# TLB` LabVIEW Architecture: Exercise Manual

NATIONAL INSTRUMENTS

# Contents

Lesson 1 TLB` Installation and Instance Creation ..... 3
Exercise 1-1 Installing TLB` ..... 3
Exercise 1-2 Creating a New Instance of the TLB` Template ..... 3
Lesson 2 Developing with TLB` ..... 5
Feature Spec ..... 5
State Chart Diagram ..... 6
Exercise 2-1 Adding Controls and Indicators ..... 6
Exercise 2-2 Create System Actions ..... 11
Exercise 2-3 Create System Triggers ..... 14
Exercise 2-4 Completing the State Machine ..... 16
Exercise 2-5 Idiot Proofing the System ..... 20
Feature Creep ..... 23
Closing Thoughts ..... 24

# Lesson 1 TLB` Installation and Instance Creation

## Exercise 1-1 Installing TLB`

1. Go to https://github.com/NISystemsEngineering/TLB-Prime/releases
2. Download the latest version 1.y.z version available. (e.g. ni_tool_tlb`-1.3.0.26.vip
3. Click **Continue**. Accept the agreements that follow to install the TLB` Template.
4. When the package has finished installing, restart LabVIEW.

## Exercise 1-2 Creating a New Instance of the TLB` Template

1. From the tools menu, select Create New TLB` Application.

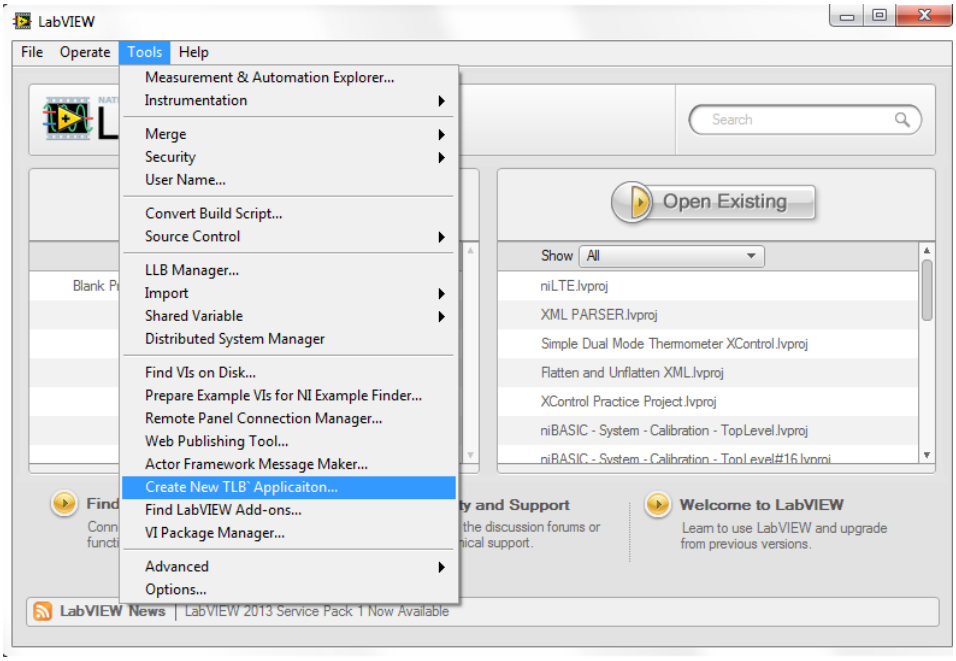


*Figure 1. LabVIEW Splash Screen*

2. Configure the options on the TLB` Springboard as shown in Figure 5 then click **Build.**

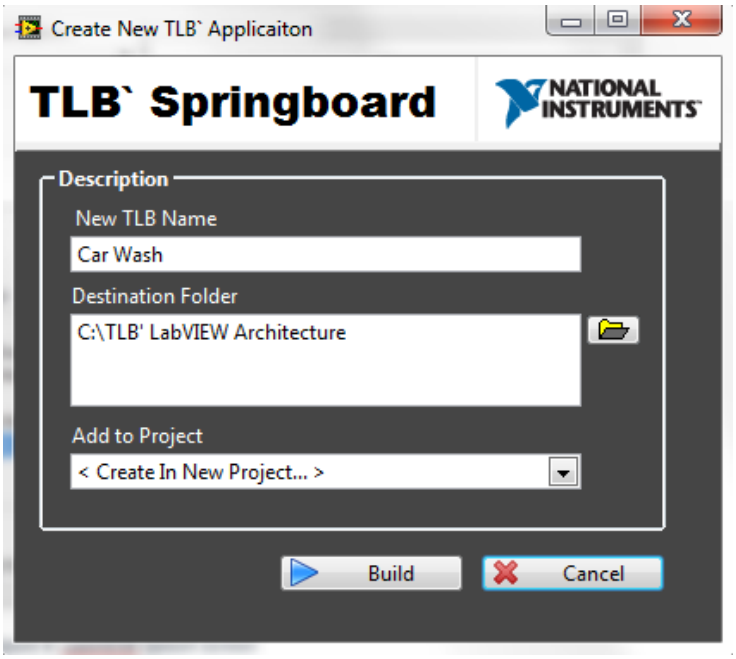


*Figure 2. TLB` Springboard*

3. A project is created. Save the project as *Car Wash.lvproj* in *C:\TLB` LabVIEW Architecture*

4. Rename *MAIN – Template Application.vi* to *MAIN – Car Wash.vi*

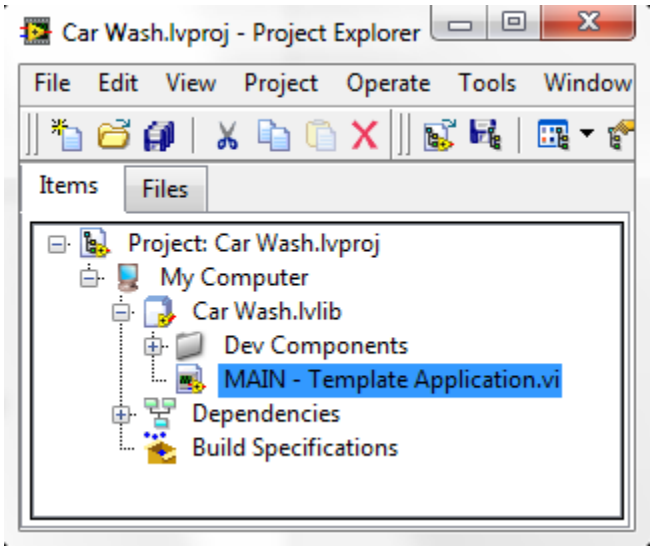


*Figure 3. Project view*

# Lesson 2 Developing with TLB`

The following exercises will guide you through the creation of a Car Wash state machine using TLB`. Before coding, you should always have a plan of action. Write a feature spec for the application and use a state chart diagram to plan out how you want your program to behave.

## Feature Spec

**Inputs**

- Checkboxes for each of the following wash options
  - Pre-Soak
  - Soap
  - Roller Brushes
  - Wax
  - Rinse
  - Giant Blowdryers
- Buttons
  - Start Wash – Sends the System Trigger to go to the Washing state
  - Reset Car – Sends the System Trigger to clear the progress bar and go to the idle state
  - Application Termination – This is already built into the TLB` Template.
    Just click the X like you're closing the window and the VI will stop, but not close.

**Outputs**

- Progress bar – displays how far along the car is in the wash process
- Status – displays current wash step and any other important information for the user

**Operation**

The user should be able to:

- select which options they would like for their car wash
- tell the app when to start the car wash
- reset the car at any time
- stop the application at any time

Additional functionality

- While the car is in the washing state the Start Wash button and wash option checkboxes should be disabled and grayed out.
- All controls should be disabled and grayed out while the program is starting and shutting down

## State Chart Diagram

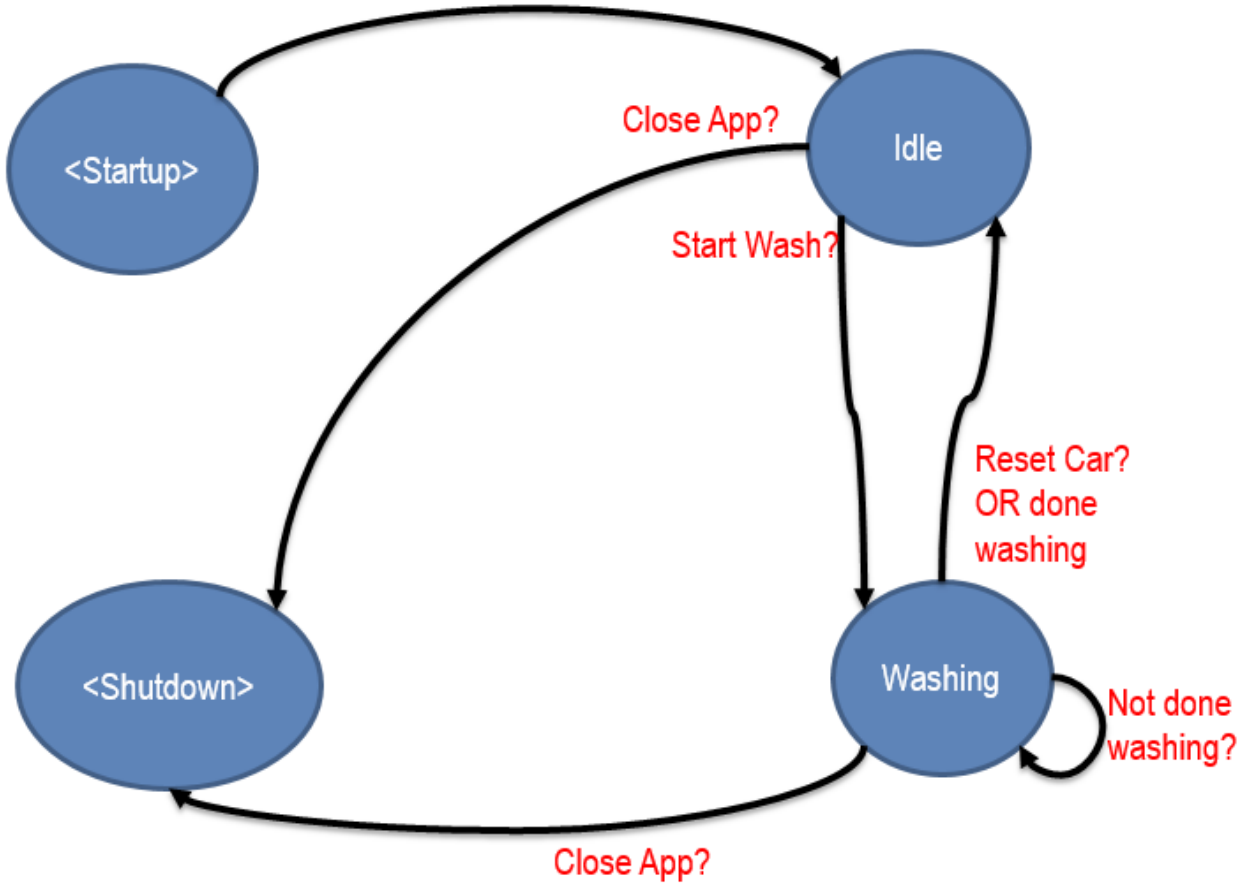


*Figure 4. State Chart Diagram*

## Exercise 2-1 Adding Controls and Indicators

1. Open the *MAIN – Car Wash.vi* and add the appropriate controls and indicators as defined by the feature spec. Also rename the heading and resize as needed.

   **Tip Use** controls and indicators from the System palette for a professional look. **System Checkbox**, **System Horizontal Progress Bar**, **System Button** and **System String** are used in the figure.

   **Tip Don't** forget to change the Labels as well as the Boolean Text for each checkbox. The default labels of Boolean, Boolean 2, etc. are not very helpful while programming.

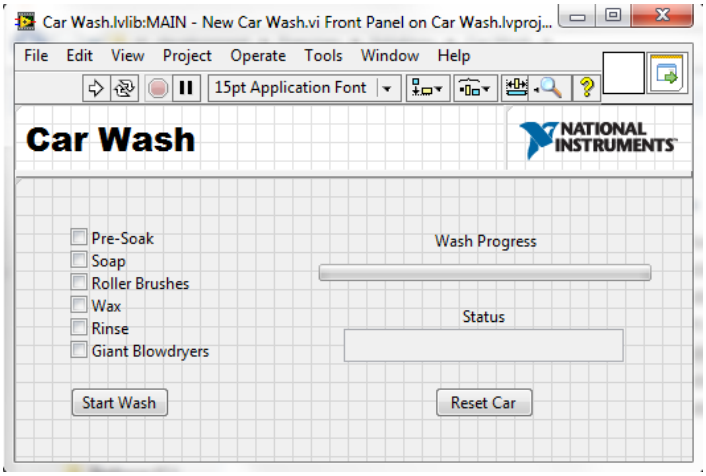


*Figure 5. Car Wash Front Panel*

2. Change the Maximum Scale Range for the Wash Progress Indicator to 6 since we have 6 wash options

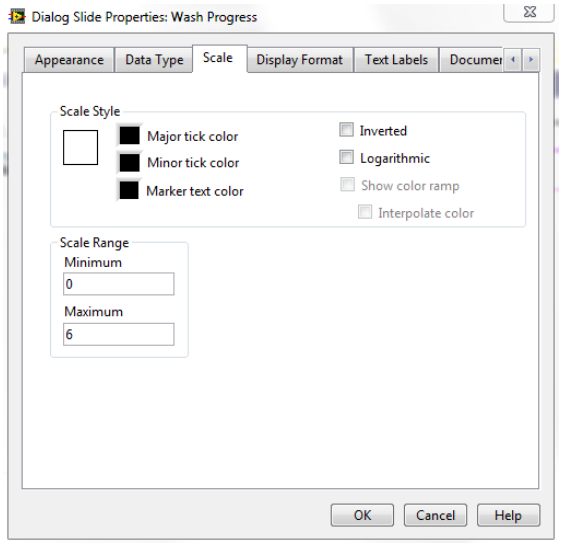


*Figure 6. Wash Progress Scale Properties*

3. Now we need to add our controls and indicators to our Mother Cluster, so we can access it.

    a. Find and double click the Static VI Reference for the *TypeDef - SHIFTER.ctl* to open it. It can be found below the blue Primary Execution Loop on the Block Diagram of *MAIN - Car Wash.vi.*


*Figure 7. Static VI Reference for TypeDef - SHIFTER.ctl*

    b. Copy and paste the check boxes and the Wash Progress indicator into the appropriate User Data and Display Data clusters as shown in the figure.

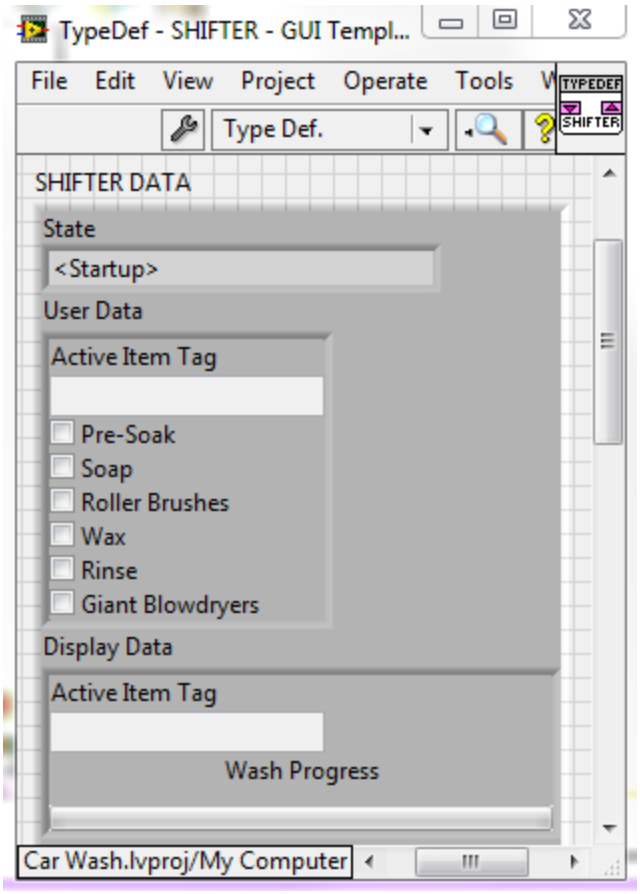


*Figure 8. TypeDef - SHIFTER.ctl Front Panel*

4. While we're here, add an array of Booleans, an array of strings, and a Boolean to the Private Data Cluster. We'll use these later. Label them as shown in the figure.


*Figure 9. Private Data Cluster*

5. Save and Close *TypeDef – SHIFTER.ctl.*

6. Switch to the Block Diagram of *MAIN – Car Wash.vi.* Move all the checkbox controls into the "Cache User Parameters" case of the System Actions / Messages case structure in the Primary Execution Loop. Use **Bundle by Name** to complete the block diagram in the following figure.

   **Tip** Open quick drop with ctrl+space then move labels to the left of your controls and references with the ctrl+t Shortcut Key.

   **Tip Align** » Right Edges and Distribute Objects » Vertical Compress save a lot of time when organizing your block diagram. If you also moved your labels to the left of your controls and references, then everything lines up perfectly with the bundle by name function!

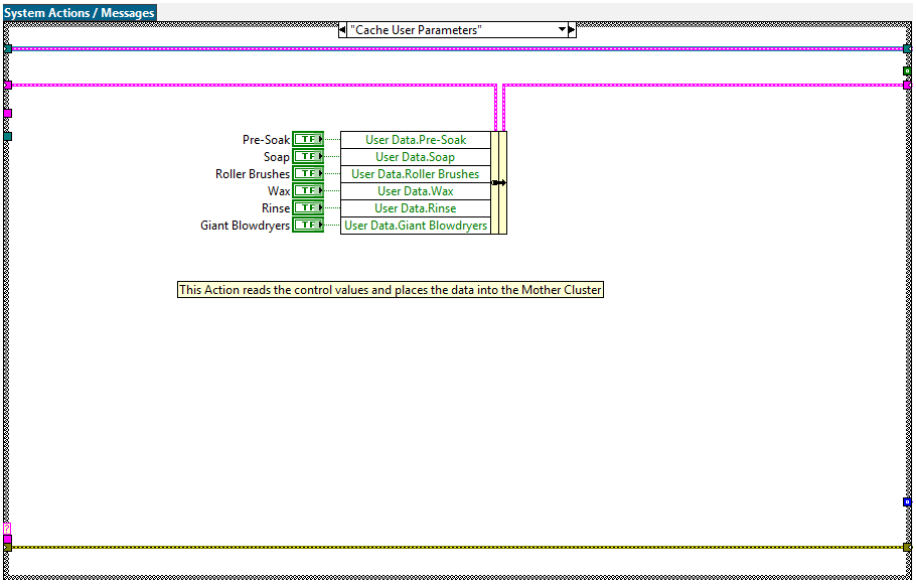


*Figure 10. Cache User Parameters Action*

7. Navigate to the "Register User Parameters" case and create the block diagram in the following figure with **References** for the checkbox controls and **Build Array**.

   **Tip Create** multiple References at once by highlighting all of the control terminals then Right-Clicking and choosing **Create » Reference**.

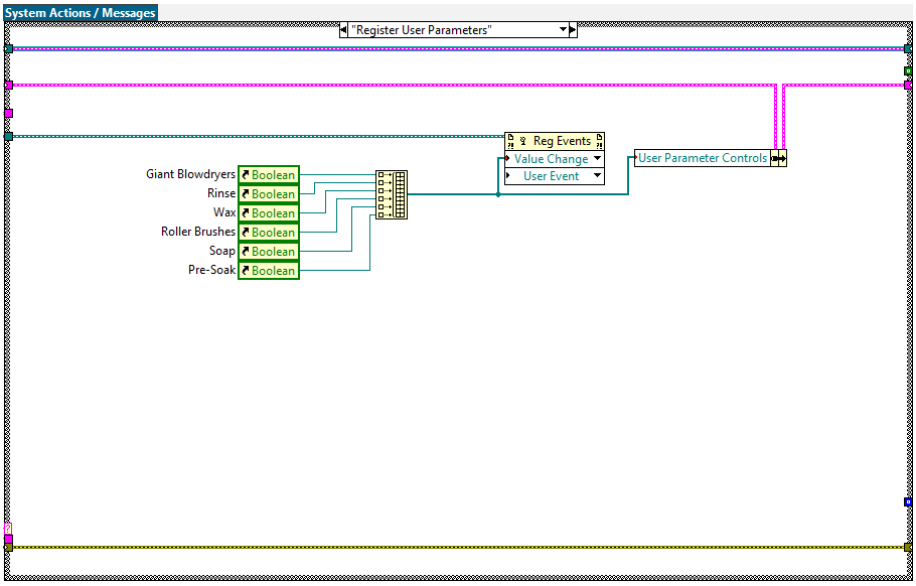


*Figure 11. Register User Parameters Action*

8. Complete the Update Display Action with your indicators and an **Unbundle by Name** function

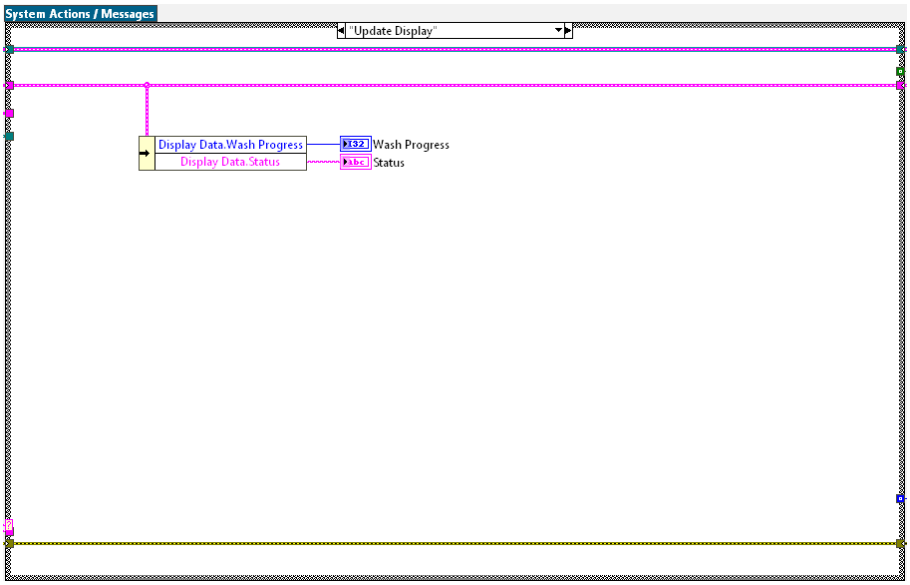


*Figure 12. Update Display Action*

9. Run the VI to observe its operation. It shouldn't do a whole lot yet, but it's nice to have a runnable VI with basic application functionality. Stop the VI by clicking the **X** in the top right - the template has a Panel Close event that stops the VI instead of closing it.

## Exercise 2-2 Create System Actions

There are many ways to accomplish the functionality described in the Feature Spec. This exercise shows a methodology that uses only three extra System Actions and a modified Initialize Action.

Initialize – Reset the Wash Progress indicator to 0 and programmatically place the labels of the wash option checkboxes in an array of strings.

Create Steps Array – Creates and array of the data from the wash option checkboxes and feeds it into the Step Booleans Array.

Increment Step – Increments the Wash Progress indicator and checks to see if the wash cycle is done.

Evaluate Step – Uses the value of the Wash Progress indicator to pull the correct index out of the Step Booleans Array to evaluate if the specific wash option will be performed or not.

1. The “Initialize” case has already been created for us, we just need to modify the behavior. In the “Initialize” case, use **References**, a **For Loop**, a **Property Node** (Label.Text), the **Build Array,** and the **Bundle by Name** functions to create the Step Names Array. Also set Wash Progress to 0, Wash Done? to False.

   **NOTE** The order of the array is important. It should be the same as the order of controls on the front panel. We could also use the User Parameter Controls data in the mother cluster to accomplish this, but explicitly using the references is more self-documenting and makes the order of them obvious.

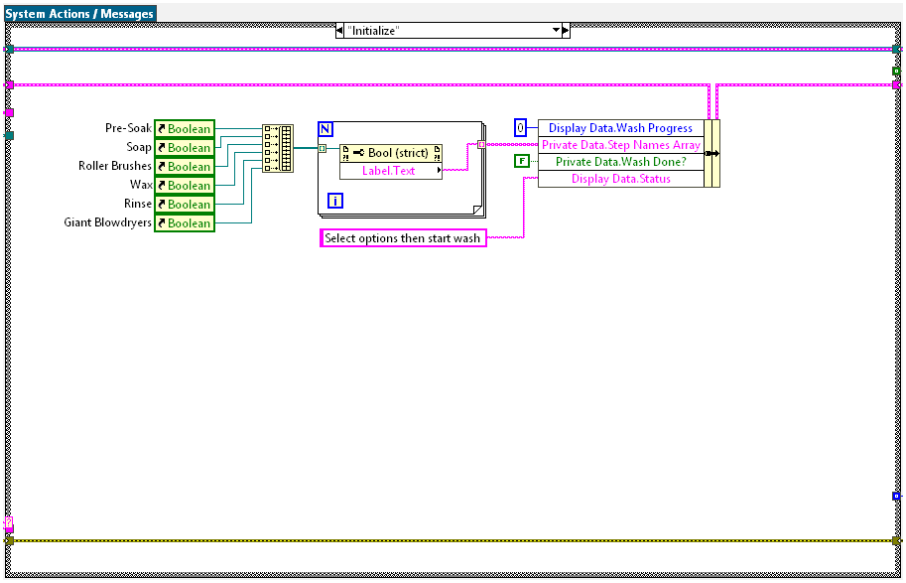


*Figure 13. Initialize Action*

2. Switch to the “/*********Primary Execution********/” case and then Right-Click on the case structure and select **Add Case After**. Name the case “Create Steps Array”. Complete the “Create Steps Array” System Action with the **Unbundle By Name**, **Build Array**, and **Bundle By Name** functions.

**NOTE** The order of the array is important.

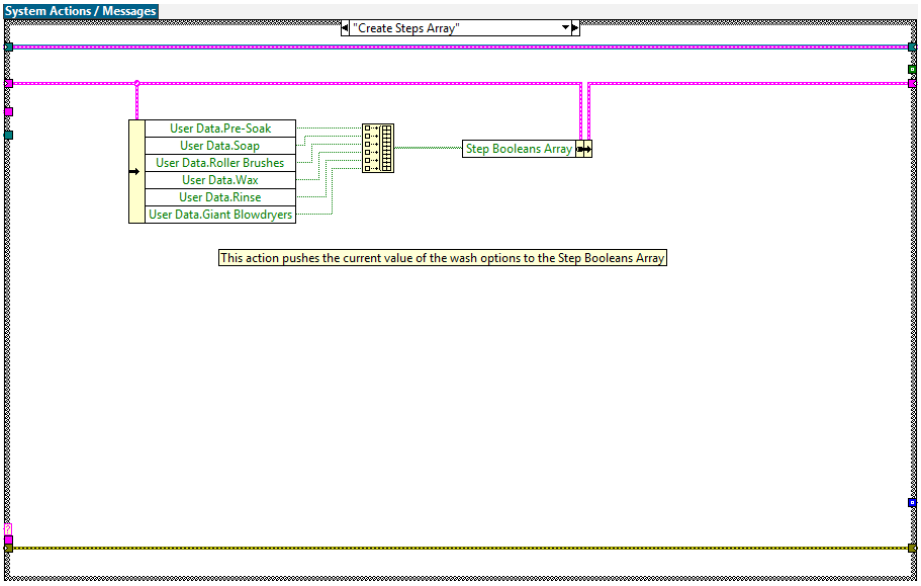


*Figure 14. Create Steps Array Action*

3. Add another case after "Create Steps Array". Call the new Action "Increment Step". Here we want to **Increment** the Wash Progress value and compare the **Array Size** of the Step Booleans Array with the value of Wash Progress. A good coding practice is to use **Greater or Equal?** instead of Equal? to compare a count variable to another number, that way if the count variable is inadvertently incremented too far or does not end up being the same as the number it's being compared against, you will not be stuck in an infinite loop.

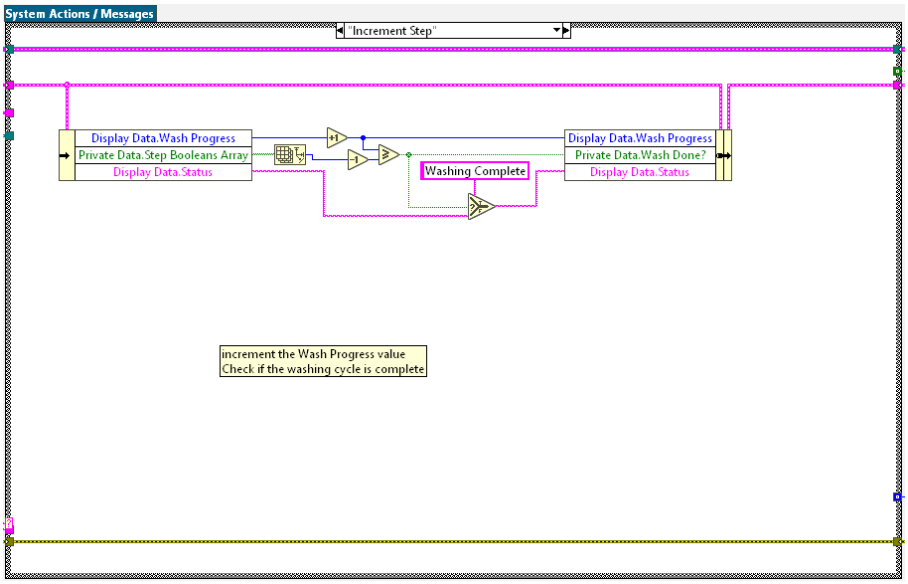


*Figure 15. Increment Step Action*

4. Add another case after “Increment Step”. Call the new case “Evaluate Step”. Use **Index Array** functions to extract the current step from the Step Booleans Array and Step Names Array. Use a **Case Structure** to evaluate whether to execute the wash option or do nothing. Use a **Wait (ms)** VI to simulate the wash option. Update the status with a **Local Variable**.

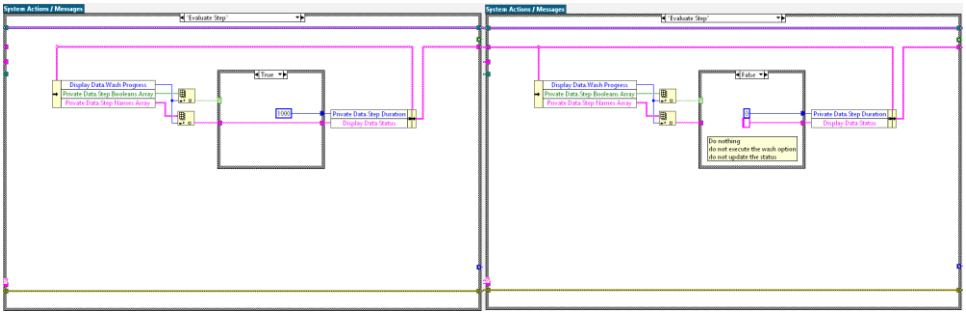

*Figure 16 (a/b). Evaluate Step Action. Note that there is only 1 case structure inside this action. Both cases are shown in the figure.*

5. Add a timeout modification to the transition complete Message. The goal is to use the timeout feature of the queue message handler at the end of going through a set of messages. In essence we will send the message to Update, Increment, Display etc and the wait 1 sec (1000 ms) for the state machine to trigger itself and go through the washing steps.

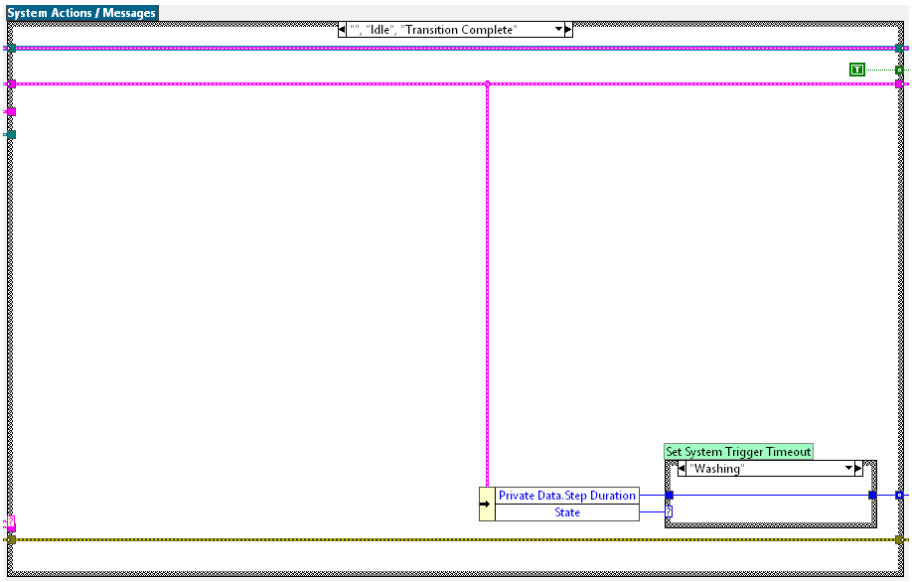


*Figure 17 (a/b). Evaluate Step Action. Note that there is only 1 case structure inside this action. Both cases are shown in the figure.*

6. Take the Car Wash for a test drive. Notice that the behavior of the application has not changed since you last ran it. This is one of the great benefits of TLB` – the actions are typically completely independent from the flow of the program.

## Exercise 2-3 Create System Triggers

Triggers are injected into the system using the EnQueue Trigger.vi. The triggers are then received by the DeQueue Trigger.vi in the System States case structure of the Primary Execution Loop.

1. Add Start Wash and Reset Car to the System Triggers Enum

    a. You can find the System Triggers Enum to the right of the yellow Event Handler Loop. Right click on the enum and select **Open Type Def.**

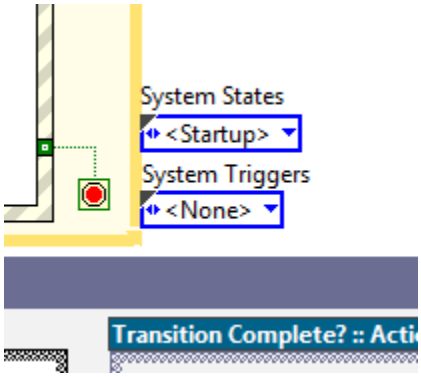


*Figure 18. The System States and System Triggers Enums. They are placed on the BD here as a shortcut for copying and editing.*

    b. Edit the items in the *System Trigger.ctl* to include the Start Wash and Reset Car items.

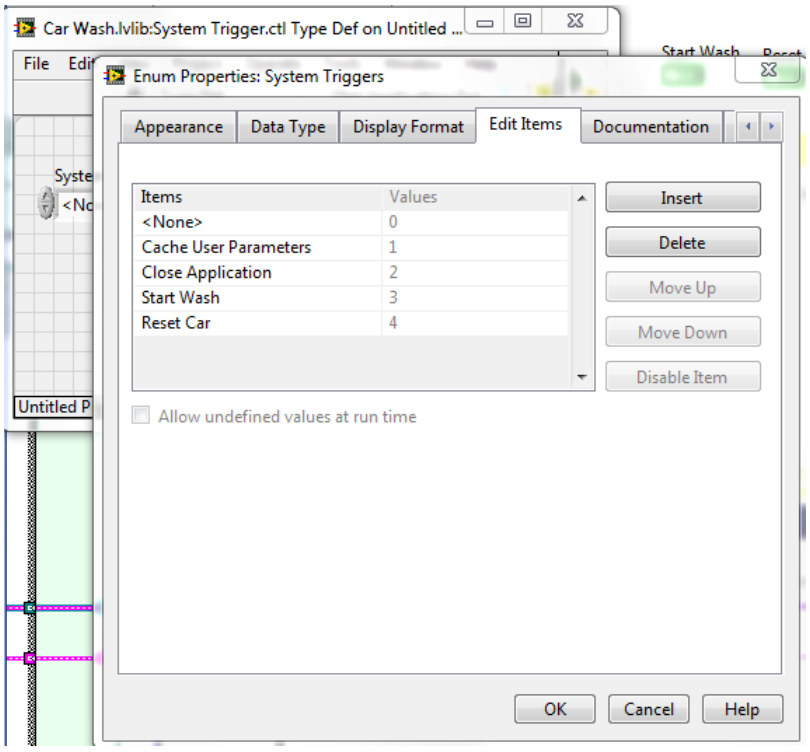


*Figure 19. System Triggers Enum Properties.*

    c. Save and close *System Trigger.ctl* when done editing the items.

2. Add an event case to the event structure in the Event Handler Loop and configure it as shown.

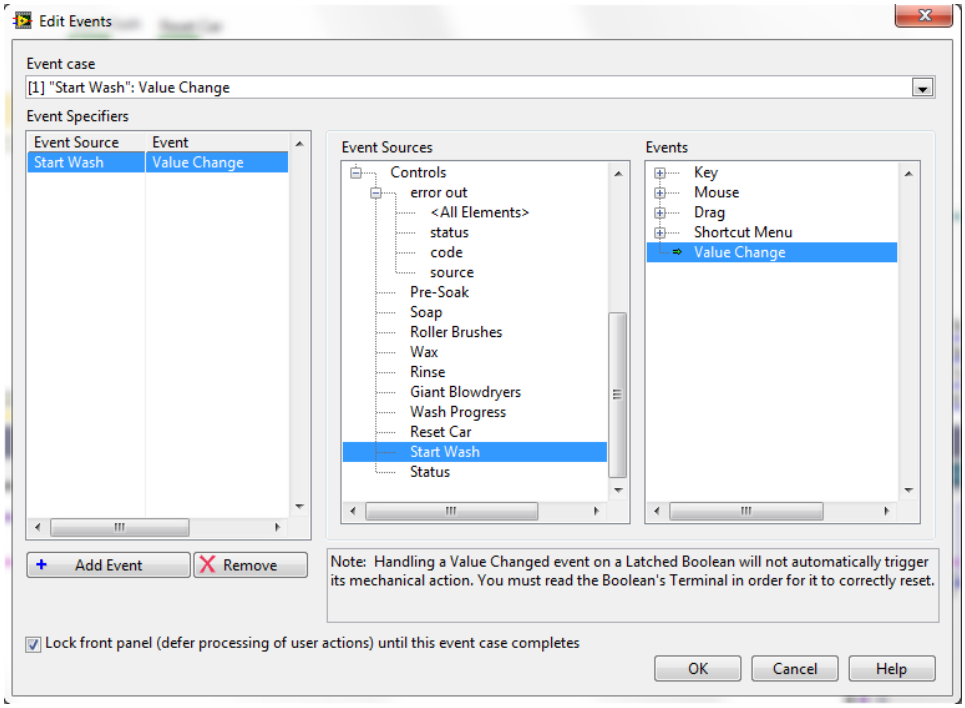


*Figure 20. Edit Events Dialog*

3. Complete the Block Diagram as shown. Note that the **Enqueue Trigger.vi** is conveniently located above the Event Handler Loop for easy copying.

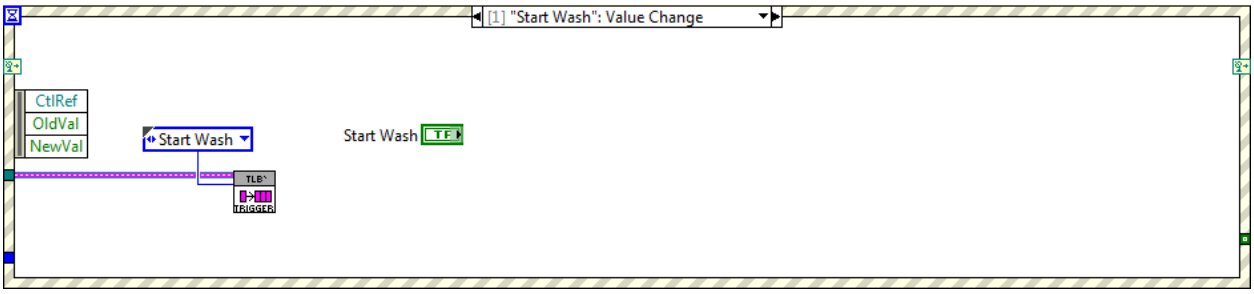


*Figure 21. Start Wash Event*

4. Repeat the above steps to create the Reset Car event.

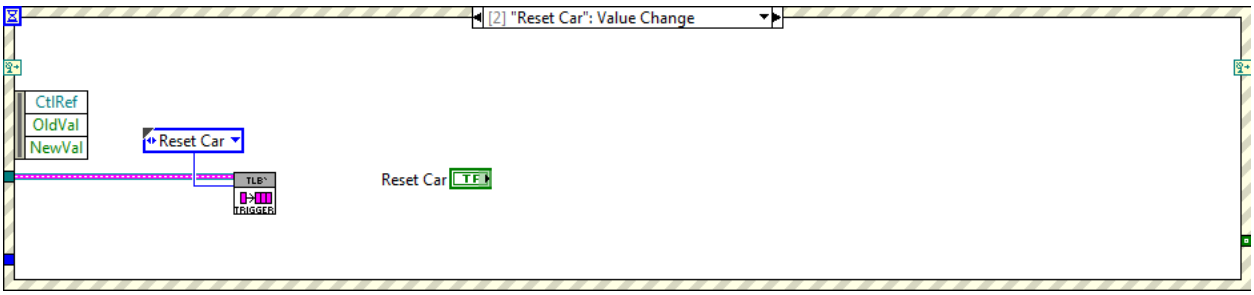


*Figure 22. Reset Car Event*

5. Run the Application. Notice that the Start Wash and Reset Car buttons are now being read by the system. In previous test drives the buttons behaved as a toggle because we were not reading the value of the control. Remember that a latch Boolean will reset itself after LabVIEW has read the value.

## Exercise 2-4 Completing the State Machine

We created our system triggers and our actions, now let's tell the program when and how to use our new triggers and actions. Reference the State Chart Diagram we created in Figure 7 to create the proper modifications to the operation of the Car Wash.

Find the System States case structure in the Primary Execution Loop. This case structure will represent which state our program is in. There is another case structure inside each of the states (or cases) that handles the triggers. This is the System Triggers case structure. This architecture allows each state to respond to a given trigger uniquely. For the following exercise I will refer to these cases as states and triggers. I.e. the Idle state is the Idle case of the System States case structure.

1. Similarly, to how we edited the System Triggers enum, we now need to edit the System States enum to include our Washing state. Open the Type def. and add the Washing item. Save and close *State Declaration.ctl* when done editing.

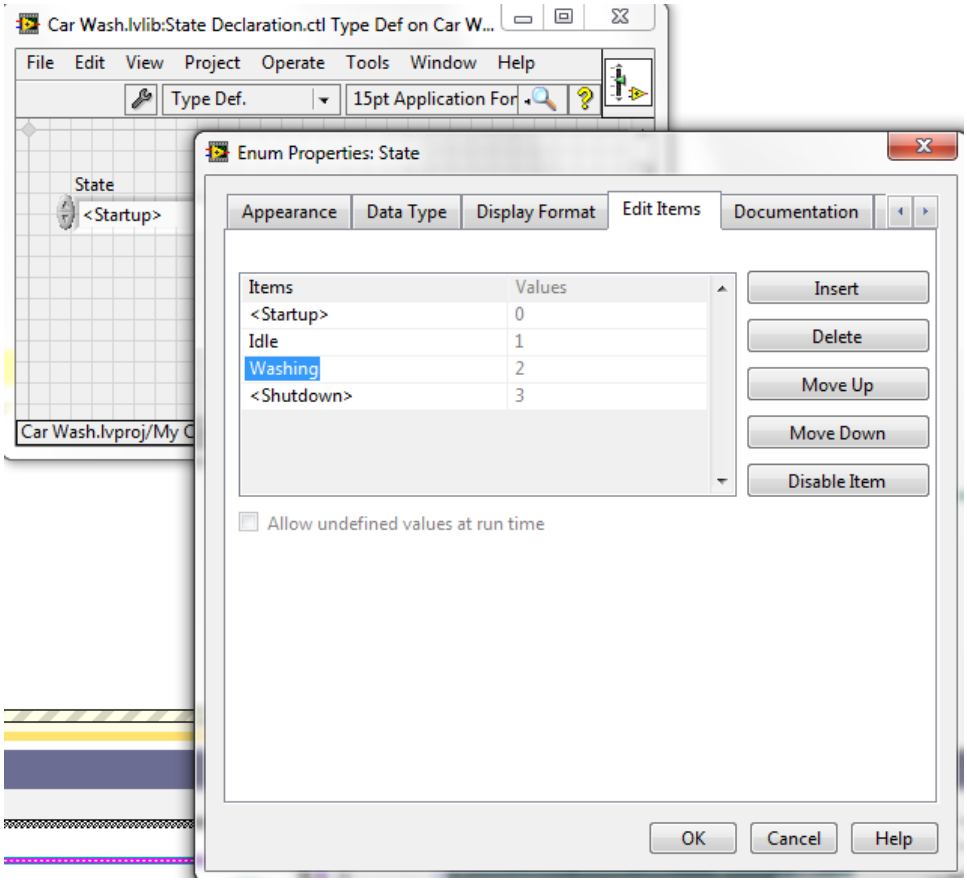


*Figure 23. Properties Dialog*

2. Navigate over to the Idle state on the Block Diagram. Create the appropriate trigger handling code by adding cases to the System Triggers case structure. The Close Application, Cache User Parameters and <None> trigger handling cases are fine as is.

   **NOTE** the Send Local Messages.vi is conveniently located for copying on the block diagram below the Primary Execution Loop

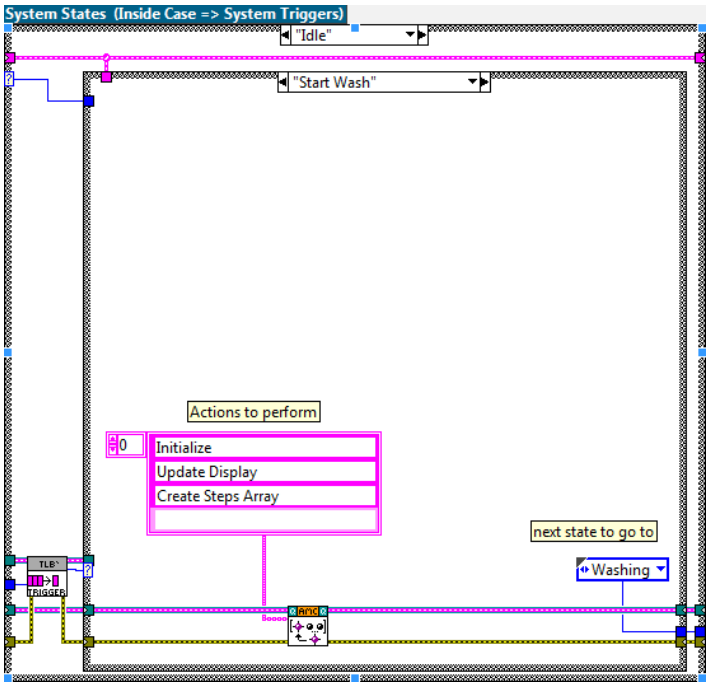


*Figure 24. Start Wash Trigger Handler in the Idle State*

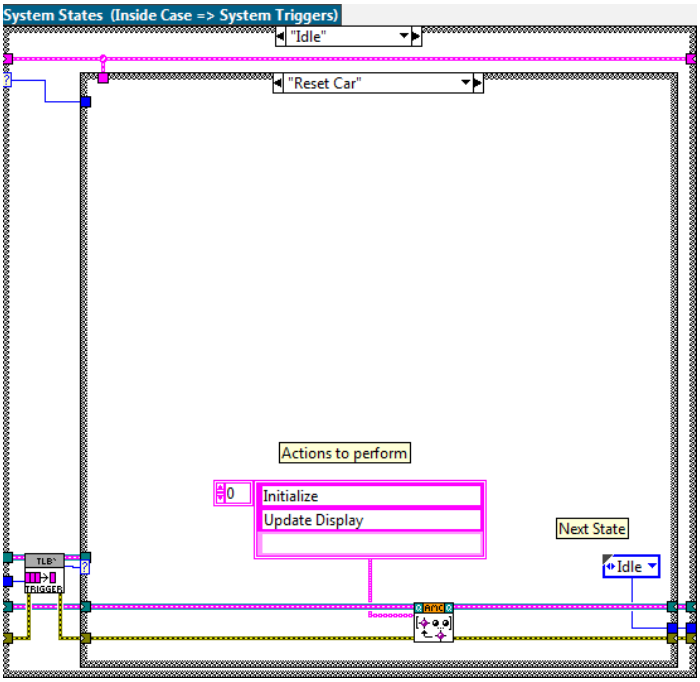


*Figure 25. Reset Car Trigger Handler in the Idle State*

3. Create the Washing state by duplicating the Idle state

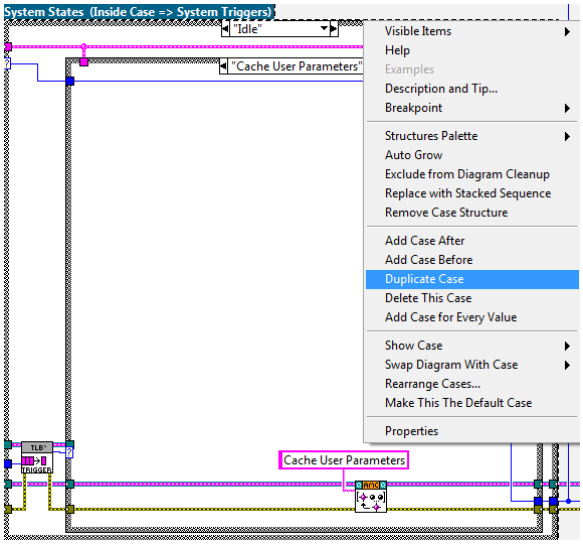


*Figure 26. Duplicating the Idle State*

4. Modify the <None> Trigger Handler as shown. The <None> trigger is executed when no other triggers are present. Use **Select** functions or a singular case structure to handle the conditional nature of this case.

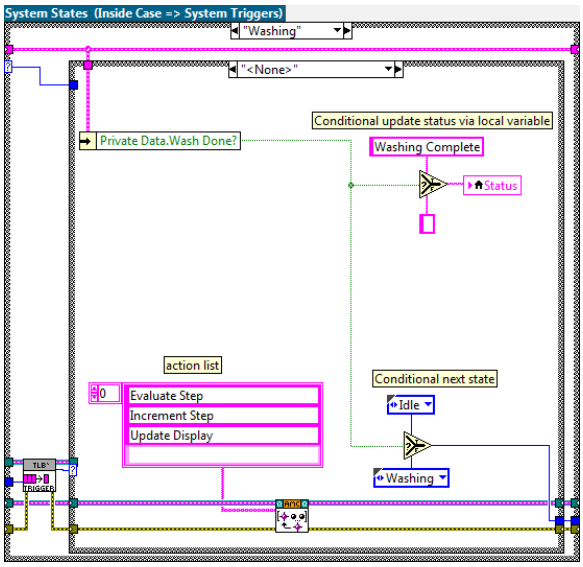


*Figure 27. <None> Trigger Handler in Washing State*

5. Fix the error that we introduced when we added the Washing State by adding a case to the Set UI Configuration case structure in the Change State Action. We'll explore this Action further in the next exercise.

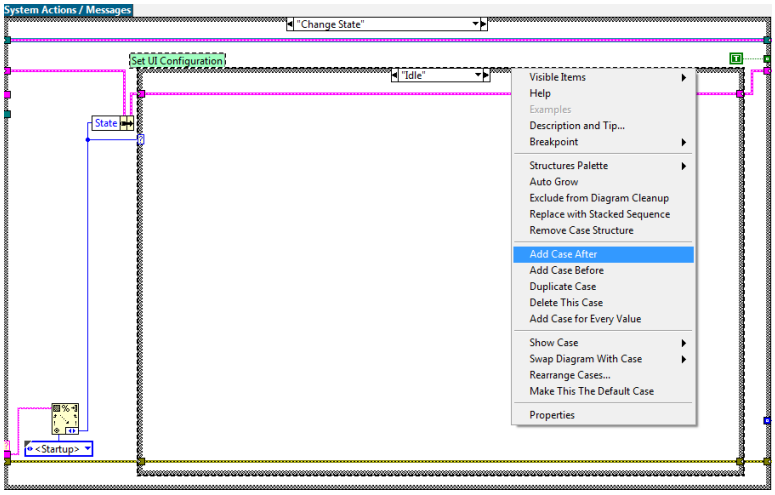


*Figure 28. Fixing the Error*

6. Run the application and observe the operation. Notice that the wash progress lags the status indicator. Let's explore why that is.

   There are two hidden actions that occur without the user queuing them up. These two actions are "Change State" and "Transition Complete".

   The first one, "Change State" is sneakily EnQueued every time the next state is not equal to the current state. This snip of code can be found to the right of the System States case structure in the Primary Execution Loop

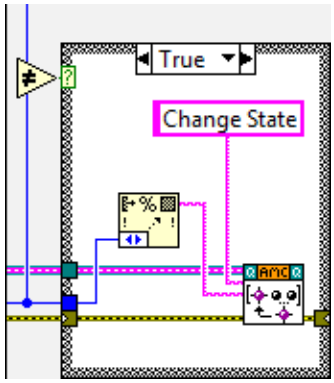


*Figure 29. Change State is EnQueued when the previous state value is not equal to the next.*

   The second hidden action is "Transition Complete". As the name suggests, the action is executed when a transition, or a list of actions that are EnQueued because of a trigger handling case being executed, is completed.

   Notice that the trigger timeout in the "Default" case of the Set System Trigger Timeout case structure in the "Transition Complete" action is set to 1000ms. Lower this timeout to 1ms for a more responsive UI.

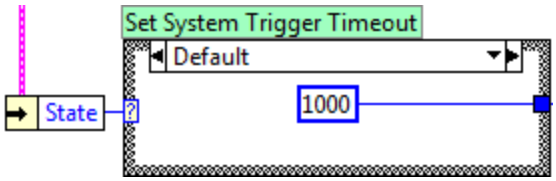


*Figure 30. 1ms timeout*

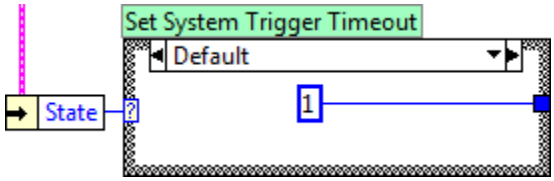


*Figure 31. Lower the timeout to 1ms*

You should only have to do this if you are using the <None> trigger handler like we are instead of receiving an explicit trigger. The <None> trigger is only sent after this timeout value has expired, therefore causing our <None> trigger handler not to execute for an extra second.

7. Run the application again to verify that the Wash Progress and Status indicators are now synced.

## Exercise 2-5 Idiot Proofing the System

I bet you're wondering why we had to have an action called "Register User Parameters". The most common use for the User Parameter Controls data is for batch enabling and disabling of front panel objects.

Let's say that you are in a real automatic car wash. What if the next customer comes up behind you, puts in their money and starts the wash. What happens to your car wash? In our application your car wash would start over. This is not ideal or expected behavior for a real-life car wash, so let's make sure it can't happen.

Batch enabling and disabling of controls is another place where TLB` shines. Ideally, we would only need to change whether controls are enabled or disabled when a state changes. Good thing we have an action for that already.

1. In the "Change State" action complete the following block diagrams for each of the Set UI Configuration cases. **Unbundle by Name**, **Build Array**, **Explicit Property Nodes**, **For Loops** and **References** are used.

   **Tip After** you make one of the cases, take advantage of the ability to duplicate cases to save time.

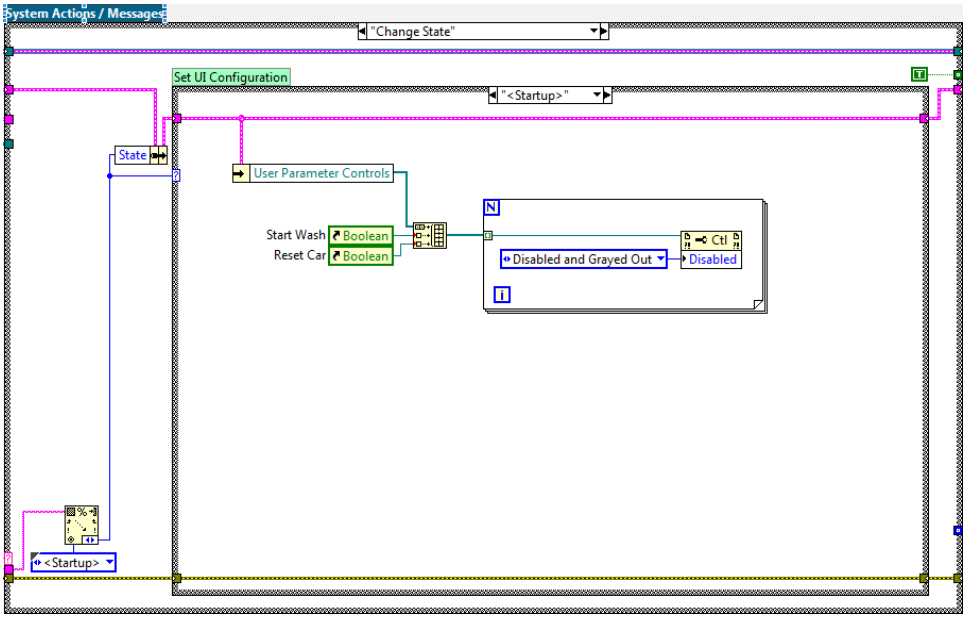


*Figure 32. <Startup> UI Configuration in the Change State Action*

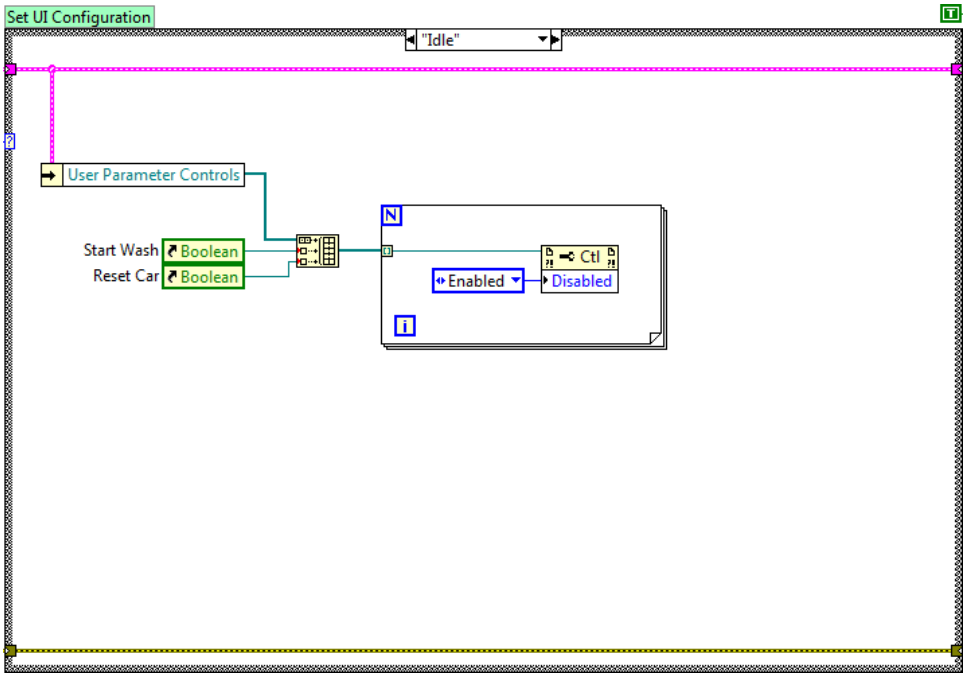


*Figure 33. Idle UI Configuration in the Change State Action*

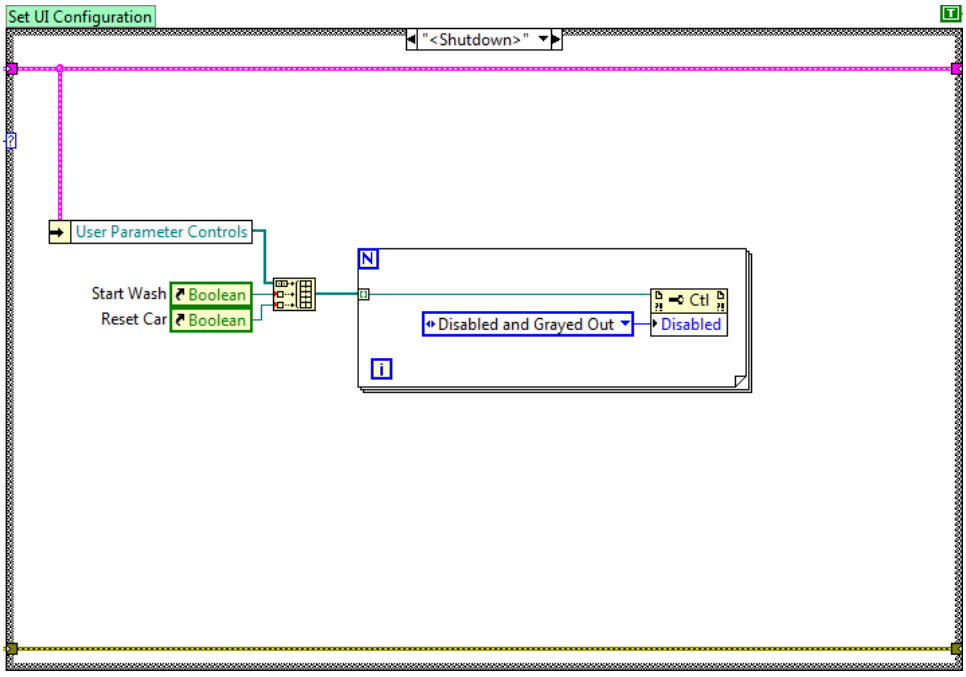


*Figure 34. <Shutdown> UI Configuration in the Change State Action*

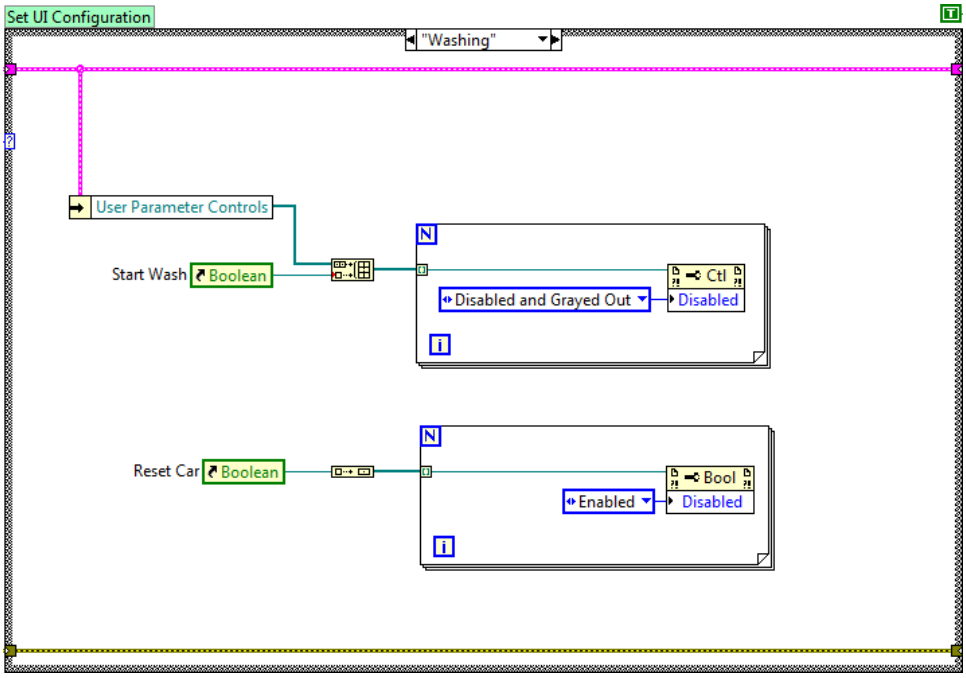


*Figure 35. Washing UI Configuration in the Change State Action*

2. Run the VI to verify that the controls are appropriately disabled and grayed out while the car wash is running.

## Feature Creep

What other features do you think this application could use? Here's a list of possible features:

- Select All wash options button
- Unselect All button
- On application close, clear the wash option checkboxes
- Dynamic checking of elapsed time for each wash option instead of using the hard Wait (ms) vi. This allows for mid-step termination of the application since you could architect it to allow for injection of a Close Application trigger

Think about how you might implement these features. What you'll find is that adding actions and new triggers is very easy when you get the hang of it. The best part is that adding new actions and triggers doesn't break existing actions or trigger handling code!

## Closing Thoughts

There are many ways to implement the car wash application in these exercises. Feel free to experiment with other solutions if you can think of improvements. Here's some general guidelines to keep in mind when working with TLB`.

Dos

- It's okay to inject a trigger programmatically. Feel free to have "internal triggers" that are caused by a condition being met rather than a button being pushed on the front panel. You can inject a programmatic trigger in an action or in the state machine. Note that if you inject a trigger while in a trigger handling case it will execute the injected trigger in the next state, not the current one (unless the next state is the current state).
- Enabling and Disabling of controls should generally be taken care of in the Change State System Action.
- Feel free to create other clusters of control references other than the User Parameters if you will have different groups of controls that need to be enabled and disabled independently of each other.

Don'ts

- Don't put your button triggers in the Cache User Parameters or Register User Parameters actions. Those terminals need to be in their respective event cases so that it gets reset when read. Also, if you register the control as a user parameter it will make enabling and disabling your trigger buttons independently of your regular user parameters difficult.
- Don't enqueue actions in actions, keep flow control in the state machine. If you feel like you want to enqueue an action during an action, inject a trigger instead and let the state machine handle the trigger.