



### **OemConfig Schema Data-Driven UI**

This document describes features permitted in Schemas defined by the Android Managed Configuration Model that OemConfig applications such as Zebra OemConfig are permitted to use, and that non-OemConfig applications are not. This is important because an EMM can be coded to handle only Managed Configuration Schemas that conform to the more restrictive subset of Schema capabilities imposed when uploading non-OemConfig applications to the Google Play Store. In such cases, an EMM is unable to properly handle an OemConfig Schema without enhancement. This document describes the sorts of additional features that might be required, and provides some examples of how a data-driven UI might be designed to handle these features.

### **Notes About Schema Snippets**

Since most EMMs will likely acquire OemConfig Schemas using the [Google Play EMM API](#), only the JSON version of the Schema will be used when presenting Schema snippets (Google mandates the JSON format). A similar point could have been made looking at the XML version of the Schema, since they are equivalent in capability.

The Zebra OemConfig Schema is quite long, and snippets are therefore excerpted and abbreviated to highlight certain aspects. An ellipsis (...) symbol is used to indicate where significant Schema content was left out. Snippets otherwise use correct JSON syntax.

### **Notes About Data-Driven UI**

A key use case for OemConfig Schemas is for producing a data-driven UI. If an EMM coded a data-driven UI to handle only the restrictive subset of Schema capabilities imposed when uploading non-OemConfig applications to the Google Play Store, that UI would likely be unable to cope with the increased complexity of an OemConfig Schema. It might not be entirely obvious what the data-driven UI should look like for an OemConfig Schema, especially one that layered multiple Bundles within a single Bundle. As an aid to EMMs, Zebra has produced a tool called McTool that comes in the form of a runnable JAR file that can be used as an example of a data-driven UI that supports the full Zebra OemConfig Schema. McTool can consume any Managed Configuration Schema in JSON or XML format, and presents a data-driven UI that allows the creation of Managed Configuration sets based on that Schema. Screen Shots of McTool are used to help demonstrate the UI aspects associated with the Schema snippets.

### **Bundles within Bundles**

A key Managed Configuration Schema capability that only OemConfig applications are permitted to use is to define Bundles within Bundles. This is allowed in Schemas defined per the Android Managed Configuration Model, but any non-OemConfig application that defines a Bundle within a Bundle is rejected by Google from the Google Play Store.

The purpose of a Bundle in a Managed Configuration Schema is to define a data structure that organizes one or more lower-level entities into a group. In an OemConfig application, where various aspects of the device are being configured, there might be different device subsystems that can be configured, each of which potentially provide multiple subsystem-specific settings. Bundles within Bundles allow a hierarchy to be introduced into an OemConfig Schema, which can be used to significantly improve the user experience of a data-driven UI generated based on that Schema.

An example of what this might look like examines a snippet of the Zebra OemConfig Schema:

<pre>{   "restrictionType": "bundleArray",   "key": "steps",   "title": "Transaction Steps",   "description": "Specifies a series of Steps to be performed by OemConfig as part of a single transaction",   "nestedRestriction":   [     {       "restrictionType": "bundle",       "key": "step",       "title": "Transaction Step",       "description": "Specifies an OemConfig Step by specifying an unordered set of operations to be performed as part of that Step",       "nestedRestriction":       [</pre>
...
<pre>        {           "restrictionType": "bundle",           "key": "blacklistStep",           "title": "Blacklist Configuration",           "description": "Specifies Blacklist configuration to be performed as part of this OemConfig Step",           "nestedRestriction":           [</pre>
...
<pre>            {               "restrictionType": "choice",               "key": "blacklistAction",               "title": "Blacklist Configuration - Action",               "description": "Specifies an Action to be performed on a built-in system application",               "entry": [ "Disallow", "Allow" ],               "entryValue": [ "DisableApplication", "EnableApplication" ]             }           ]         }       ]     }   ] }</pre>
...
<pre>    ]   } }</pre>
...
<pre>  ] }</pre>

Referring to the Zebra OemConfig Schema snippet shown above, notice the following:

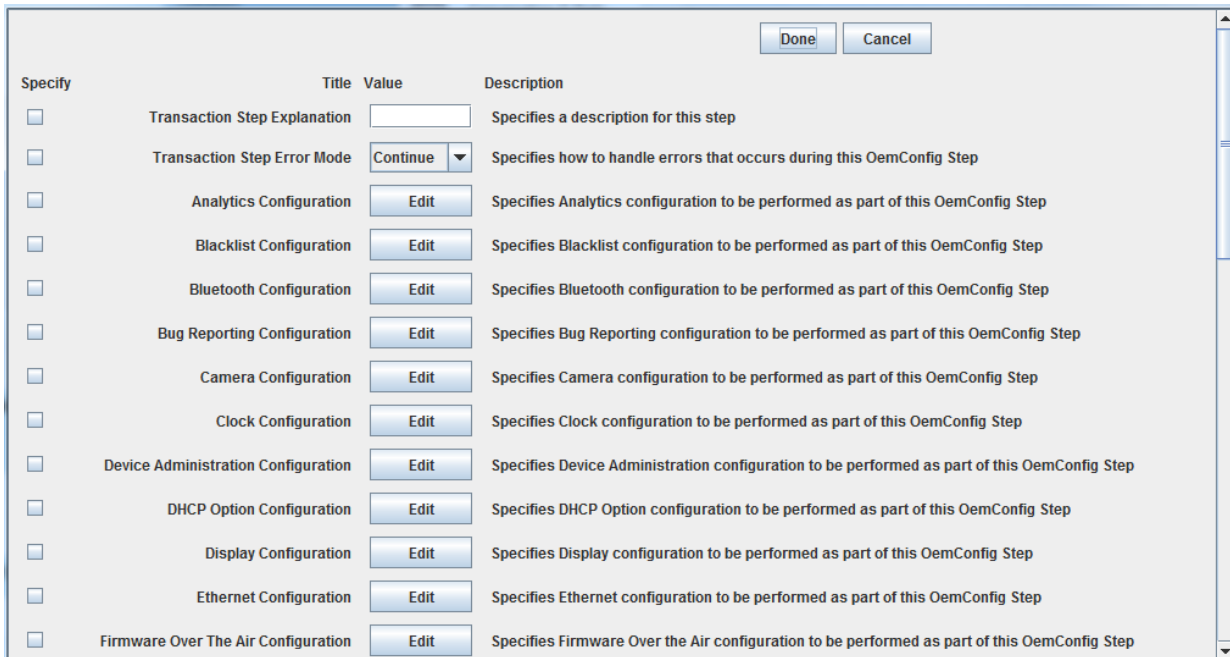
- It defines a **Bundle Array** named “steps” (highlighted in **Yellow**).  
Per the Android Managed Configuration Model, a Bundle Array must contain exactly one Bundle, which defines the data structure that can be repeated any number of times as the elements of the Bundle Array.
- It defines a single Bundle inside the Bundle Array named “step” (highlighted in **light blue**).  
Per the Android Managed Configuration Model and the Google Play Store rules for OemConfig applications, a Bundle can contain any combination of Scalars, Bundles, or Bundle Arrays.  
Per the Google Play Store rules for non-OemConfig applications, a Bundle can contain only Scalars.
- It defines a sub-Bundle named “blacklistStep” (highlighted in **light green**) within the “step” Bundle.

This is one of many sub-Bundles defined within the “step” Bundle, but only one is shown for demonstration purposes.

- It defines a Scalar named “blacklistAction” (highlighted in red) within the “blacklistStep” sub-Bundle, which allows the user to select an action to be performed for controlling whether a built-in system application is blacklisted (prevented from being run) or not (allowed to be run).

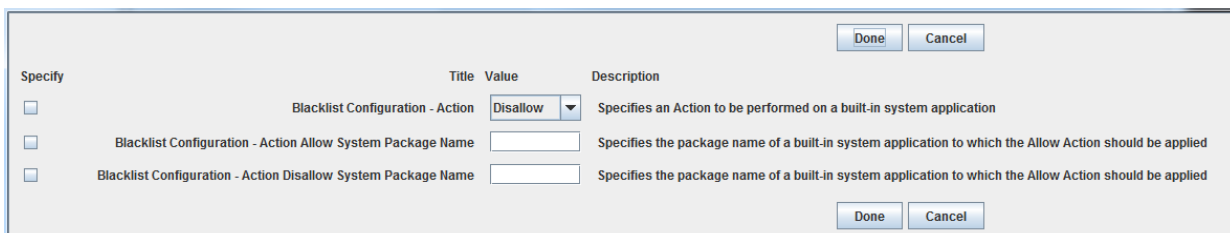
So, what would a Bundle within a Bundle look like in a data-driven UI? Consider the screen shots below from the Zebra McTool used with the Zebra OemConfig Schema.

First, when using McTool with the Zebra OemConfig Schema, the top-level Bundle Array named “steps” allows for any number of instances of the Bundle named “step” to be included. The UI aspects related to supporting Bundle Arrays will be covered later in this section. Once a step is created, McTool presents a UI below:



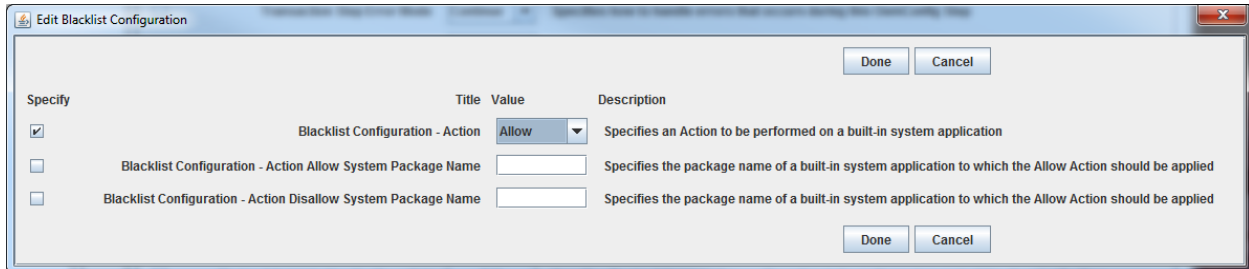
Specify	Title	Value	Description
<input type="checkbox"/>	Transaction Step Explanation	<input type="text"/>	Specifies a description for this step
<input type="checkbox"/>	Transaction Step Error Mode	Continue ▼	Specifies how to handle errors that occurs during this OemConfig Step
<input type="checkbox"/>	Analytics Configuration	Edit	Specifies Analytics configuration to be performed as part of this OemConfig Step
<input type="checkbox"/>	Blacklist Configuration	Edit	Specifies Blacklist configuration to be performed as part of this OemConfig Step
<input type="checkbox"/>	Bluetooth Configuration	Edit	Specifies Bluetooth configuration to be performed as part of this OemConfig Step
<input type="checkbox"/>	Bug Reporting Configuration	Edit	Specifies Bug Reporting configuration to be performed as part of this OemConfig Step
<input type="checkbox"/>	Camera Configuration	Edit	Specifies Camera configuration to be performed as part of this OemConfig Step
<input type="checkbox"/>	Clock Configuration	Edit	Specifies Clock configuration to be performed as part of this OemConfig Step
<input type="checkbox"/>	Device Administration Configuration	Edit	Specifies Device Administration configuration to be performed as part of this OemConfig Step
<input type="checkbox"/>	DHCP Option Configuration	Edit	Specifies DHCP Option configuration to be performed as part of this OemConfig Step
<input type="checkbox"/>	Display Configuration	Edit	Specifies Display configuration to be performed as part of this OemConfig Step
<input type="checkbox"/>	Ethernet Configuration	Edit	Specifies Ethernet configuration to be performed as part of this OemConfig Step
<input type="checkbox"/>	Firmware Over The Air Configuration	Edit	Specifies Firmware Over the Air configuration to be performed as part of this OemConfig Step

Referring to the above screen shot, the Bundle “blacklistStep” within the Bundle “step” is implemented by a button that can be used to edit an instance of that Bundle within the “step” Bundle. The button is preceded by the label “Blacklist Configuration,” which is obtained from the “title” of that element within the Schema, and is labeled “Edit” to indicate that the button allows the user to edit an instance of that sub-Bundle “step” Bundle instance. The “Specify” checkboxes to the far left indicate which elements within the “step” Bundle should be included. Clicking the “Edit” button for “Blacklist Configuration” produces the additional “pop-up” UI shown below:



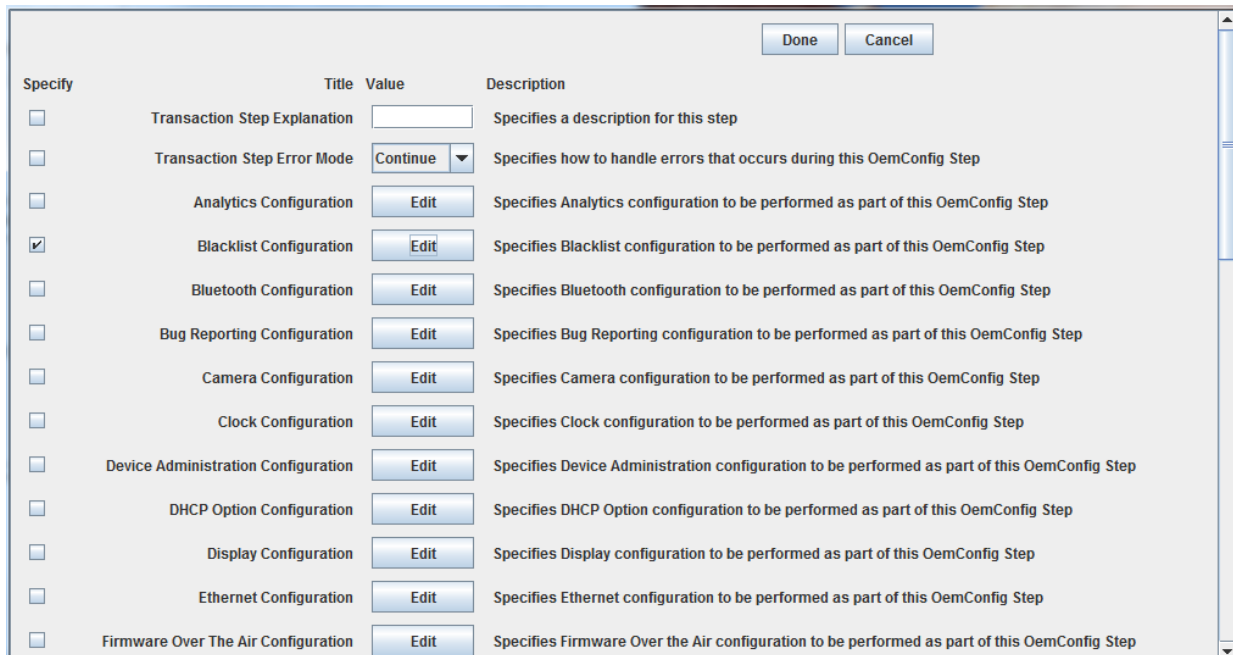
Specify	Title	Value	Description
<input type="checkbox"/>	Blacklist Configuration - Action	Disallow ▼	Specifies an Action to be performed on a built-in system application
<input type="checkbox"/>	Blacklist Configuration - Action Allow System Package Name	<input type="text"/>	Specifies the package name of a built-in system application to which the Allow Action should be applied
<input type="checkbox"/>	Blacklist Configuration - Action Disallow System Package Name	<input type="text"/>	Specifies the package name of a built-in system application to which the Allow Action should be applied

Referring to the above screen shot, the new UI allows all the sub-elements within the “blacklistStep” Bundle to be edited. Note that for simplicity, the snippet above shows only the first element named “blacklistAction,” whereas the UI is showing additional elements present in the Zebra OemConfig Schema. By using this same approach—with an “Edit” button and a new “pop-up” UI—any number of Bundles within Bundles can be handled, as long as they’re defined within the Schema. Also note that when changes are made to any element, the “Specify” checkbox automatically gets checked, although the user could uncheck it at any time if desired. If the value “Allow” is chosen for the element named “blacklistAction,” the UI would change as shown below:



Specify	Title	Value	Description
<input checked="" type="checkbox"/>	Blacklist Configuration - Action	Allow	Specifies an Action to be performed on a built-in system application
<input type="checkbox"/>	Blacklist Configuration - Action Allow System Package Name		Specifies the package name of a built-in system application to which the Allow Action should be applied
<input type="checkbox"/>	Blacklist Configuration - Action Disallow System Package Name		Specifies the package name of a built-in system application to which the Allow Action should be applied

Referring to the above screen shot, the UI has been updated to show that the changed value for the element named “blacklistAction” should be included into the set of Managed Configurations being constructed. If the “Cancel” button is clicked, then this “pop-up” UI would be closed and all “Edit” operations done within it would be discarded. If the “Done” button is clicked, then the “pop-up” UI is closed and all “Edit” operations done within it are included in an instance of the sub-Bundle “blacklistStep” within the containing “step” Bundle. The “Specify” checkbox is set to reflect that, as shown below:



Specify	Title	Value	Description
<input type="checkbox"/>	Transaction Step Explanation		Specifies a description for this step
<input type="checkbox"/>	Transaction Step Error Mode	Continue	Specifies how to handle errors that occurs during this OemConfig Step
<input type="checkbox"/>	Analytics Configuration	Edit	Specifies Analytics configuration to be performed as part of this OemConfig Step
<input checked="" type="checkbox"/>	Blacklist Configuration	Edit	Specifies Blacklist configuration to be performed as part of this OemConfig Step
<input type="checkbox"/>	Bluetooth Configuration	Edit	Specifies Bluetooth configuration to be performed as part of this OemConfig Step
<input type="checkbox"/>	Bug Reporting Configuration	Edit	Specifies Bug Reporting configuration to be performed as part of this OemConfig Step
<input type="checkbox"/>	Camera Configuration	Edit	Specifies Camera configuration to be performed as part of this OemConfig Step
<input type="checkbox"/>	Clock Configuration	Edit	Specifies Clock configuration to be performed as part of this OemConfig Step
<input type="checkbox"/>	Device Administration Configuration	Edit	Specifies Device Administration configuration to be performed as part of this OemConfig Step
<input type="checkbox"/>	DHCP Option Configuration	Edit	Specifies DHCP Option configuration to be performed as part of this OemConfig Step
<input type="checkbox"/>	Display Configuration	Edit	Specifies Display configuration to be performed as part of this OemConfig Step
<input type="checkbox"/>	Ethernet Configuration	Edit	Specifies Ethernet configuration to be performed as part of this OemConfig Step
<input type="checkbox"/>	Firmware Over The Air Configuration	Edit	Specifies Firmware Over the Air configuration to be performed as part of this OemConfig Step

The same process is applied when editing any other sub-Bundle at the same level or when editing sub-Bundles at lower levels.

### **Bundle Arrays within Bundles**

A key Managed Configuration Schema capability that only OemConfig applications are permitted to use is defining Bundle Arrays within Bundles. This is allowed in Schemas defined per the Android Managed Configuration Model, but any non-OemConfig application that defines a Bundle Array within a Bundle will be prevented from being uploaded to the Google Play Store.

The purpose of a Bundle Array in a Managed Configuration Schema is to define an element that can be repeated an indefinite number of times. This is generally required when configuring something that can have multiple instances. The classic example used for this feature in the documentation for the [Android Managed Configuration Model](#) is to configure a list of VPNs. Since there might need to be any number of VPNs defined, and since the configuration of each VPN is inherently independent of every other VPN, an indefinite repeat is a logical approach to use. Non-OemConfig applications are permitted to use Bundle Arrays only at the top-level of their Schemas, not within Bundles. This makes sense, for example, when an application has a list of things to configure, such as user credentials, accounts, etc.

Since an OemConfig application can support configuration of many aspects of a device, and might need or organize hierarchically by sub-systems using sub-Bundles, and when a sub-system has a need to configure an indefinite list of similar entities, a Bundle Array within a Bundle may be the most appropriate way to go. Because of this, OemConfig applications have been granted special dispensation to include Bundle Arrays within Bundles to enable an improved user experience.

The design of the Zebra OemConfig Schema takes advantage of Bundle Arrays within Bundles in a variety of circumstances under which a user might need or want to easily configure multiple similar aspects of a device. For example, when re-mapping the keys on a device keyboard, it is natural to specify an indefinite list of re-mappings to be applied. While not every sub-system that can be configured via Zebra OemConfig Schema needs this level of sophistication, there are several that do.

An example of what this might look like examines a snippet of the Zebra OemConfig Schema:

```
...
{
  "restrictionType": "bundle",
  "key": "keymapStep",
  "title": "Key Mapping Configuration",
  "description": "Specifies Key Mapping configuration to be performed as part of this OemConfig Step",
  "nestedRestriction":
  [
    {
      "restrictionType": "choice",
      "key": "keymapAction",
      "title": "Key Mapping Configuration - Action",
      "description": "Specifies a Key Map Action to be performed",
      "entry": [ "Add Mapping", "Reset All Mappings" ],
      "entryValue": [ "1", "2" ]
    },
  ],
}
```



- It defines a Bundle named “keymapStep” (highlighted in yellow), which is a sub-Bundle under the Bundle named “steps” (not shown).

Per the Google Play Store rules for non-OemConfig applications, a Bundle can contain **only** Scalars.

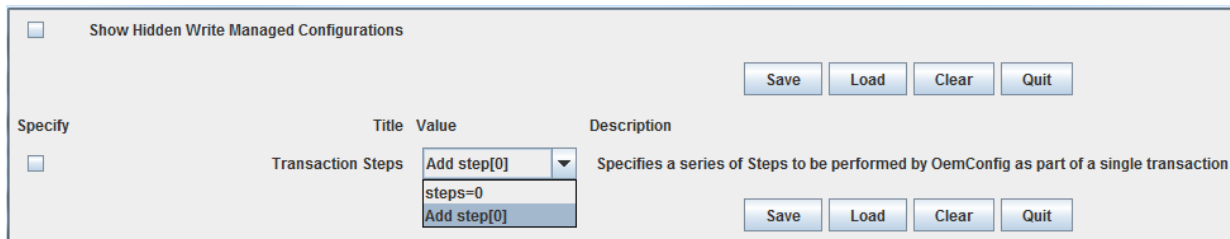
- ZEBRA EMM TOOLKIT

entries are required, whereas if the value “Reset All Mappings” is chosen, then no additional entries are required.

- It defines a Scalar named “keymapActionAddMappingKeyId” (highlighted in light green), which is used when adding a re-mapping to choose an ID value that identifies the key to be re-mapped.
- It defines a sub-Bundle Array named “keymapActionAddMappingBehaviors” (highlighted in red), which specifies a list of behaviors to be assigned to the key being re-mapped.
- It defines a Bundle named “keymapBehavior” (highlighted in magenta), which specifies a set of elements that collectively define a single behavior to be performed when a key is pressed in a specified keyboard state.
- It defines a Scalar named “keymapBehaviorTableName” (highlighted in dark blue), which defines the keyboard state table into which the re-mapping behavior is stored and hence controls the keyboard state that must be active when the key is pressed for the specified behavior to be performed.
- It defines a Scalar named “keymapBehaviorType” (highlighted in dark green), which defines the type of behavior to be performed when the key is pressed while in the specified keyboard state.

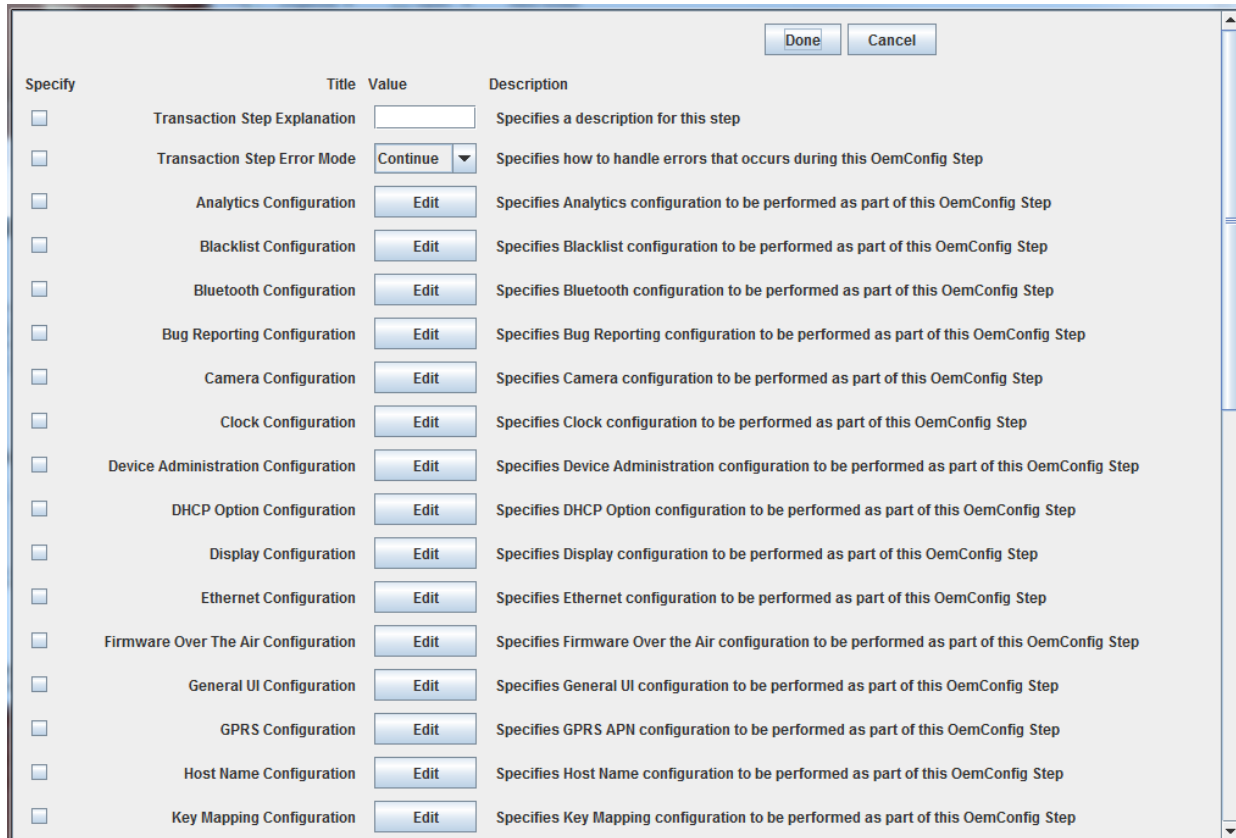
What does a Bundle Array within a Bundle look like in a data-driven UI? It looks much like a Bundle Array at the top-level of a Schema. The only significant difference is the need to be able to support the addition of managing elements at any level of hierarchical context. Since Bundle Arrays at the top-level of a Managed Configuration Schema are supported for all applications uploaded to the Google Play Store, any EMM that supports Managed Configurations should already have some sort of support for handling Bundle Arrays in their data-driven UI.

Before examining how Bundle Arrays look within a sub-Bundle in McTool, let’s examine the data-driven UI produced by McTool for the “steps” Bundle at the top-level of the Zebra OemConfig Schema, as illustrated in the following screen shot:



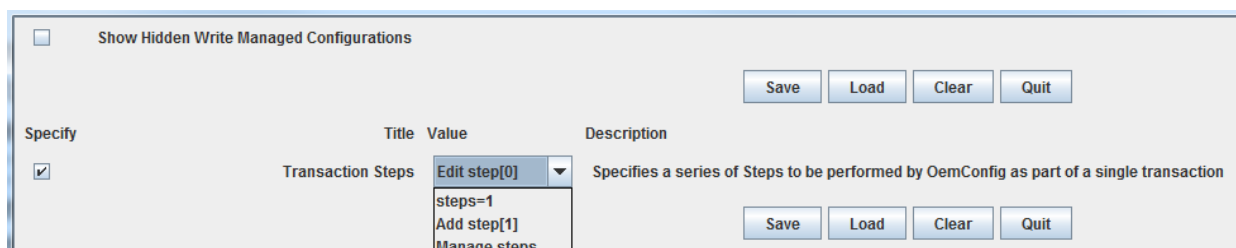
Referring to the above screen shot, the Bundle Array “steps” appears at the top-level of the Zebra OemConfig Schema, and is represented by a “pull-down” list containing the choices “steps=0” and “Add step[0]” preceded by the label “Transaction Steps,” which is obtained from the “title” of that element within the Schema. The value “steps=0” communicates the fact that there are currently no “step” Bundle instances created within that Bundle Array. The value “Add step[0]” enables a first “step” Bundle instance to be created within that Bundle Array.

If the user chooses “Add step[0],” a new “pop-up” UI is presented as illustrated in the following screen shot:



Specify	Title	Value	Description
<input type="checkbox"/>	Transaction Step Explanation	<input type="text"/>	Specifies a description for this step
<input type="checkbox"/>	Transaction Step Error Mode	Continue ▼	Specifies how to handle errors that occurs during this OemConfig Step
<input type="checkbox"/>	Analytics Configuration	Edit	Specifies Analytics configuration to be performed as part of this OemConfig Step
<input type="checkbox"/>	Blacklist Configuration	Edit	Specifies Blacklist configuration to be performed as part of this OemConfig Step
<input type="checkbox"/>	Bluetooth Configuration	Edit	Specifies Bluetooth configuration to be performed as part of this OemConfig Step
<input type="checkbox"/>	Bug Reporting Configuration	Edit	Specifies Bug Reporting configuration to be performed as part of this OemConfig Step
<input type="checkbox"/>	Camera Configuration	Edit	Specifies Camera configuration to be performed as part of this OemConfig Step
<input type="checkbox"/>	Clock Configuration	Edit	Specifies Clock configuration to be performed as part of this OemConfig Step
<input type="checkbox"/>	Device Administration Configuration	Edit	Specifies Device Administration configuration to be performed as part of this OemConfig Step
<input type="checkbox"/>	DHCP Option Configuration	Edit	Specifies DHCP Option configuration to be performed as part of this OemConfig Step
<input type="checkbox"/>	Display Configuration	Edit	Specifies Display configuration to be performed as part of this OemConfig Step
<input type="checkbox"/>	Ethernet Configuration	Edit	Specifies Ethernet configuration to be performed as part of this OemConfig Step
<input type="checkbox"/>	Firmware Over The Air Configuration	Edit	Specifies Firmware Over the Air configuration to be performed as part of this OemConfig Step
<input type="checkbox"/>	General UI Configuration	Edit	Specifies General UI configuration to be performed as part of this OemConfig Step
<input type="checkbox"/>	GPRS Configuration	Edit	Specifies GPRS APN configuration to be performed as part of this OemConfig Step
<input type="checkbox"/>	Host Name Configuration	Edit	Specifies Host Name configuration to be performed as part of this OemConfig Step
<input type="checkbox"/>	Key Mapping Configuration	Edit	Specifies Key Mapping configuration to be performed as part of this OemConfig Step

Referring to the above screen shot, we see the same UI shown earlier when describing Bundles within Bundles since each “step” Bundle can contain any of a variety of sub-Bundles associated with different sub-systems to be configured. When any of these sub-Bundles are configured by using the associated “Edit” button, the “Specify” checkbox associated with that sub-Bundle is checked to indicate that a sub-Bundle is to be included in the set of Managed Configurations created. If the “Cancel” button is clicked, the “pop-up” UI closes and the addition of the new “step” Bundle into the Steps Bundle Array is cancelled, along with any sub-Bundle configurations. If the “Done” button is clicked, then the “pop-up” UI is closed and the addition of the new “step” Bundle into the Steps Bundle Array is confirmed along with any sub-Bundle configurations defined within that “step” Bundle. This results in a return to the prior UI with a slight difference, as shown in the following screen shot:



Specify	Title	Value	Description
<input checked="" type="checkbox"/>	Transaction Steps	Edit step[0] ▼ steps=1 Add step[1] Manage steps	Specifies a series of Steps to be performed by OemConfig as part of a single transaction

Referring to the above screen shot, notice that the value “steps=0” is replaced with the value “steps=1” in the “pull-down” list to communicate that there is now one “step” Bundle instance created within that Bundle Array.





The value “Add step[0]” is replaced with the value “Add step[1]” to communicate that a new instance created within that Bundle Array will be added at the end. Finally, a new value “Manage steps” has been added to the “pull-down” list to invoke a UI to allow existing “step” Bundles within that Bundle Array to be deleted, reordered, viewed or edited. As additional “step” Bundle instances are created within the Bundle Array, the UI continues to be updated to reflect the new array size and the index at which new instances will be created.

Any EMM that already supports Managed Configurations via a Schema-oriented data-driven UI should already have some mechanism to handle all the cases described above for Bundle Array, such as “steps”, that appear at the top-level of a Schema. When used with an OemConfig Schema that has Bundle Arrays within Bundles, such as the Zebra OemConfig Schema, the mechanism used might or might not scale. The mechanism used by McTool is adept at handling Bundle Arrays within Bundles at any level of nesting.

In the Zebra OemConfig Schema snippet presented above, which described keyboard re-mapping, the UI presented for the sub-Bundle “keymapStep” is illustrated in the following screen shot:

Specify	Title	Value	Description
<input type="checkbox"/>	Key Mapping Configuration - Action	Add Mapping	Specifies a Key Map Action to be performed
<input type="checkbox"/>	Key Mapping Configuration - Action Add Mapping Key ID	0	Specifies the identifier of the key to be remapped for an AddMapping Action
<input type="checkbox"/>	Key Mapping Configuration - Action Add Mapping Behaviors	keymapBehaviors=0	Specifies a list of behaviors to be defined for a key for an AppMapping Action

Referring to the above screen shot, we see that the user can select an action to be performed. If the action value “Add Mapping” is chosen, then an ID also should be selected to identify the key to be remapped. Further, one or more behaviors to be performed for that re-mapped key also should be defined. Managing the list of “Behavior” sub-Bundles instances within the “Behaviors” Bundle Array is handled the same way described previously for managing the instances of “step” Bundles within the “steps” Bundle Array. By using a single UI element to represent a Bundle Array, the UI can easily handle any number of Bundle Arrays anywhere within the hierarchical structure. Performing operations on the Bundle Array or elements within it are all handled via “pop-up” UI, thus allowing the UI to scale to any needed depth.