Example Applications

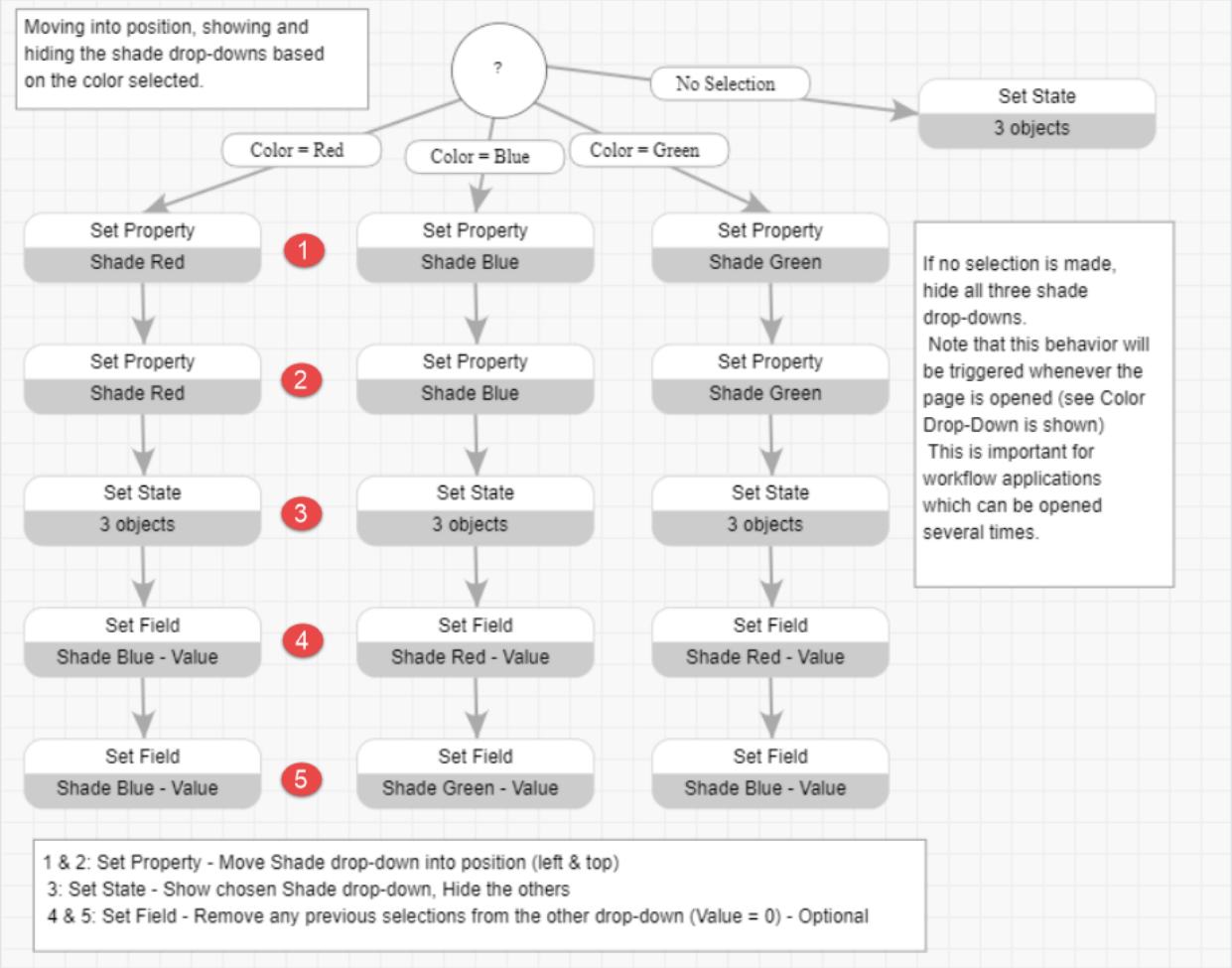
Cascading Drop-downs

This example demonstrates two ways to manage dynamic drop-downs where a selection in one drop-down can dynamically affect the choices in a secondary drop-down.
Note that while this example uses a drop-down for the initial selection, it could just as easily be a radio button group. In either case, only a single initial selection can be made.

Method 1:

This option uses multiple drop-downs for the secondary (*shade*) selection. These drop-down objects are already pre-populated with the secondary selections and are hidden on the page until an initial selection is made.

When a selection is made in the 'Color' drop-down (Check drop-down changes field data), the following logic is triggered:



Example Applications

1 & 2: **Set Property** commands to move the related *shade* drop-down into the correct position. Note that you could place all secondary drop-downs in the same position, in which case this logic would be redundant.

3: A **Set State** command shows the related shade drop-down but keeps the others hidden. Note that if no *color* selection is made, all three *shade* drop-downs are hidden.

4 & 5: **Set Field** commands to reset the unused *shade* drop-downs; this means that if a user selects a *color* and a related *shade* but then changes their choice of *color,* then the related *shade* for the previous *color* is reset to blank.

As there are multiple *shade* drop-down objects, there are multiple objects to look in to discover the actual *shade* selection. From a reporting perspective, this would make it difficult, if not impossible, to report on the *shade* selection.
For that reason, in the example, you will see a 'normally hidden' helper field into which the *shade* selection is passed. Check the logic on the individual *shade* selection 'drop-down changes' behavior events, and you will see a **Copy Fields** command that manages this. By doing so, there is now a single field where the *shade* selection will be stored and readily available for reporting.

Also, note that on the *color* drop-down is shown event, there is a **Run Behavior** command that will trigger the same logic on the *color* drop-down changes event. This behavior will ensure that when the application is opened for the first time, no *shade* drop-downs will be visible and, probably more importantly, when an instance is re-opened later, typically in a workflow environment, the *shade* drop-down and its selection made previously will remain visible.

Method 2:

This option uses only a single drop-down object for the secondary selection.
When a selection is made in the Department drop-down, behavior is triggered using the **Set Selection Items** command that will dynamically populate the secondary employee drop-down object.

Also, note that this method to populate a drop-down object will lead to the first populated option being pre-selected. To prevent this from happening, we can use a **Set Field** command to set the drop-down value to zero, removing any selection in the employee drop-down object after the population.

Summary:
An example is included with this document. Import it into your account, and it should work without any set-up necessary. Just use the preview feature to check the logic.

This example assumes that the secondary drop-downs (shade & employee) options are known at design time. Suppose the secondary drop-down options are unknown at design time but might be taken from a dynamically changing third-party data source. In that case, the second method could be modified so that the options are populated using a Connect command after making the primary selection.