallow and older • Acceptable method to use alone or to augment other methods to manage Zebra Android devices with Oreo and older
Zebra OemConfig Managed Configurations (ZMC)	Partial Support from Android 7.x (Nougat) Full Support from Android 8.x (Oreo)	Can be used by all EMMs • Recommended method to augment a DO to manage Zebra Android devices with Nougat and newer when used for COSU deployments

Graphical Timeline of Privilege Escalation Methods on Zebra Android Devices



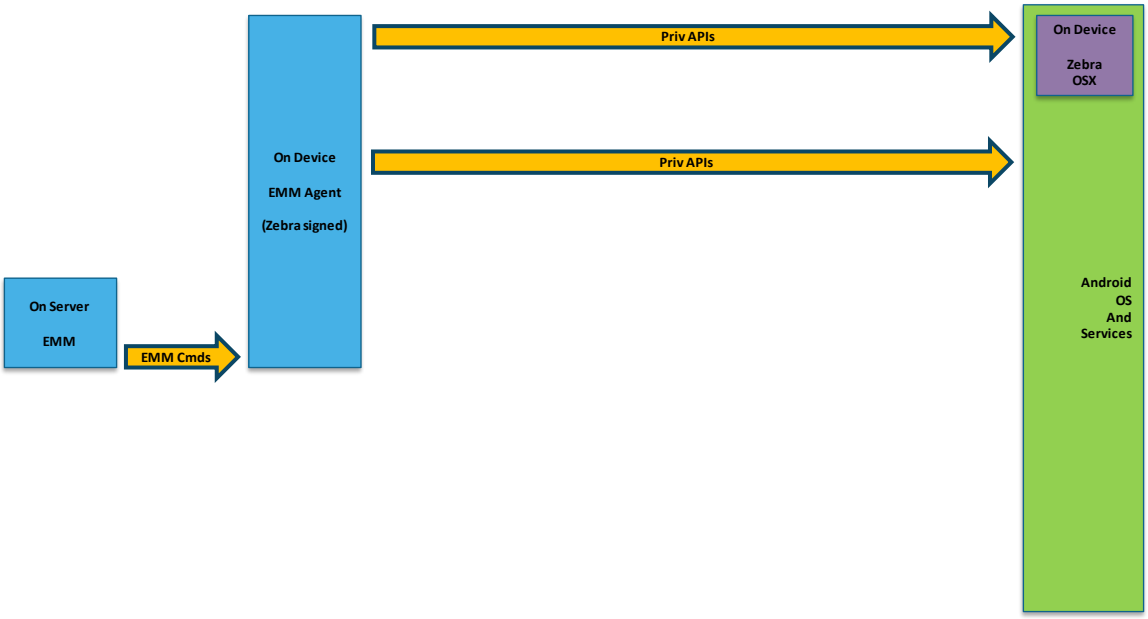
Legend:

Unsupported on Zebra Devices
Partially Supported or Supported with Limited Functionality on Zebra Android Devices
Fully Supported with Full Functionality on Zebra Android Devices
Deprecated (Supported but Not Recommended) on Zebra Android Devices
Discontinued (No Longer Supported) on Zebra Android Devices

EMM Solutions for Managing Legacy Zebra Android Devices

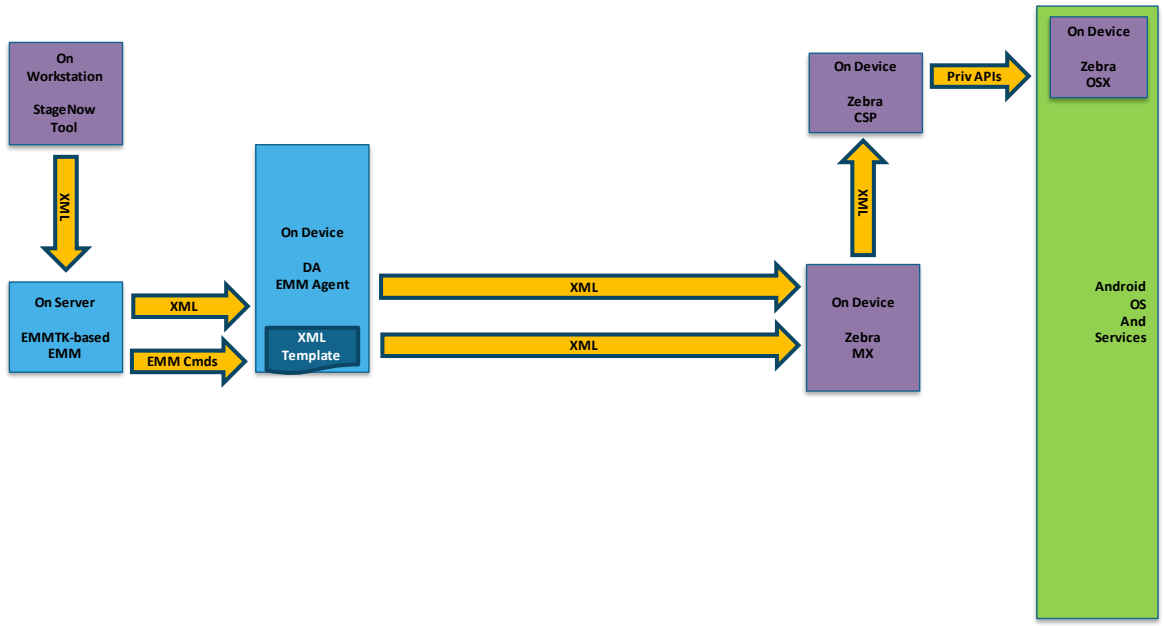
Successful existing EMM Solutions for managing Legacy Zebra Android devices running Android Marshmallow (6.x) and older generally followed one of the following approaches:

❖ Signed EMM Agent



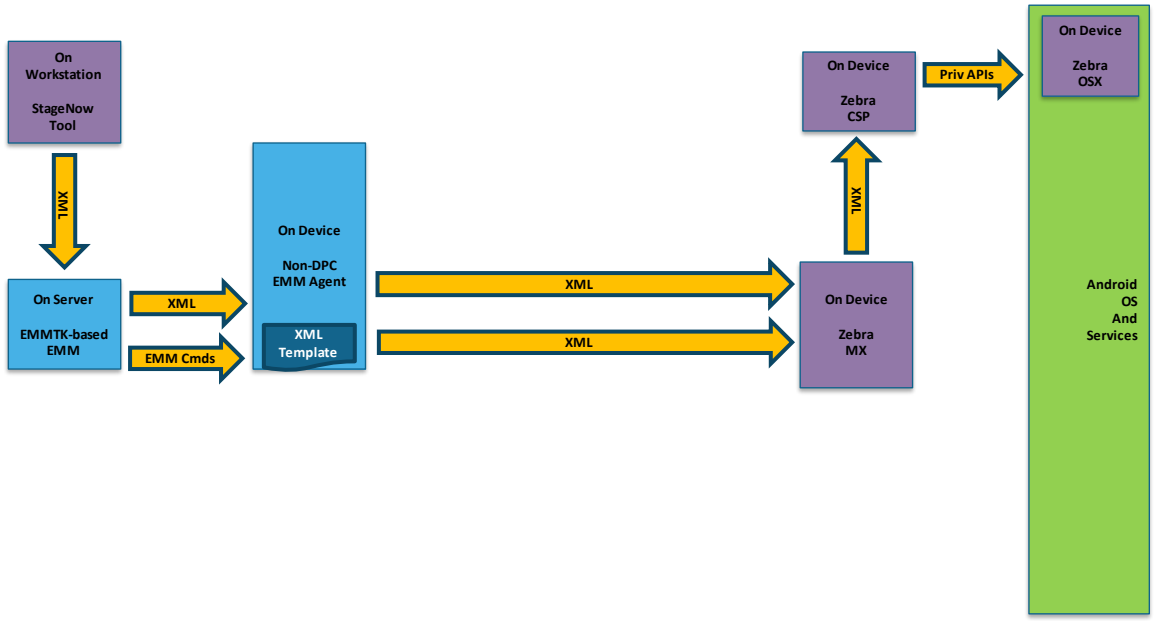
- As seen in the above diagram, and as summarized in the preceding tables, Tier 1 EMMs originally relied on a Zebra Signed EMM Agent.
  - All Tier 1 EMMs must begin porting to a more modern approach since Zebra will began phasing out support for Signed Agents in Nougat and will have discontinued all such support as of Oreo.
  - EMM Agents already signed can continue to be used to manage Zebra devices running Android versions that support Signed Agents.
  - EMM Agents already signed cannot effectively be used to manage Zebra devices running Android versions that do not support Signed Agents since they will be unable to gain access to the privileged functionality they require.
  - Zebra might at some point decline to sign new versions of EMM Agents that were previously signed for use on older devices.
- Tier 1 EMMs should switch from a Signed Agent to an alternate Privilege Escalation Method based on the versions of Android they wish to support.
  - To port a Signed EMM Agent to a single EMM Agent that could be used to manage all Zebra Android devices running Kit Kat through Oreo, the method Unsigned DA EMM Agent Augmented by MX, described below, would likely be preferable in most cases.
  - Once such an Agent was available, the EMM Vendor would be well positioned to support current and past Zebra Android devices and to begin porting to the new standards-based approach that will be required to manage future Zebra Android devices.

❖ Unsigned DA EMM Agent Augmented by MX



- As can be seen in the above diagram, and as summarized in the preceding tables, most Tier 2 EMM Vendors that already support complete solutions for managing Zebra Android devices have already developed a DA EMM Agent and augmented it using MX to provide the additional privileged functionality. Such EMM Vendors will be well positioned to support current and past Zebra Android devices and to begin switching to the new standards-based approach that will be required to manage future Zebra Android devices.

❖ Unsigned Non-DPC EMM Agent Augmented by MX

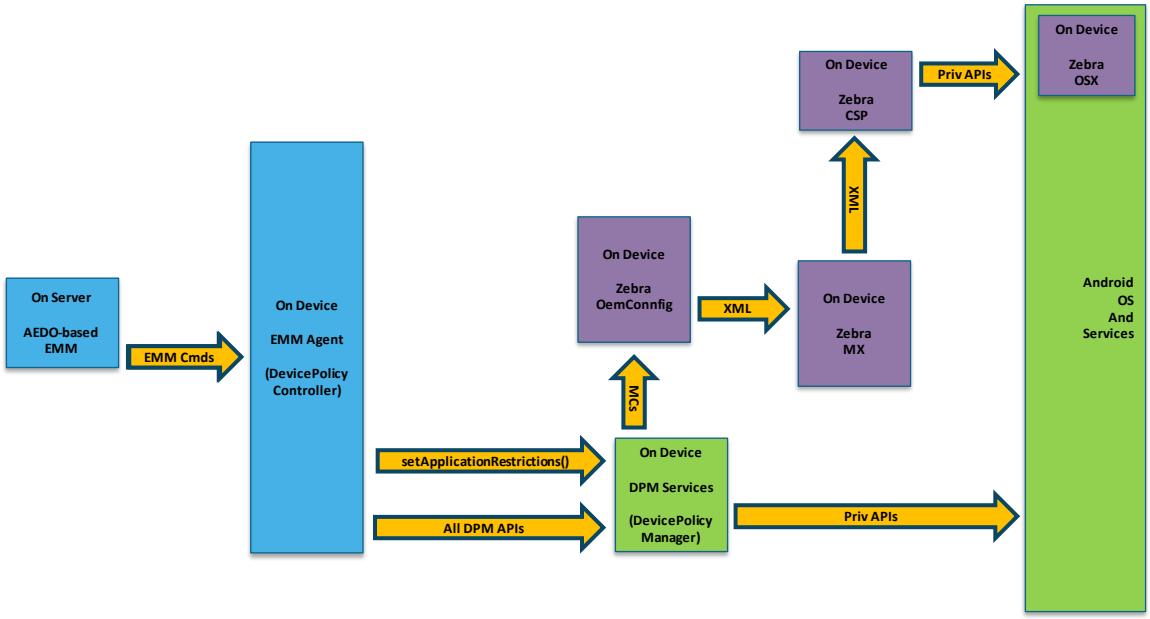


- As can be seen in the above diagram, and as summarized in the preceding tables, if a Tier 2 EMM Vendor already supports a complete solution for managing Zebra Android devices based on a non-DPC EMM Agent that is augmented using MX to provide the additional privileged functionality, they can continue to use that approach to support management of all Zebra Android devices through Oreo.
- Since such an approach will no longer work from Android P onward, switching to the new standards-based approach will be required to continue to manage future Zebra Android devices.
- Starting from a non-DPC EMM Agent will likely complicate the porting process to a DO EMM Agent and hence should not be left to the last minute. Once available, the new DO EMM Agent could be used to manage Zebra Android devices as far back as Android Nougat, if appropriate.

EMM Solutions for Managing Current and Future Zebra Android Devices

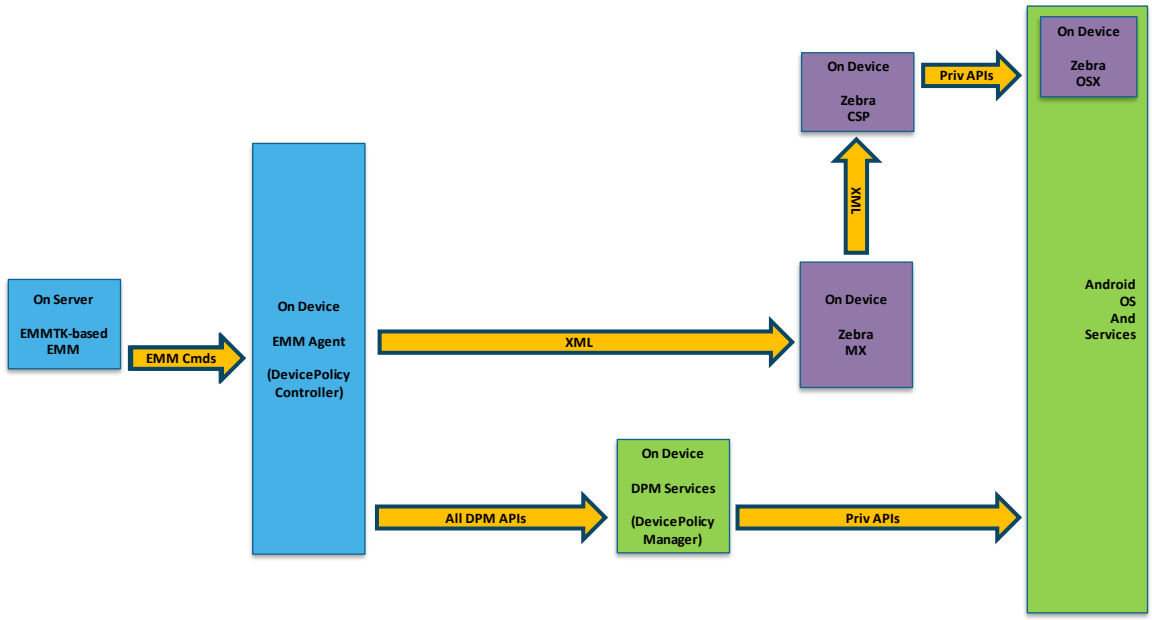
Successful EMM Solutions for managing Current and Future Zebra Android devices running Android Nougat (7.x) and later are recommended to follow one of the following approaches:

❖ Unsigned DO EMM Agent Augmented by ZMC



- This is the most preferred approach for the following reasons:
  - It is based on standard interfaces rather than Zebra-proprietary interfaces (e.g. MX).
  - It allows an EMM Agent to be completely generic and standard, with ZERO Zebra-specific code.
  - It allows the same EMM Agent to be used to manage devices from all OEMs.
  - It will support all Zebra Android devices running Nougat or later, albeit with more functionality in later versions.
  - It provides a means for an EMM to provide immediate support to customers for new ZMC features as they are added to devices by Zebra, without the need for any changes to the EMM Agent or EMM Server.
  - Once an EMM has successfully invested the one-time cost to port their existing solution to this new approach, they should be immune to the need for significant additional changes moving forward.
- This approach does have a few downsides:
  - It cannot be effectively used to support Zebra Android devices running Marshmallow and older.
  - It requires the most changes to port to this approach from any Agent using another approach.

❖ Unsigned DO or DA EMM Agent Augmented by MX



❖

- This approach has a few advantages that might make it attractive as an interim solution:
  - It allows the same Agent to be used to manage Zebra Android devices running Oreo and older, although use on Oreo is discouraged.
  - It might require fewer changes to port to this approach from an Agent using another approach.
- This approach has the following significant downside:
  - It will not be usable to manage Zebra Android devices running Android P and newer, because direct use of MX is discontinued beginning in Android P, and therefore should only be considered as a temporary stopgap measure.

High Level Porting Process

Everything prior to this point has dealt with establishing a baseline for porting by defining terms and explaining the possible starting state(s), available options, and roadmap. From this point forward, the focus is on the process used to accomplish porting from an existing EMM Agent to a new DO-based EMM Agent.

Depending on the actual starting state, the porting process can be simple or complicated. Before getting into the detail of the various steps that might be required, the following high-level list includes the steps that might or might not be required depending on the starting conditions.

1. Make the EMM Agent a DPC
   - If the EMM Agent is not already a DPC, then it must be made a DPC by following available Android documentation.
   - This document will not repeat this entire process, which can be found in the Android documentation, including at the following links:
     - https://developer.android.com/guide/topics/admin/device-admin
     - https://developer.android.com/work/dpc/build-dpc.
     - https://developer.android.com/reference/android/app/admin/DeviceAdminReceiver
2. Make the EMM Agent a DO
   - The EMM Agent must be coded to be a DO, and as such must support a suitable DO enrollment process (as described in a later step), detect when it has successfully been granted DO status, and utilize appropriate Android Device Management APIs once access to them has been granted.
   - This document will not repeat this entire process, which can be found in the Android documentation, including at the following links:
     - https://developer.android.com/reference/android/app/admin/DevicePolicyManager.html#ACTION_PROVISIONING_SUCCESSFUL
     - https://developer.android.com/reference/android/app/admin/DevicePolicyManager.html#ACTION_DEVICE_OWNER_CHANGED
       https://developer.android.com/reference/android/app/admin/DevicePolicyManager.html#ACTION_MANAGED_PROFILE_PROVISIONED
     - https://github.com/googlesamples/android-testdpc/blob/master/app/src/main/java/com/afwsamples/testdpc/AddAccountActivity.java#L75
3. Replace Legacy Methods
   - The EMM Agent might need to be changed to replace legacy Privilege Escalation Methods with more appropriate methods. This can include calling Android Device Administration API to which it gains access when it is granted DO status, or by using another suitable method (e.g. MX or ZMC) to perform functions not available via the Android Device Administration API.
   - This document will not repeat this entire process, which can be found in the Android documentation at the following link:
     - https://developer.android.com/reference/android/app/admin/DevicePolicyManager
   - This document will not explain the capabilities of the MX, information about which can be found at the following link:
     - http://techdocs.zebra.com/mx
   - ZMC is be covered in more detail elsewhere in this document.
4. Accommodate DO Enrollment

- The EMM Agent must provide a suitable method to support its enrollment as a DO. This might be done using DO enrollment modalities provided by Google and/or by using the Zebra StageNow Tool. No matter how it is accomplished, DO enrollment would generally need to accomplish all the following:
  - Optionally establish suitable network connectivity
  - Download the EMM Agent to the device from a suitable location on the network
  - Install the downloaded EMM Agent on the device
  - Grant the installed EMM Agent DO status
  - Supply the EMM Agent with configuration information required to contact and authenticate with a suitable instance of the EMM Server
- This document does not explain the entire process of implementing Google DO enrollment modalities, but it is appropriate to provide a high-level overview of Google DO enrollment modalities and a summary of available modalities.
  - Google DO enrollment modalities can be used to perform enrollment of an EMM Agent as a DO on GMS-enabled devices that can reach Google servers on the internet. Such methods will not work on Zebra devices running AOSP, nor will they work on Zebra devices that do not have a connection to the internet. The following high-level methods might be available, depending on the specifics of the Zebra Android device:
    - NFC Bump
      In this modality, the Administrator, using information provided by the EMM and tools provided by Google and/or the EMM, creates an NFC tag payload that contains all the data required to perform DO enrollment. This information can be burned into a suitable NFC tag, or another device can be equipped to host this payload an emulate an NFC tag.

      The NFC tag payload can include Wi-Fi settings information to enable configuration of a suitable network, can specify the location on the internet where the EMM Agent can be downloaded, and can specify the configuration information (in JSON format) required to enable the EMM Agent to connect to and authenticate with the proper EMM Server instance.

      Zebra GMS-enabled Android devices that are equipped with NFC reading capabilities have a provisioning service that runs in parallel with the Google Setup Wizard (SUW). In normal usage, while on an early screen of the SUW, the NFC tag (or tag emulated by another device) is "bumped" to perform DO enrollment.

    - QR Code
      In this modality, the Administrator, using information provided by the EMM and tools provided by Google and/or the EMM, creates a QR code barcode that contains all the data required to perform DO enrollment. This information can be printed, displayed on a computer screen, etc. as needed to enable it to be scanned by the device on which DO enrollment is to be performed.

      The QR code barcode can include Wi-Fi settings information to enable configuration of a suitable network, can specify the location on the internet where the EMM Agent can be downloaded, and can specify the configuration information (in JSON format) required to enable the EMM Agent to connect to and authenticate with the proper EMM Server instance.

      Zebra GMS-enabled Android devices that are equipped with QR code barcode reading capabilities have a provisioning service that runs in parallel with the Google Setup Wizard (SUW). In normal usage, while on an early screen of the SUW, the QR Code barcode is scanned using the device camera to perform DO enrollment.

- Hash Tag
  In this modality, the Administrator enters a “hashtag” value into a prompt in the SUW on a device that identifies the EMM to be enrolled as a DO. For this to work, the EMM Vendor must be registered with Google and receive a unique token. For example, an EMM might be issued the token “myemm,” in which case the Administrator would enter the value “afw#myemm” to indicate that the device should be enrolled as a DO for management by an instance of that EMM.

  Since this “hashtag” value must be hand-entered on a device, it should be kept short to simplify the administrator experience and reduce entry errors, hence it does not contain any network configuration or EMM Agent configuration information. If the device already has a suitable network connection to the internet (e.g. a Cellular data connection), then enrollment can proceed right away. If not, then the administrator will be required to hand-enter Wi-Fi configuration information to establish suitable network connectivity to the internet before enrollment can proceed.

  A provisioning service will be initiated by SUW on the device, will contact the Google provisioning server on the internet and check the “hashtag” value to determine if it identifies a valid registered EMM. If it does, then the provisioning service is provided with the information about the EMM Agent that should be downloaded, installed, and granted DO status on the device. Since no configuration information is available to configure that EMM Agent, the information required to connect to and authenticate with a suitable EMM Server instance must be entered by the administrator. This might take the form of a server address, email address, credentials, etc., depending on the specific EMM implementation.

- Zero Touch
  In this modality, an OEM or reseller that distributes the device to a customer is responsible for entering information (through a Google Console interface on the internet) that associates a set of device identifiers (e.g. IMEI, serial number, etc.) to the customer identifier of a customer to whom those devices have been transferred. The customer EMM Administrator would then log into the Google Console interface using their customer identifier and credentials, claim those devices and associate them to a specific EMM Server instance.

  The information entered by the customer via the Google Console interface can provide the same information that an NFC tag or QR Code barcode could contain EXCEPT network configuration. If the device already has a suitable network connection to the internet (e.g. a Cellular data connection), then enrollment can proceed right away. If not, the administrator must hand-enter Wi-Fi configuration information to establish suitable network connectivity to the internet before enrollment can proceed.

  A provisioning service will be initiated automatically on the device when it boots up that will contact the Google provisioning server on the internet, and verify that the identity of that device is associated with a valid registered EMM. If so, then the provisioning service is provided with the information about the EMM Agent that should be downloaded, installed, and granted DO status on the device. The provisioning service will also be provided with the configuration information that the EMM Agent will require to connect to and authenticate with a suitable EMM Server.

- Additional information on Google DO enrollment modalities can be found in the Android documentation, at the following links:
  - https://source.android.com/devices/tech/admin/provision

- https://developers.google.com/android/work/prov-devices
- https://developers.google.com/android/work/prov-devices#device_owner_provisioning_methods

- It is appropriate for this document to explain how to use the Zebra StageNow Tool to facilitate enrollment of a DPC-enabled EMM Agent as a DO.

  The Zebra StageNow Tool is designed to meet the needs of Enterprise customers that need to rapidly and efficiently configure large numbers of Zebra Android devices to prepare them for production use. When such preparation requires DO enrollment, StageNow provides a fast and reliable means to perform the steps previously identified for DO enrollment. The following are some of the key factors that can cause customers to choose StageNow over Google modalities when performing DO enrollment:

  - StageNow supports:
    - DO enrollment of Zebra devices running AOSP
    - DO enrollment of Zebra devices with no connectivity to the internet
    - DO enrollment of Zebra devices after SUW is complete, including when SUW is bypassed
    - Configuration of complex Wi-Fi networks, including those using Zebra-proprietary extensions, which might be required to establish network connectivity
    - Configuration of custom Cellular APNs, which might be required to establish network connectivity
    - Configuration of ethernet, which might be required to establish network connectivity

5. Enable Support for ZMC
   - The DO EMM Agent might require changes to enable and/or maximize its ability to leverage ZMC. While the need to make these changes can be driven by the need to support ZMC (in that they might not be needed if ZMC were not used), all of them CAN be accomplished without including Zebra-proprietary code into the EMM Agent. This will be covered in more detail later in this document.

Porting Process Details

The following additional details should help with the planning and execution of the porting process by identifying Zebra proprietary aspects and/or special considerations to ensure optimal support of Zebra Android devices.

1. Replace Legacy Methods

   As noted earlier, an EMM Agent that is being ported will likely need to be changed to replace legacy Privilege Escalation Methods with newer, more standard methods. Assuming that porting begins with an EMM Agent that is using MX as a Privilege Escalation Method, the most important first step would be to identify all the MX functions that are used by the EMM Agent and to understand how and why they are being used by the EMM Agent to support the existing functionality offered by the EMM Solution.

   For each MX function identified as being used by the EMM Agent, and identified as being needed to provide functionality for the EMM Solution, a porting plan must be created. Since EMM Agents can continue to use MX functions via the EMMTK through Android Oreo, it might not be critical to port all such functions immediately.

   An interim hybrid EMM Agent could be developed that uses a combination of MX and ZMC to augment the methods of the Android Device Administration API that the EMM Agent can access as a DO. But it must be understood that such an EMM Agent is NOT a generic EMM Agent, so long as it includes Zebra-proprietary code to invoke MX functions. Hence, such an EMM Agent might not be suitable for use on non-Zebra devices, or might require conditional coding to suppress use of MX functions on non-Zebra devices. And since the EMM

Agent will no longer be able to use MX functions from Android P onwards, the EMM Vendor should still have a plan for how to replace all such functions and reaching a truly generic EMM Agent that contains zero Zebra-proprietary code.

Zebra strongly recommends that EMM Vendors immediately replace all MX functions that can be replaced using methods of the Android Device Administration API that can be accessed as a DO. This would enable that functionality to be performed on any standard Android device from any OEM and would be an important first step on the road to a generic EMM Agent. To assist in this process, Zebra provides MX-to-AEDO and MC function maps, which match functions pof the proprietary Zebra CSPs to methods of the Android Device Administration API and Managed Configurations of OemConfig available to a DO and that provide equivalent or comparable functionality. These tables can be used to jump-start this part of the porting process.

While many of the MX functions identified as being used by the EMM Agent can likely be replaced by methods of the Android Device Administration API, some MX functions will have no equivalent in the Android Device Administration API. Such MX functions will need to be accomplished by using ZMC. Since ZMC functions are exposed via the standard Managed Configuration features of the Android Device Administration API, they will become available for use by the EMM Agent as soon as it becomes a DO.

In many cases, an EMM Vendor might have embedded use of MX functions directly into their legacy EMM Agent, since this practice was recommended via the Zebra EMMTK. While an EMM Vendor COULD embed code to use ZMC right into their DO EMM Agent, it would require the EMM Vendor to know the details of ZMC and embed that knowledge directly into the EMM Agent. This is generally discouraged for the following reasons:

- A key reason for switching from OEM-proprietary interfaces, such as MX functions, to more standard interfaces, such as Managed Configurations, is the desire to have a single EMM Agent that has no OEM-specific code embedded within it. Such a Generic EMM Agent could run on all OEM-devices, which is a major improvement from the Legacy model where EMM Vendors had to maintain at least one (and sometimes MORE than one) unique EMM Agent for each OEM. Embedding ZMC into an EMM Agent is contrary to that goal and would make the EMM Agent Zebra-specific again.
- ZMC is subject to change over time, including the addition of new functions and (less frequently) deprecating and eventually discontinuing obsolete functions. By embedding specific knowledge about ZMC into the EMM Agent, and EMM Vendor creates a potential future situation where the EMM Agent code could need to change and/or might a need to support multiple EMM Agent variants to handle a variety of Zebra Android devices.

To keep their EMM Agent generic, an EMM would need to avoid building any code that is unique and specific to any OEM, including Zebra, into their EMM Agent. This could problematic, in some cases, since an existing end-to-end EMM feature might rely on the EMM Agent to have implicit knowledge of how to implement a given feature on a given device. But that is no longer truly practical when the EMM Agent is generic, since it should not have ANY embedded knowledge of how to perform functions that cannot be implemented using standard interfaces.

For example, if an EMM Solution offers an end-to-end “OS Update” feature, the Console UI might provide a specialized UI to invoke that feature. When that feature is invoked by a customer EMM Administrator, the EMM Server might send an EMM-specific “OS Update” command to the EMM Agent, with parameters, with the expectation that the EMM Agent will know how to perform that command on the device on which it was running. But a generic EMM Agent, that could run on any Android device from any OEM could not have any such embedded knowledge about how to perform that command.

To achieve a truly generic OEM-agnostic EMM Agent, all specialized knowledge about how to perform OEM-specific operations would need to be removed from the EMM Agent. Instead, the EMM Agent must rely solely on generic, non-OEM-specific standard interfaces to perform all functionality requested by the EMM Server. When a standard-based interface can be used to perform an OEM-specific function, bypassing OEM-specific data, the EMM Server (not the EMM Agent) must be the source of that data.

This can mean that the EMM Server gets more complicated because it must be the repository for specialized knowledge about OEM-specific functions that can be accomplished on specific OEM devices via the standard functions exposed by the generic EMM Agent. This is the tradeoff required to achieve a truly generic EMM Agent. This will likely be a beneficial tradeoff in the long run since it eliminate the even greater complexity of developing, maintaining, and managing large numbers of OEM-specific EMM Agents. Further, it is generally simpler and faster to add new OEM-specific logic to one common point, the EMM Server, than trying to add it to many different EMM Agents residing on many devices in many different locations.

2. Support Managed Configurations

To use ZMC via a generic EMM Agent to replace MX functions that cannot be accomplished using methods of the Android Device Administration API, the EMM must be able to create and submit Managed Configurations to Zebra OemConfig via the EMM Agent. This creates the following high-level requirements:

- There must be a way to generate sets of Managed Configurations at the EMM Server.

  The most common, and simplest, approach is for the EMM Server to acquire the Schema for the Zebra OemConfig application and generate a data-driven UI as part of the EMM Console UI that allows the customer EMM Administrator using the EMM Console UI to create sets of Managed Configurations based on that Schema.

  This approach has the following benefits:
  - It avoids the need to embed knowledge of ZMC into the EMM Server.
  - It allows the EMM Solutions to adapt to changes in the Zebra OemConfig Schema (such as the addition of new features) without making changes to the EMM Server code.
  - It allows new features of Zebra OemConfig to be immediately supported by the EMM Solution as soon as the new Zebra OemConfig Schema is published.

  This approach has the following drawbacks:
  - It requires the customer EMM Administrator using the EMM Console UI to be knowledgeable about Zebra OemConfig and the functions it can be used to perform. This can result in a less integrated, and perhaps less friendly, user experience for the customer EMM Administrator.
  - Since the customer EMM Administrator, not the EMM Server, creates Managed Configuration sets, the EMM Server cannot know what any given Managed Configuration is intended to accomplish. As a result, the EMM Server cannot easily provide feedback, status, or tracking about its execution.
  - If the EMM Server is managing new and legacy devices, it might be difficult or impractical to provide the same user experience across all devices, especially if the user experience provided for legacy devices relied upon direct use of MX functions by the EMM Agents now being replaced by ZMC functions.

  An alternate approach is for the EMM Server to embed knowledge of selected ZMC functions that can be performed and to automatically generate Managed Configuration sets to perform these functions.

This approach has the following benefits:

- It allows the EMM Server to present a friendlier user experience for the customer EMM Administrator by tightly integrating functions into the EMM Console UI workflow. This could be especially important if legacy devices already have a tightly integrated experience and switching to a data-driven UI results in a significantly inferior user experience for new devices as compared to legacy devices.
- Since the EMM Server directly creates Managed Configuration sets, the EMM Server knows what they do and hence can more easily provide feedback, status, and tracking about their execution.

This approach has the following drawbacks:

- It requires ZMC-specific knowledge to be embedded into the EMM Server code. This could significantly increase the complexity of the EMM Server implementation.
- ZMC is subject to change over time, including adding new functions and, less frequently, deprecating and eventually discontinuing obsolete functions. By embedding specific knowledge about ZMC into the EMM Server, an EMM Vendor creates a potential future issue under which the EMM Server code could need to change and/or might need to support multiple variations.

A given EMM Solution might choose a hybrid of these two approaches, where existing Console UI features that most benefit from maintaining the existing user experience use latter approach, while simpler and less integrated features and those with less demanding user experiences use the former approach.

- There must be a way to deliver generated sets of Managed Configurations to a Zebra device.

  The assumption behind this document is that the EMM Vendor has its own EMM Agent that runs on the device, is in communication with the EMM Server and that is responsible for performing actions on the device at the request of the EMM Server. This generally translates into commands sent by the EMM Server to the EMM Agent to request that specific operations be performed, and responses sent back to the EMM Server by the EMM Agent to indicate the results of those operations.

  Each EMM Vendor is expected to have its own exchange framework, consisting of protocols and formats for commands and responses between their EMM Server and EMM Agent. The format in which sets of Managed Configurations are stored on the EMM Server and the format in which they are transported to the EMM Agent can be anything the EMM Vendor finds suitable for that purpose. In most cases, an EMM Vendor will simply add support for transporting sets of Managed Configurations within their existing Server/Agent exchange framework. Whatever format the EMM Vendor chooses to encode Managed Configurations must provide a reliable transportation for delivering them to the EMM Agent without loss of fidelity.

- There must be a way to submit sets of Managed Configurations to Zebra OemConfig once they are delivered to a Zebra device.

  The EMM Agent is generally responsible for interpreting commands received from the EMM Server into appropriate actions invoked using device-side interfaces. A generic EMM Agent must do this using only standard Android interfaces. In the case of Managed Configurations received from the EMM Server, the EMM Agent would be responsible for translating sets of Managed Configurations from whatever format was used by the EMM Server to transport them to the EMM Agent into the Bundle format required by the setApplicationRestrictions() method of the Android Device Administration API. Successful execution of the Managed Configurations would depend on a faithful translation.

- There must be a way to determine when processing for a submitted set of Managed Configurations to a Zebra device has finished, thus indicating it is safe to submit another set of Managed Configurations.

  The standard Android Managed Configurations model does not currently provide any mechanism for a DO to know when a set of Managed Configurations that it has submitted for processing by a specific application has completed. This is a result of the original design that was intended to configure applications. Since ZMC supports configuring the device through the Zebra OemConfig Application, the impact of a set of Managed Configurations and the need to know when it has completed are more critical. In addition, the order of operation prior to, during, and after execution of a set of Managed Configurations could also be quite important.

  Zebra OemConfig supports a mechanism whereby an EMM can request to be notified asynchronously (via Android Intent) of the completion of the processing of a set of Managed Configurations. There are three key aspects to this mechanism:

  - The set of Managed Configurations published by Zebra OemConfig is organized to form a Transaction, consisting of a Transaction ID and an ordered series of Transaction steps. This organization is encoded in the Zebra OemConfig Application Schema published by Zebra.
  - Each separate submission of Managed Configurations to Zebra OemConfig is required to assign a new unique value to the Transaction ID Managed Configuration defined within the Schema. When executing sets of Managed Configurations, Zebra OemConfig will not begin execution for a submitted set of Managed Configurations until the Transaction ID changes, at which time the complete set of Managed Configurations will be executed.
  - As part of the definition of a Transaction, a set of Managed Configurations defined within the Schema can optionally be used to define the characteristics of an Android Intent that should be sent by Zebra OemConfig when the processing of that Transaction is completed.
- There should be a way to determine the completion status and information about any failures that occurred during the processing of a set of Managed Configurations once processing of the Transaction has finished on a Zebra device.

  When Zebra OemConfig is directed to send an Android Intent when the processing of that Transaction is completed, Zebra OemConfig will automatically attach to the intent an indication of the Transaction Status (success/failure) and, if failure, an Error List containing descriptions of all errors that occurred during the processing of that Transaction. To take best advantage of this information, the EMM Agent should support the Generic Intent Listening Mechanism described below.

  An ideal approach would be for the EMM Server to define an Intent to be used for OemConfig. The EMM Server would then use the Generic Intent Listening mechanism to configure the Generic EMM Agent to listen for that Intent and send the information attached to it, including the Transaction ID , the Completion Status, and Error List. Each time the EMM Server sends a Transaction to the EMM Agent to be submitted to Zebra OemConfig, it would describe that same Intent, thus directing OemConfig to send that Intent when the Transaction is complete. This would create a generic feedback loop that would allow the EMM Server to track the full lifecycle of each Transaction without requiring the EMM Agent to contain any Zebra-specific code.

3. Support for Acquisition of Managed Configuration Schemas

   A key benefit of the Android Managed Configuration Model is the ability of an EMM Server to acquire the Schema for an application of which it has no prior knowledge and immediately support configuring that

application without the need to make any changes to the EMM Server or EMM Agent. EMM Vendors whose solutions already support configuring applications using the Android Managed Configuration Model likely already provide a mechanism to acquire Managed Configuration Schemas. Once the Managed Configuration Schema for a given application is acquired, the EMM Server can process the Schema and present a data-driven UI for creating Managed Configuration sets for delivering to that application via their Generic EMM Agent.

There are a variety of ways that an EMM Server could acquire Managed Configuration Schemas, including:

- Request the Schema for an application published in the Google Play Store

  This is the method that is most commonly supported by existing EMMs since most applications that support configuration using the Android Managed Configuration Model are published to the Google Play Store.

  An EMM that has registered with Google can add logic to its EMM Server to acquire Managed Configuration Schemas for applications published in the Google Play Store by using the Google Play Services EMM APIs and supplying the unique package name of the application for which the Schema is desired. Information about these APIs can be found at the following link:

  - https://developers.google.com/android/work/play/emm-api/managed-configurations

  One downside of this approach is that there can be only one version of any application published to Google Play Store. This means that if the Managed Configuration Schema evolves over time, the Google Play Store will likely only have the latest version, even if older versions with subsets of the support Managed Configurations might be present on devices being managed. This makes knowledge of what worked and what didn't (as supported by Zebra OemConfig) even more important.

- Extract the Schema for an application from a APK file uploaded to the EMM Server

  In some cases, a customer might develop their own application or acquire one from a third-party and might not want (or be able) to publish that application to the Google Play Store. If an Administrator elects to deploy that application to their devices, it must be uploaded to the EMM Server. At that point, the EMM Server could provide a means to extract the Managed Configuration Schema from that APK file and allow the Administrator to use it to create configurations to be sent to devices to which that application is deployed.

  This method is quite flexible, but is limited to cases under which the Administrator has possession of the APK file and is willing to upload it to the EMM Server. In cases where the EMM Server will be used to deploy the application, this is quite likely. If the EMM supports multiple versions of APK files for the same package name, this could enable the EMM to support multiple versions of the Managed Configuration Schemas published by those multiple APK versions.

- Import the Schema for an application from a Schema file uploaded to the EMM Server

  In some cases, a customer wishes to configure an application that is already present on their devices and hence might not want to upload that APK to the EMM Server. If the Administrator can acquire the Managed Configuration Schema somehow (such as from the application developer), and upload it to the EMM Server, then the EMM Server could allow the Administrator to use it to create configurations to be sent to devices on which that application is installed.

This method is sort of a catch-all since it allows an Administrator to utilize any Managed Configuration Schema for any application they might be using. The EMM could also choose to enable multiple Schema versions for the same package name to be uploaded.

- Extract the Schema for an application in device using the EMM Agent, which then sends it to the EMM Server

  Every application that publishes Managed Configurations and is installed on a device can have its Managed Configuration Schema extracted by an EMM Agent that enrolled on that device as a DO. The EMM Agent could then send that Schema to the EMM Server which could allow an Administrator to use it to create configurations to be sent to that device.

  This method can provide maximum flexibility, but can be quite complex to implement. In the plus side, having EMM Agent in the device acquire the Managed Configuration Schema from the version of an application that is on a device and send it to the EMM Server ensures that the capabilities defined in the Schema exactly match the capabilities of the application. The complexities mostly occur at the EMM Server, where the versions of Schema extracted from a multitude of devices would need to be correlated and managed. But if done properly, this could enable the EMM Server to offer groupings of devices by common Managed Configuration Schema versions, and even perform detection of "common denominator" subsets that could be used to configure larger populations of devices.

- Request the Schema for an OEM-specific application through some OEM-specific Server API

  If an OEM such as Zebra builds an application into its devices that publishes Managed Configurations, it also might choose to provide its own APIs that can be used by an EMM Server to request the Managed Configuration Schema for that application.

4. Support for Complex Managed Configuration Schemas

The Android Managed Configuration Model allows the definition of Schemas that are quite flexible and complex. Because the original purpose of Managed Configurations (formerly known as Application Restrictions) was to configure simple settings for applications, Google elected to limit applications published to the Google Play Store to those that publish Managed Configuration Schemas that fall within a narrow subset of the capabilities allowed by the Android Managed Configuration Model.

This model allows Managed Configurations that define Scalar values (String, Boolean, Integer, etc.), Bundle values (collections of Scalar, Bundle, and Bundle Array values), and Bundle Array values (ordered lists of a single Bundle type). While an Android application can publish a Schema that uses any combination of the above, the Google Play Store will refuse to import applications that exceed the following subset:

- Scalar values at the top level or within any Bundle
- Bundle values at the top level or within a Bundle Array
- Bundle Array values at the top level or within a Bundle
- No more than 2 levels of nested Bundles
- No Bundles directly within another Bundle

Because Zebra OemConfig is designed to enable configuration of many different settings across many device subsystems, it would produce a poor user experience if the settings for each subsystem all had to be at the same level since an Administrator trying to configure the device would have scroll through hundreds of

settings to find the ones of interest. To improve the user experience, OemConfig uses Bundles within Bundles, (sometimes several levels deep) to hierarchically organize the large set of Managed Configurations into groupings related to each other. This allows an Administrator to better focus on a group of Managed Configurations that relate to a specific subsystem or use case.

Because Zebra OemConfig exceeds the limitations imposed by Google for applications published to the Google Play Store, it would not normally be possible for Zebra to publish Zebra OemConfig to the Google Play Store. Zebra OemConfig is pre-installed on all Zebra devices on which it is supported, and would therefore not need to be in the Google Play Store to enable EMMs to deploy it to devices. But it WOULD need to be in the Google Play Store if EMMs needed or wanted to acquire the Managed Configuration Schema for Zebra OemConfig from the Google Play Store.

As it happens, acquiring Managed Configuration Schemas from the Google Play Store is currently the most common method supported by EMMs. Google has agreed that Zebra OemConfig needs a complex Schema, and has consented to waive the limitation and allow Zebra OemConfig to be published to the Google Play Store despite its unusually rich Schema. Google expects to waive the limitations for other OEMs that elect to produce their own OemConfig applications. Google has approved this approach and recommends that other OEMs follow the OemConfig approach pioneered by Zebra when they need to expose OEM-specific extensions to EMMs using DO Agents.

The upshot of this is that an EMM Vendor whose support for Managed Configuration Schemas only supports the more limited Managed Configuration Schemas supported by most applications in the Google Play Store will likely be unable to support the use of Zebra OemConfig until their Managed Configuration Schema support is extended to support the richer Schema subset used by Zebra OemConfig. Such extensions will not interfere with backward compatibility to applications using more limited Managed Configuration Schemas and will enable EMMs to support other OEMs if/when they adopt a similar OemConfig approach and also need to use a similarly rich schema.

5. Support for Hidden Managed Configurations with Macros

   The nature of the Android Managed Configuration Model is such that an EMM Server can specify, in a single set of Managed Configurations to be sent to a given application, as few or as many Managed Configurations as is appropriate for the circumstances, chosen from among the Managed Configurations within the Managed Configuration Schema for that application. Further, for each Managed Configuration included in a set of Managed Configurations to be sent to a given application, the value provided for that Managed Configuration could be any value consistent with the definition of that Managed Configuration Schema within the Managed Configuration Schema for that application.

   The Android Managed Configuration Model allows the definition of Schemas that include Hidden Managed Configurations. A Hidden Managed Configuration is one defined within the Schema but that is not intended to be presented in the data-driven UI generated using that Schema. Instead, the intent of a Hidden Managed Configuration is to provide an indication to the EMM Server processing the Schema that the information should be consumed or produced by the EMM Server itself, without interaction with the Administrator.

   In the case of non-Hidden Managed Configurations, the choice of which Managed Configurations to be specified in a set of Managed Configurations that will be sent to a given application is left to the Administrator, via the data-driven UI generated based on the Configuration Schema for that application. Further, the choice

of which value to be included for a given Managed Configuration is left to the Administrator using the data-driven UI generated based on the Configuration Schema for that application.

But for Hidden Managed Configurations, since they will not appear in the data-driven UI, the choice of whether to include them in a set of Managed Configurations must be made by the EMM Server. Further, for each Managed Configuration included in a set of Managed Configurations, the choice of which value to be specified for that Managed Configuration must be made by the EMM Server. And both decisions must generally be made without any feedback from the Administrator since Hidden Managed Configurations do not appear in the data-driven UI.

The Android Managed Configuration Model is silent on how Hidden Managed Configurations should be used or implemented within an EMM Server (aside from specifying that they should be hidden from the Administrator). Since Hidden Managed Configurations are intended for use by the EMM Server directly instead of by the Administrator indirectly through the data-driven UI, it is unclear exactly how an EMM Server should decide what to do with a given Hidden Managed Configuration without some form of direction provided by the Schema. Nothing in the Android Managed Configuration Model Schema defines such a mechanism.

Zebra OemConfig uses Hidden Managed Configurations for a variety of purposes, and needed a way to communicate (via the Schema) how the EMM Server should interpret and interact with them. Since the Android Managed Configuration Model defines no such mechanism, Zebra defined one. To avoid creating a deviation from the standard, Zebra fit this information into the standard Schema by leveraging the fact that certain attributes that can be specified in the Schema for a Managed Configuration have no relevance for a Hidden Managed Configuration since they are related to the data-driven UI, and no such UI is shown for a Hidden Managed Configuration.

Zebra has repurposed the following attributes for Hidden Managed Configurations (identified by setting the attribute android:restrictionType="hidden"):

- android:title

  Since a Hidden Managed Configuration will not be presented in the data-driven UI, the title attribute, normally used to specify the prompt, has no relevance. Zebra has thus repurposed this attribute to communicate whether the Hidden Managed Configuration should be consumed by the EMM Server ("read"), produced by the EMM Server ("write"), or both.

  At present, the only "read" Hidden Managed Configuration (with android:title="read") used by Zebra OemConfig is used to identify the version of the Schema. It is somewhat surprising that there is no currently defined way for an EMM Server to identify the version of a Managed Configuration. Perhaps this is because when obtaining a Managed Configuration Schema from an APK file, the version of the APK file could be taken as an implicit indication of the Managed Configuration Schema version. But in cases where the Managed Configuration Schema is acquired independently of an APK, a means to identity the Schema version from the Schema itself can be quite useful.

  Zebra OemConfig uses several "write" Hidden Managed Configurations (with android:title="write") to identify information that the EMM Server should supply, as opposed to getting it from the Administrator through the data-driven UI. More information on the handling of "write" Hidden Managed Configurations can be found below in the explanation of android:defaultValue.

- android:description

  Since a Hidden Managed Configuration will not be presented in the data-driven UI, the description attribute (usually used to specify help text) has no relevance. Zebra has thus repurposed this attribute to communicate explanatory information to an EMM Vendor about the purpose and/or usage of the Hidden Managed Configuration. Since this is a human-readable text string, it is not likely useful to be consumed by the EMM Server programmatically. But it might come in handy when troubleshooting, or in situations where an EMM Vendor elects to build logic into their EMM Server that "knows about" Zebra OemConfig.

- android:defaultValue

  Since a Hidden Managed Configuration will not be presented in the data-driven UI, the defaultValue attribute (usually used to specify the value used to initialize the Managed Configuration) has no relevance. Zebra has thus repurposed this attribute for "write" Hidden Managed Configurations to communicate information to the EMM Server about the sort of value it should produce and place into the Hidden Managed Configuration.

  The intention of a "write" Hidden Managed Configuration is for the EMM Server to produce information, as opposed to getting it from the Administrator through the data-driven UI. In order for an EMM Server to do this without having prior knowledge of the Schema, the android:defaultValue can contain macros that indicate specific types of information to be supplied. While Zebra has defined these macros to suit the needs of Zebra OemConfig, they are generic, and could be used by other OEMs that elect to implement their own OemConfig.

  Google has approved this approach and sanctioned the set of macros defined by Zebra for OemConfig. Over time, other macros might well be defined by Zebra, Google, or other OEMs. The point of using a macro in the android:defaultValue attribute to trigger value production instead of using the Managed Configuration android:key attribute is it allows more flexibility in Schema design and avoid the need for EMM Vendors to code different OEM-specific key values. An EMM Server must implement the full set of macros defined by Zebra OemConfig to leverage all the features of Zebra OemConfig. But having implemented those macros, they could be used by any other OEM that elected to provide comparable features.

  To include a macro in the value of the android:defaultValue attribute, simply choose a macro name and enclose it in curly braces (e.g. "{macro}"). In theory, the value of the android:defaultValue attribute could contain non-macro text and/or more than one macro. At present, Zebra OemConfig uses only a single macro in the value of the android:defaultValue attribute and no other non-macro text.

  The following macros are used by Zebra OemConfig:

  - {transactionId}
    The EMM Server should replace this macro with the value on an EMM Server-defined ID for the Transaction to be processed by Zebra OemConfig. Since Zebra OemConfig will not process a Managed Configuration set without a Transaction ID, this macro should be considered mandatory.

    The EMM Server is free to define Transaction IDs in any manner it chooses. The only requirement is that two Transactions submitted sequentially to the Zebra OemConfig on any given device MUST have different Transaction IDs.

    An EMM Server could theoretically reuse a pair of Transaction IDs, by alternating them. But most EMM Servers will likely use some form of algorithm to generate successively unique Transaction IDs,

such as by incrementing a counter, to allow Transactions to be differentiated from each other over time.

- {intentType}
  The EMM Server should replace this macro with the type of intent it wishes to have sent when the Transaction is complete. Since sending an intent when a Transaction is complete is optional, an EMM Server could elect not to include in a Managed Configuration set all "write" Hidden Managed Configurations that have "intent" macros in the values of their android:defaultValue attributes.

  If an EMM Server elects to request the sending of an intent when a Transaction is complete, this macro is required to specify the type of intent to be sent. The allowed values are:

  - startActivity
    This value is not commonly used since it would generally cause an Android Activity to pop up and interact with the Device User, which is inconsistent with normal expectations for an EMM Agent. In most cases, EMM actions are expected to occur silently in the background and not produce any visible results that could impact the workflow of the Device User.
  - startService

    This value is commonly used since it causes an Android Service to be invoked in the background, which is consistent with normal expectations of an EMM Agent. To be notified successfully, the EMM Agent must include an Android Service that could be initiated using this type of intent. See the explanation later regarding Add Generic Intent Listening Mechanism.
  - sendBroadcast

    This value is commonly used since it allows an Android Service to be notified, via a broadcast intent, in the background, which is consistent with normal expectations of an EMM Agent. To be notified successfully, the EMM Agent must include an Android Service that registers a listener for this type of intent. See the explanation later regarding Add Generic Intent Listening Mechanism.
- {intentComponent}
  The EMM Server should replace this macro with the Android Component Name of the Android Component to which the intent should be sent to signal Transaction completion. This macro is used by Zebra OemConfig only if the {intentType} macro is used to indicate that an intent should be sent using startActivity or startService. Because broadcast intents do not allow specification of a destination component, an EMM Server can elect not to include in a Managed Configuration any "write" Hidden Managed Configurations that have this macro when the {intentType} indicates that an intent should be sent using sendBroadcast.
- {intentAction}
  The EMM Server should replace this macro with the Action Value to be specified in the intent that should be sent to signal Transaction completion. It is up to a specific EMM whether this macro must be filled in, and depends on whether the EMM wishes to have an Action Value in the intent sent to indicate completion of a Transaction. In most cases, an EMM would elect to specify a Component Name, Action Value, or both. If an EMM decides that an Action Value is not required, the EMM can elect not to include in a Managed Configuration any "write" Hidden Managed Configurations that have this macro.

- {intentExtraName}
  The EMM Server should replace this macro with the Extra Name to be specified in the intent that should be sent to signal Transaction completion. It is up to a specific EMM whether this macro must be filled in, and depends on whether the EMM wishes to have an Extra Name/Value pair attached to the intent sent to indicate completion of a Transaction. If an EMM decides that an Extra Name/Value pair is not required, the EMM can elect not to include in a Managed Configuration any "write" Hidden Managed Configurations that have this macro.

  Zebra OemConfig allows only a single Extra Name/Value pair to be attached to the intent sent to indicate completion of a Transaction. The Name and Value can be chosen by the EMM to suit its own needs, and allows the EMM to attach a single EMM-specific piece of data to the Transaction that will be sent back to the EMM Attached to the Transaction-completion intent. It is always required to specify the {intentExtraName} and {intentExtraValue} macros as a pair. If an EMM elects to specify a Managed Configuration with a "write" Hidden Managed Configuration containing one of these two macros, it must specify both. If an EMM elects not to specify a Managed Configuration with a "write" Hidden Managed Configuration containing one of these two macros, it must not specify the other.
- {intentExtraValue}
  The EMM Server should replace this macro with the Extra Value to be specified in the intent sent to signal Transaction completion. It is up to a specific EMM whether this macro must be filled in, and depends on whether the EMM wishes to have an Extra Name/Value pair attached to the intent sent to indicate completion of a Transaction. If an EMM decides that an Extra Name/Value pair is not required, then the EMM can elect not to include in a Managed Configuration any "write" Hidden Managed Configurations that have this macro.

6. Add Generic Intent Listening Mechanism

   As noted above, Zebra OemConfig supports a set of Managed Configurations that can optionally be used to define an Android Intent to be sent when the processing of a Transaction is completed, and that indicates whether a completed Transaction was successful and if not, sends a list of error messages that detail all failures that occurred.

   A primary goal of porting from DA to DO is to produce an OEM-agnostic EMM Agent that can run on an Android device and that embeds no OEM-proprietary logic. It should be obvious that including OEM-proprietary code into an EMM Agent is a slippery slope. Such logic could seriously complicate the maintenance of an EMM Agent, especially if different incompatible OEM- proprietary features need to be added. In the extreme, that could reverse the course back to a need to maintain different EMM Agents for different OEMs, which would be highly undesirable.

   It might first seem contrary to the goal of having a generic EMM Agent to embed logic into the EMM Agent to listen for and forward to the EMM Server an intent sent by Zebra OemConfig to provide notification of Transaction completion. The key is to understand that Intents are a generic Android mechanism and not an OEM-specific mechanism, even if a SPECIFIC Intent is OEM-specific. Many standard Android Intents exist, and an EMM solution would benefit from a generic EMM Agent that could be configured to listen for and forward information about Intents to the EMM Server. Zebra OemConfig is designed to be configurable, allowing customization of the details of the intents it sends. This flexibility allows the EMM to configure Zebra OemConfig to send an intent that best fits its needs when using Zebra OemConfig vs. the capabilities of its generic EMM Agent.