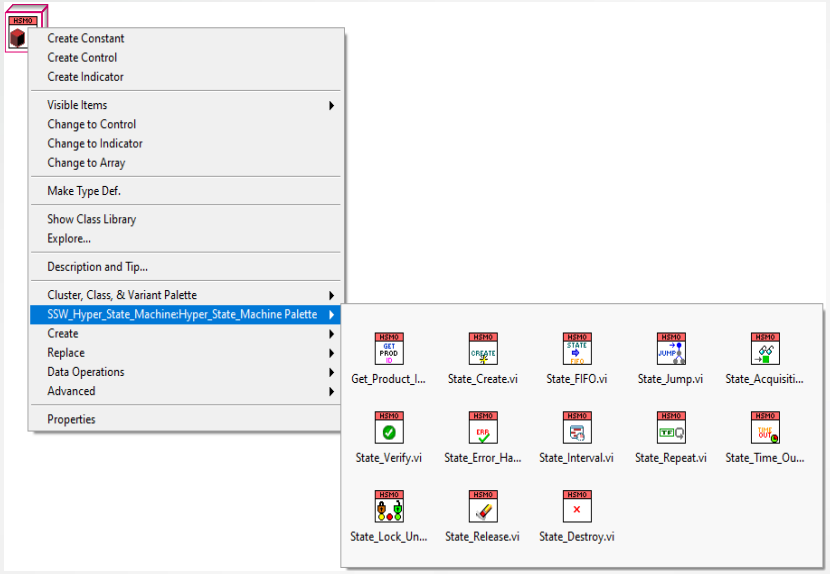
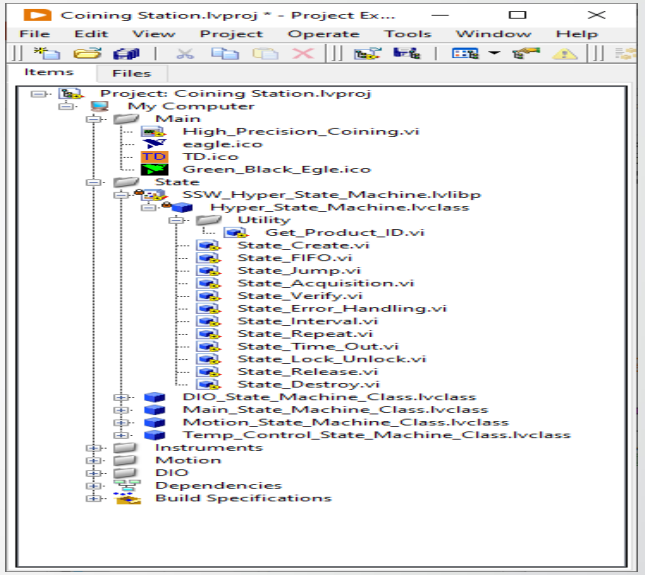
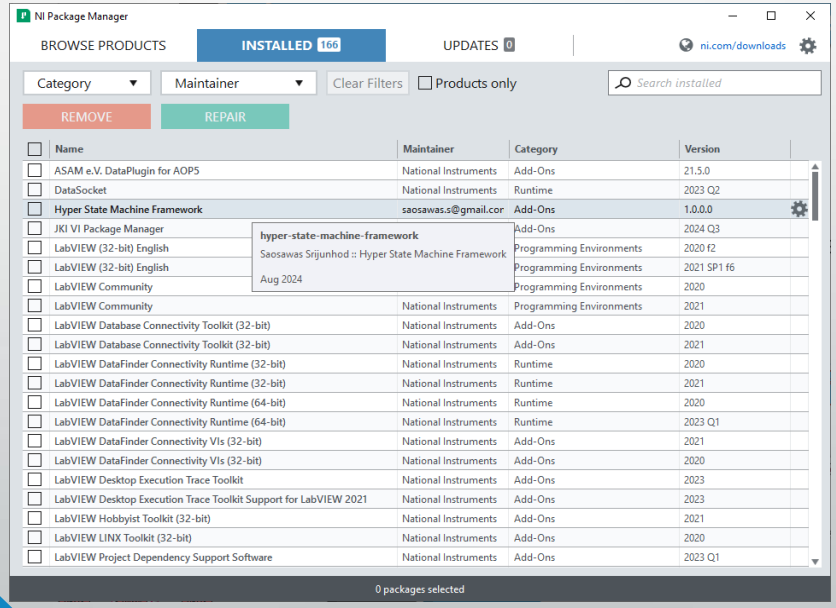
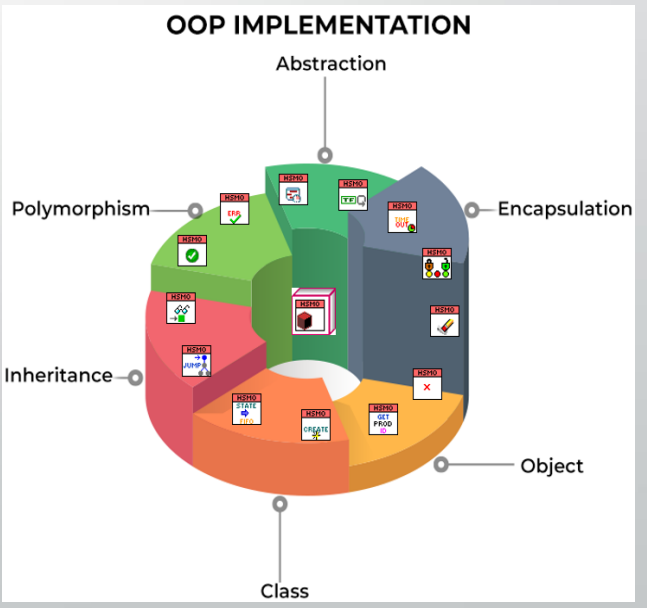
Hyper State Machine Object (HSMO)

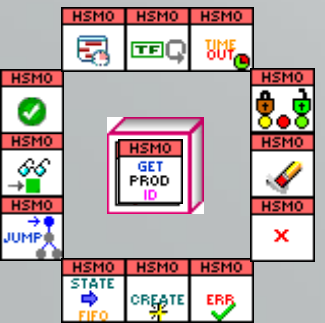
- Background
- Framework Installation
- Framework Remove
- Object Oriented Programming
- Shortcut Menu & Class Member
- Design Pattern
- Implementation
- Benefit





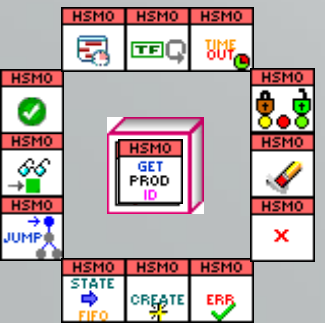






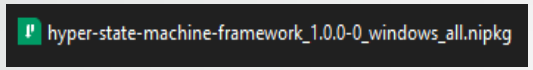
Hyper State Machine Object (HSMO)

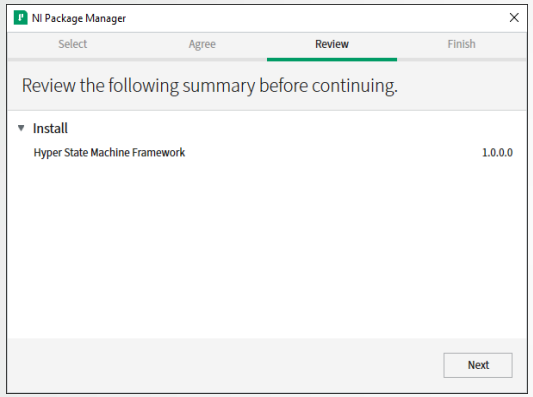
- Background
  - The complex hardware application have the mixed multiple transfer rate of data communication and multiple timing requirement base on hardware efficiency ?
  - The regular method of parallelize execution or multithreading is using multiple sub program (sub vi) to separate task for each device such as Clone, Actor framework, vi template, sub panel, FGV/Action Engine,etc
  - Yes, above method is the good one of programming technique but it's require more memory usage, more class member and more sub vi is very complicate to maintain on larger application.
  - This Framework we design with less memory usage and single space management by State Machine and handling with Case Structure base on OOP , which Base Class is the Precompile and Implement to all delivered class, each Base Class has independent Timing and State Mechanism
  - Replace the delay function with "State Interval" behavior, it's the key point of this framework, the multiple instance is using individual "State Interval" to keep each timing of hardware requirement and do not impact to each other State on parallel execution or multithreading

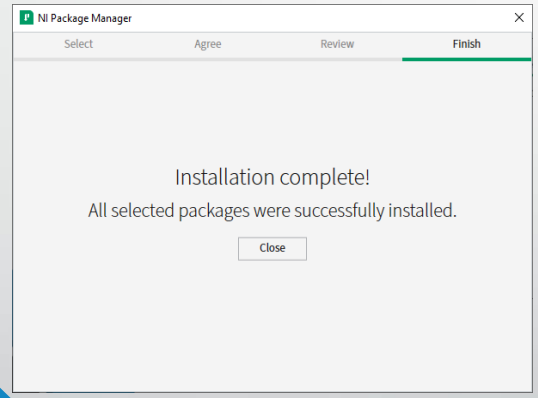


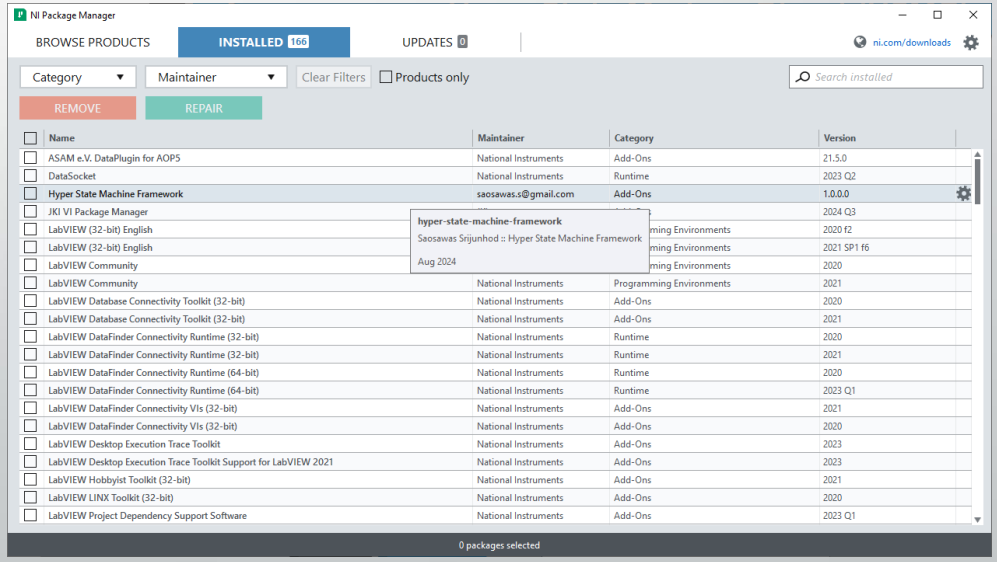
Hyper State Machine Object (HSMO)

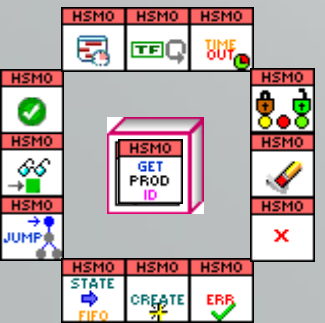
- Framework Installation





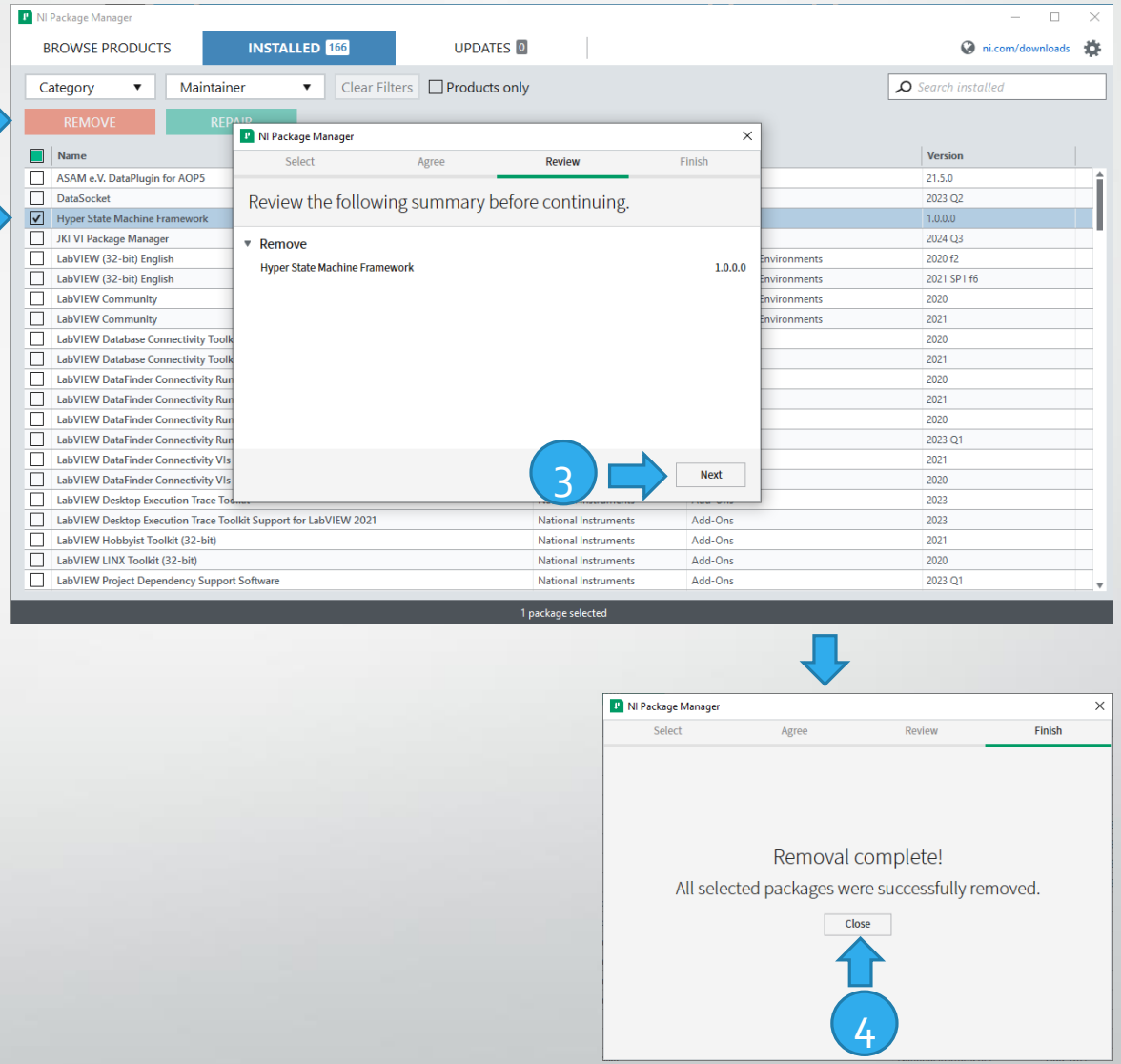


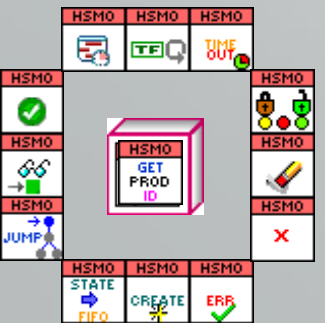




Hyper State Machine Object (HSMO)

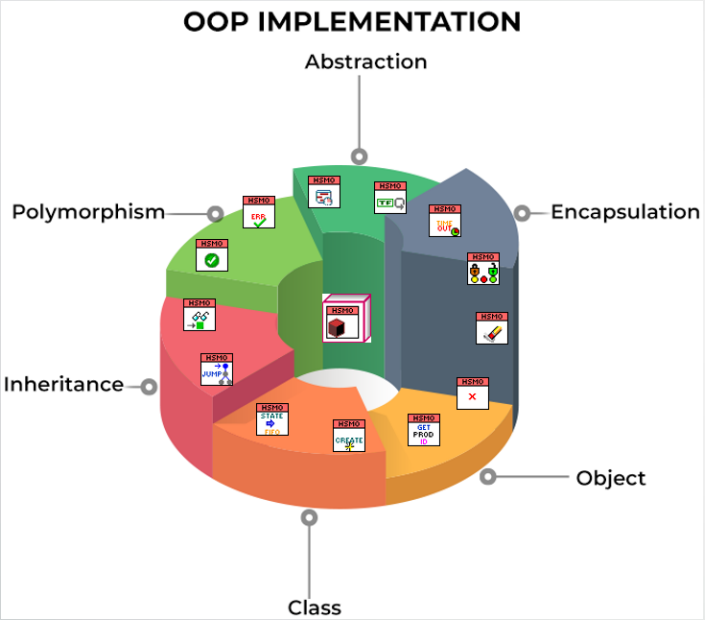
- Framework Remove





Hyper State Machine Object (HSMO)

- **Object Oriented Programming (OOP)**



- Abstraction :

  Data abstraction is a design pattern in which data are visible only to semantically related functions, so as to prevent misuse. The success of data abstraction leads to frequent incorporation of data hiding as a design principle in object oriented and pure functional programming.

- Inheritance :

  inheritance. This allows classes to be arranged in a hierarchy that represents "is-a-type-of" relationships. For example, class Employee might inherit from class Person. All the data and methods available to the parent class also appear in the child class with the same names. For example, class Person might define variables "first_name" and "last_name" with method "make_full_name()". These will also be available in class Employee, which might add the variables "position" and "salary". This technique allows easy re-use of the same procedures and data definitions, in addition to potentially mirroring real-world relationships in an intuitive way. Rather than utilizing database tables and programming subroutines, the developer utilizes objects the user may be more familiar with: objects from their application domain.

- Encapsulation :

  Encapsulation prevents external code from being concerned with the internal workings of an object. This facilitates code refactoring, for example allowing the author of the class to change how objects of that class represent their data internally without changing any external code (as long as "public" method calls work the same way). It also encourages programmers to put all the code that is concerned with a certain set of data in the same class, which organizes it for easy comprehension by other programmers. Encapsulation is a technique that encourages decoupling.

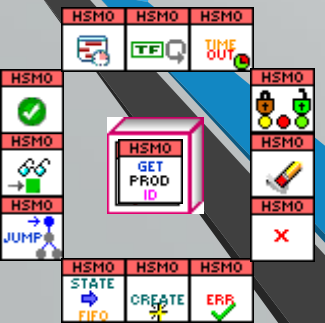
- Polymorphism :

  polymorphism – is when calling code can be independent of which class in the supported hierarchy it is operating on – the parent class or one of its descendants. Meanwhile, the same operation name among objects in an inheritance hierarchy may behave differently.

  For example, objects of type Circle and Square are derived from a common class called Shape. The Draw function for each type of Shape implements what is necessary to draw itself while calling code can remain indifferent to the particular type of Shape being drawn.

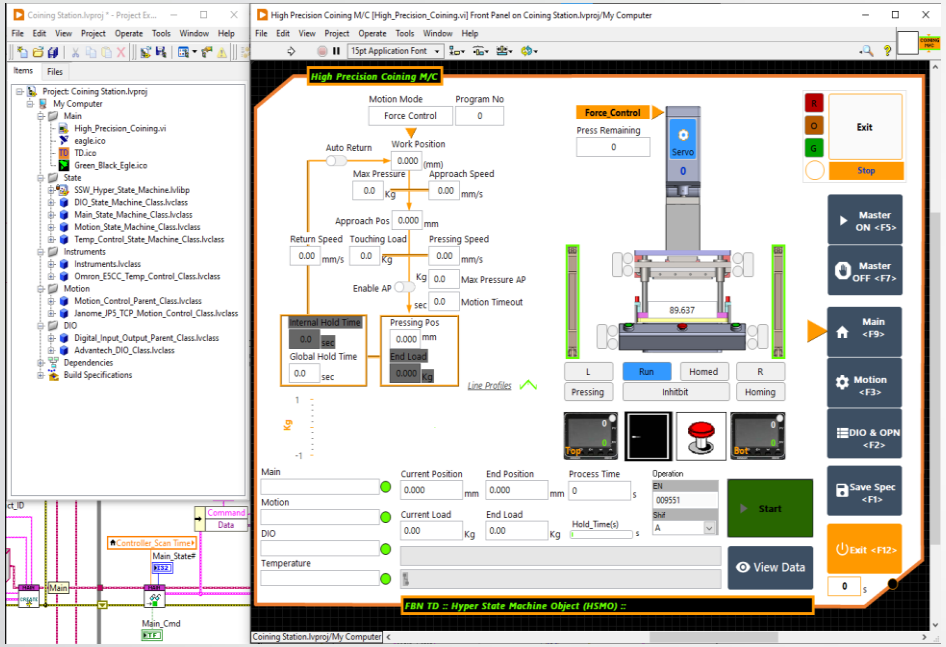
  This is another type of abstraction that simplifies code external to the class hierarchy and enables strong separation of concerns.

- *Object is an Instant of Class*
- *In object-oriented programming, a class is a template definition of the methods and variables in a particular kind of object. Thus, an object is a specific instance of a class; it contains real values instead of variables. The class is one of the defining ideas of object-oriented programming.*

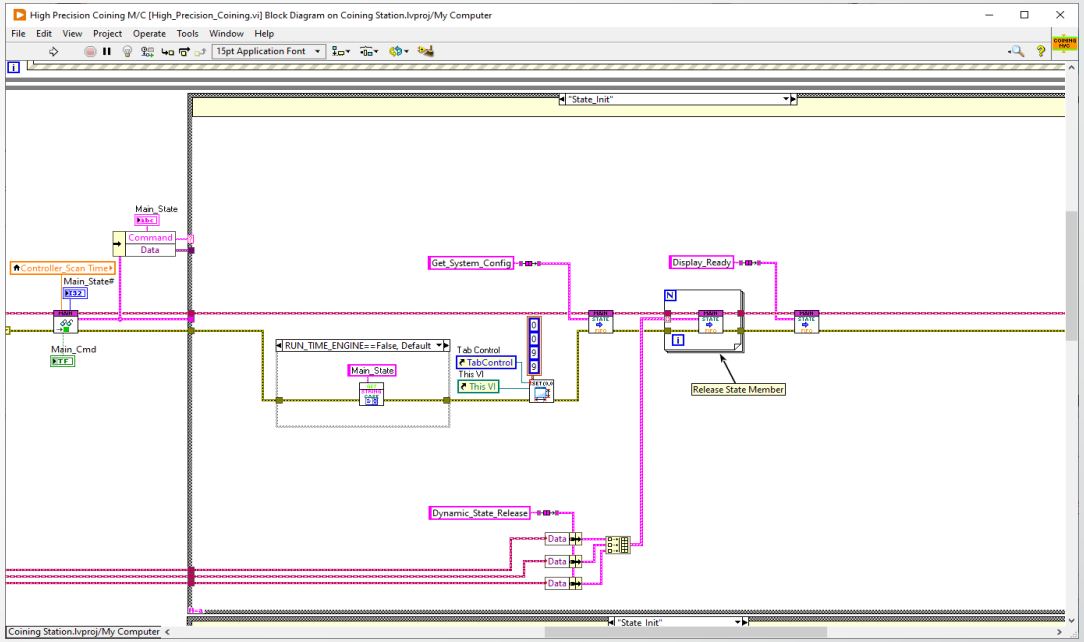


Hyper State Machine Object (HSMO)

Project

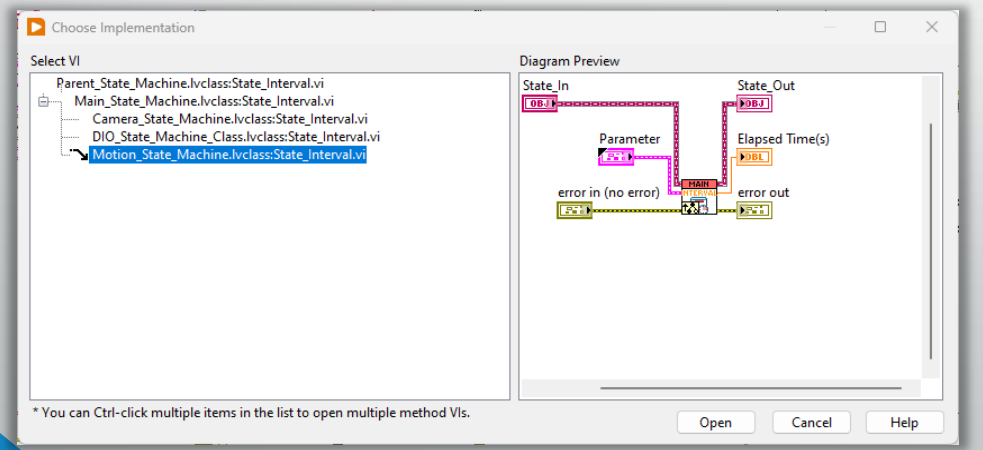


Coding

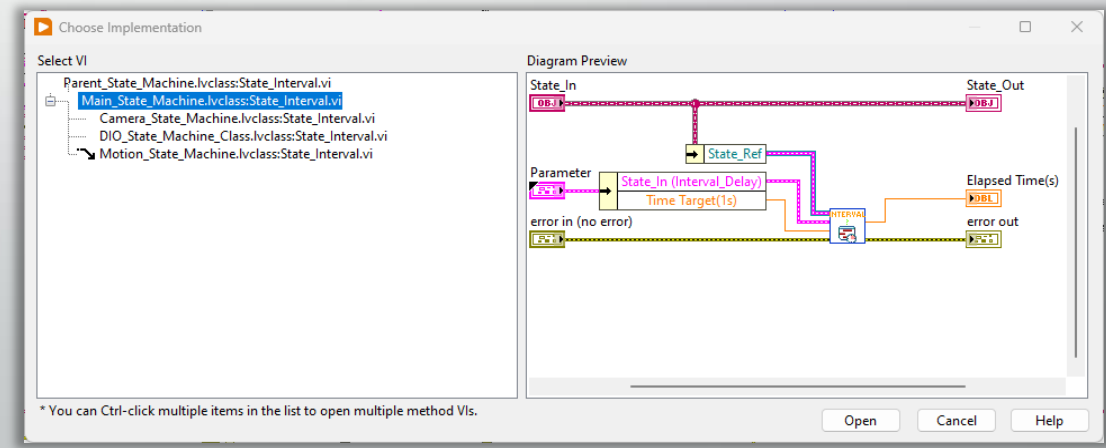


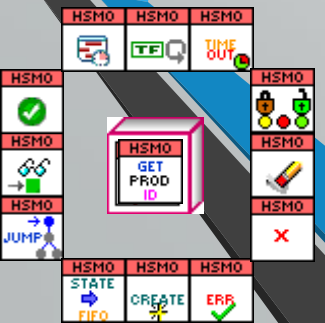
Implementation)

Dynamic Dispatching



Method Overriding



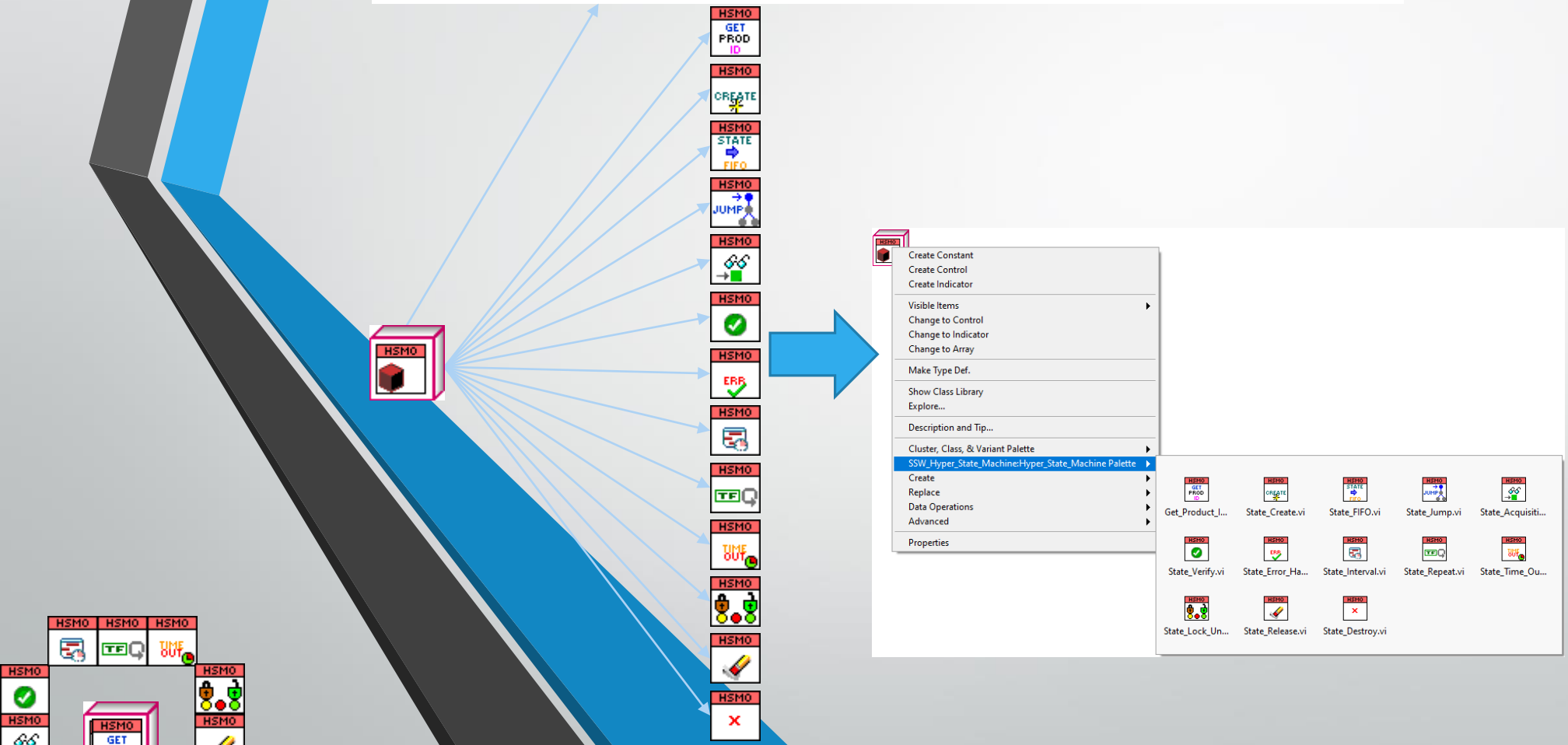


Hyper State Machine Object (HSMO)

SSW_Hyper_State_Machine.lvlibp:Hyper_State_Machine.lvclass

No description available.

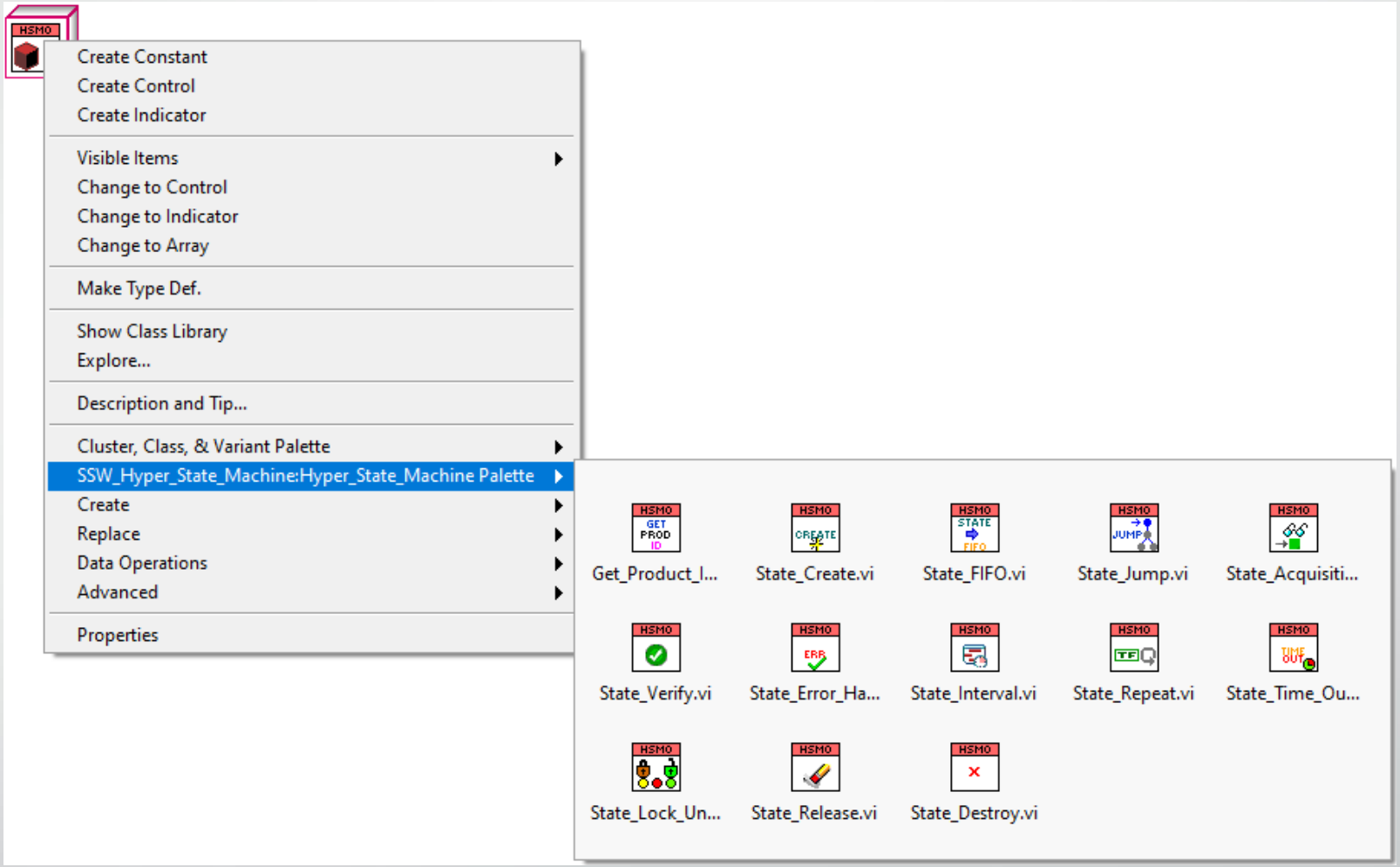
OBJ SSW_Hyper_State_Machine.lvlibp:Hyper_State_Machine.lvclass (SSW_Hyper_State_Machine.lvlibp:Hyper_State_Machine)

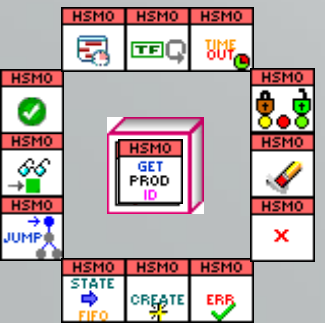


- Shortcut Menu
- Class Member (12)
  - Get_Product_ID
  - State_Create
  - State_FIFO
  - State_Jump
  - State_Acquisition
  - State_Error_Handling
  - State_Interval
  - State_Repeat
  - State_Time_Out
  - State_Lock_Unlock
  - State_Verify
  - State_Release
  - State_Destroy

Hyper State Machine Object (HSMO)

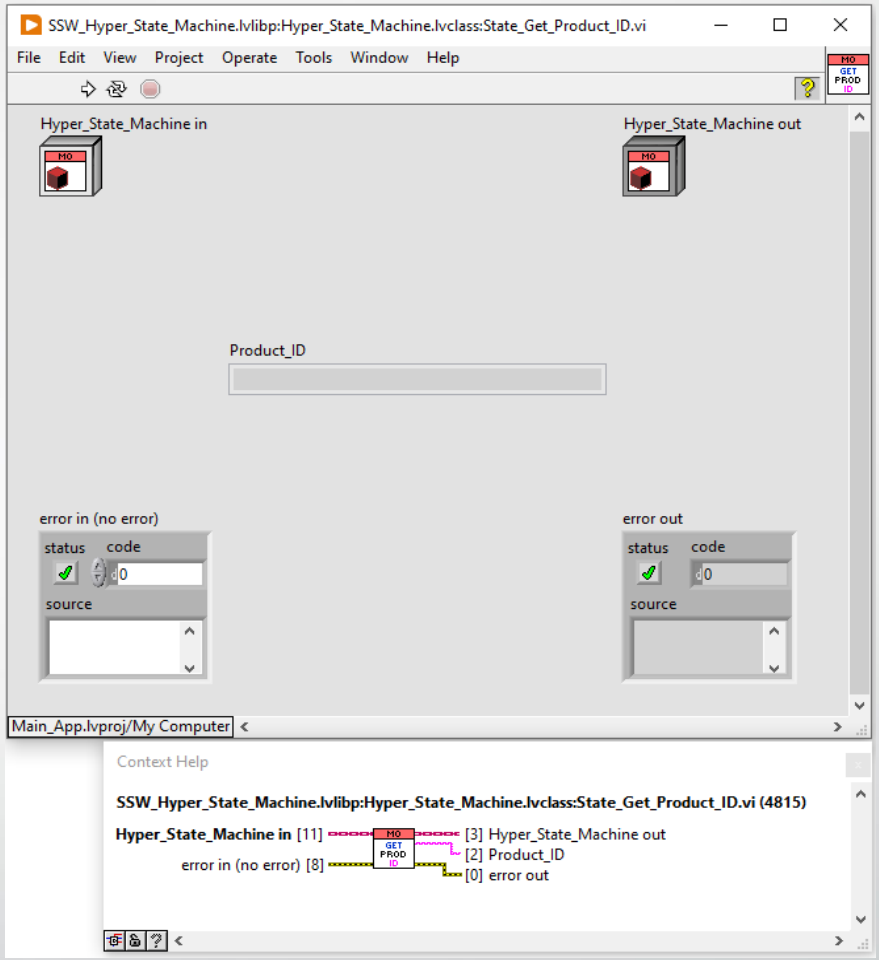
- Shortcut Menu





Hyper State Machine Object (HSMO)

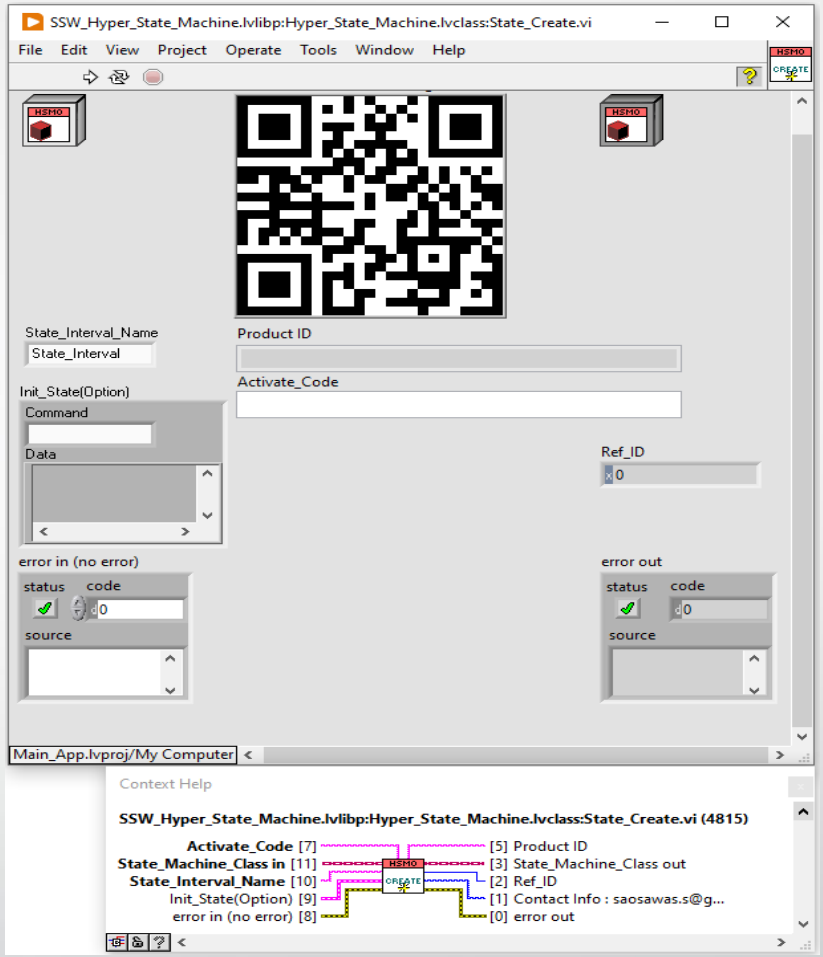
- Class Member (12)
  - Get_Product_ID
  - State_Create
  - State_FIFO
  - State_Jump
  - State_Acquisition
  - State_Error_Handling
  - State_Interval
  - State_Repeat
  - State_Time_Out
  - State_Lock_Unlock
  - State_Verify
  - State_Release
  - State_Destroy



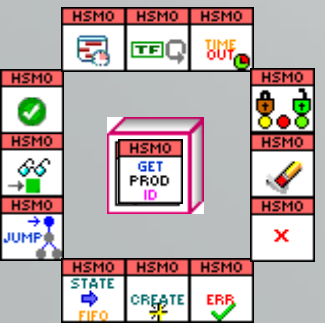
- Method :
  - Get HSMO Product ID for License Activate in State_Create Method

Hyper State Machine Object (HSMO)

- Class Member (12)
  - Get_Product_ID
  - State_Create
  - State_FIFO
  - State_Jump
  - State_Acquisition
  - State_Error_Handling
  - State_Interval
  - State_Repeat
  - State_Time_Out
  - State_Lock_Unlock
  - State_Verify
  - State_Release
  - State_Destroy

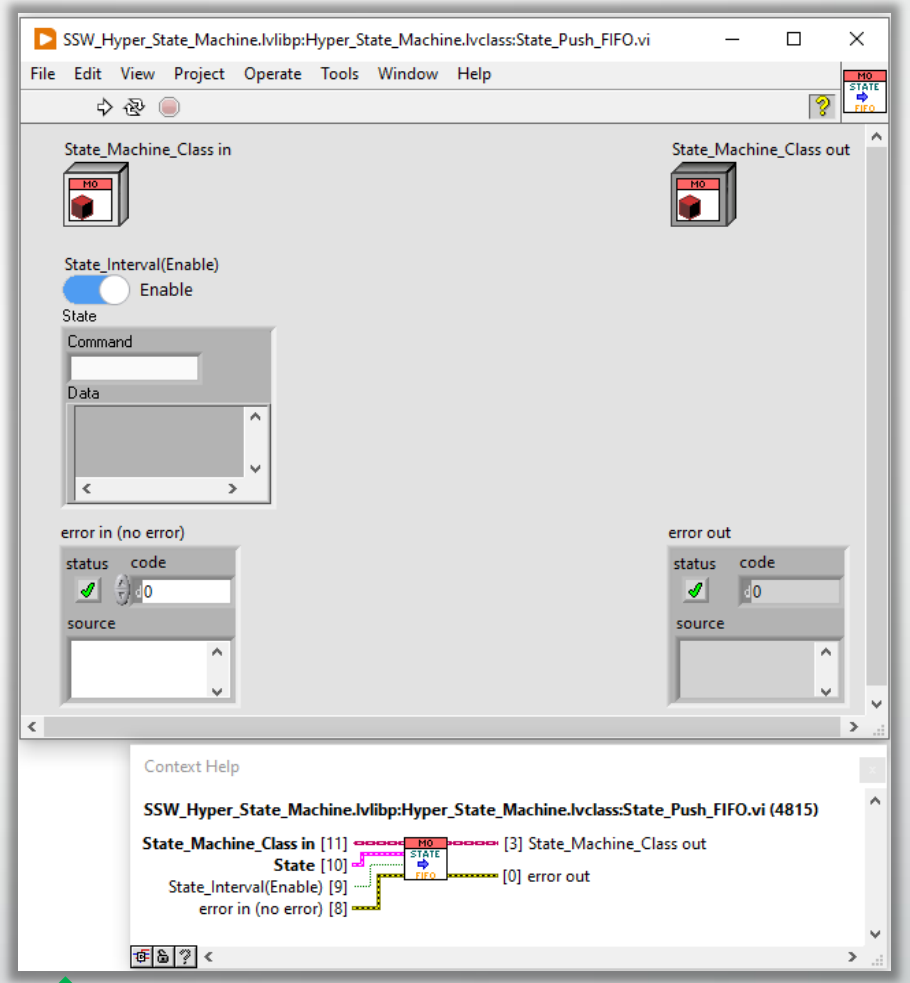


- Method :
  - Create HSMO

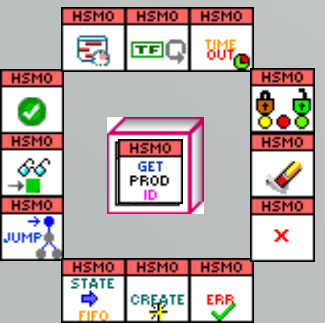


Hyper State Machine Object (HSMO)

- Class Member (12)
  - Get_Product_ID
  - State_Create
  - State_FIFO
  - State_Jump
  - State_Acquisition
  - State_Error_Handling
  - State_Interval
  - State_Repeat
  - State_Time_Out
  - State_Lock_Unlock
  - State_Verify
  - State_Release
  - State_Destroy

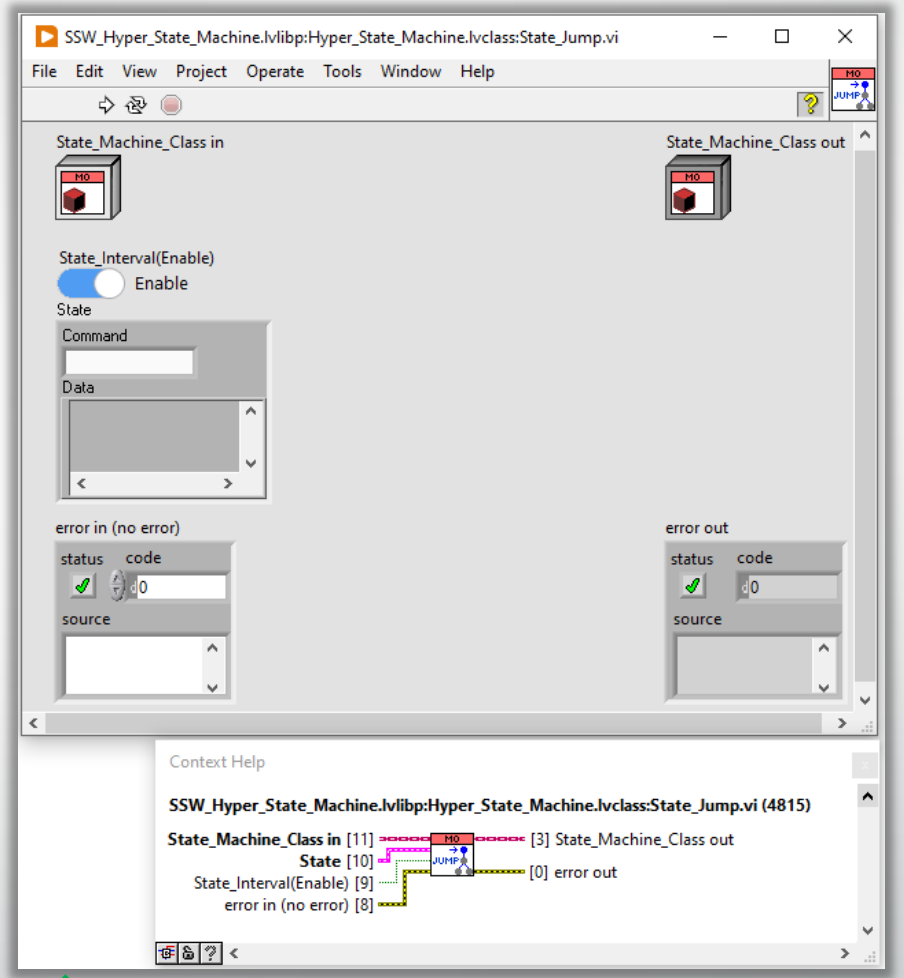


- Method :
  - Insert HSMO as FIFO (First In First Out)

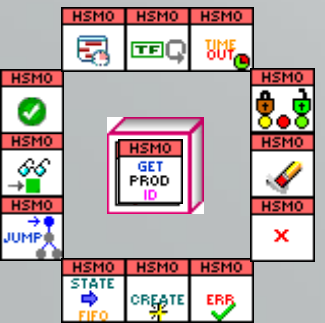


Hyper State Machine Object (HSMO)

- Class Member (12)
  - Get_Product_ID
  - State_Create
  - State_FIFO
  - **State_Jump**
  - State_Acquisition
  - State_Error_Handling
  - State_Interval
  - State_Repeat
  - State_Time_Out
  - State_Lock_Unlock
  - State_Verify
  - State_Release
  - State_Destroy

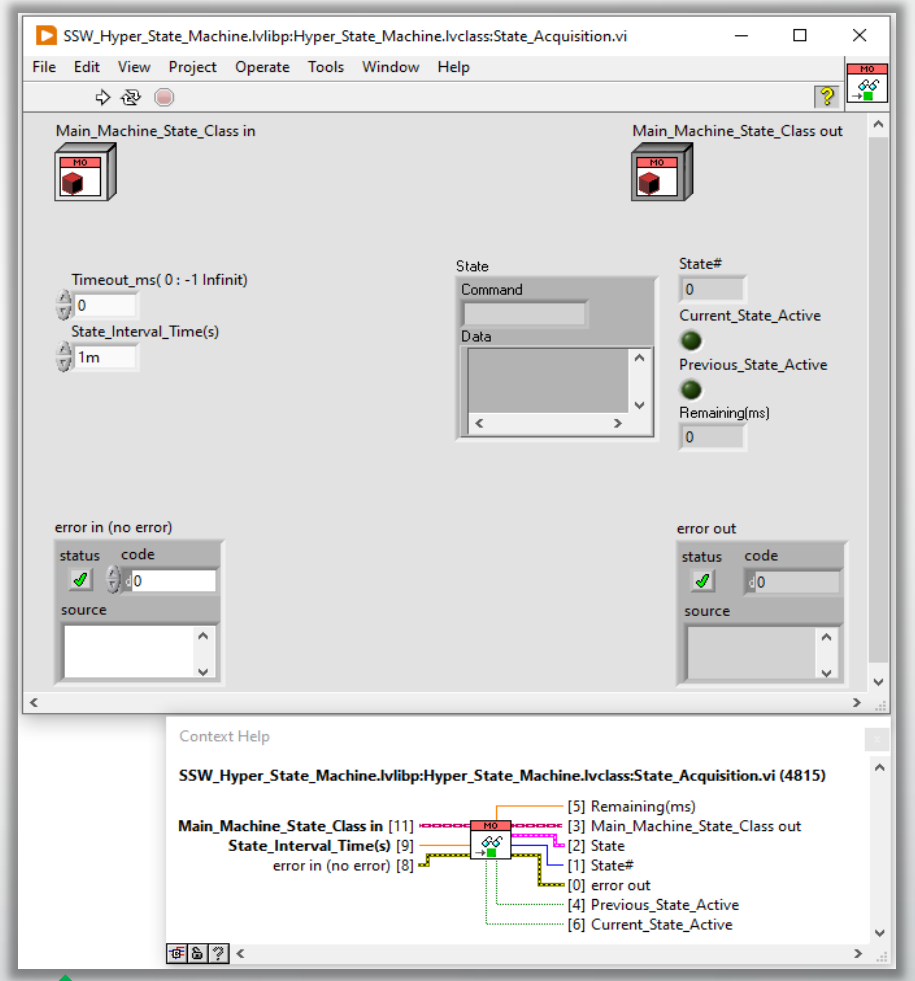


- Method :
  - Insert HSMO as Front most

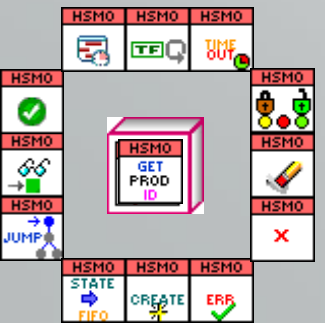


Hyper State Machine Object (HSMO)

- Class Member (12)
  - Get_Product_ID
  - State_Create
  - State_FIFO
  - State_Jump
  - State_Acquisition
  - State_Error_Handling
  - State_Interval
  - State_Repeat
  - State_Time_Out
  - State_Lock_Unlock
  - State_Verify
  - State_Release
  - State_Destroy

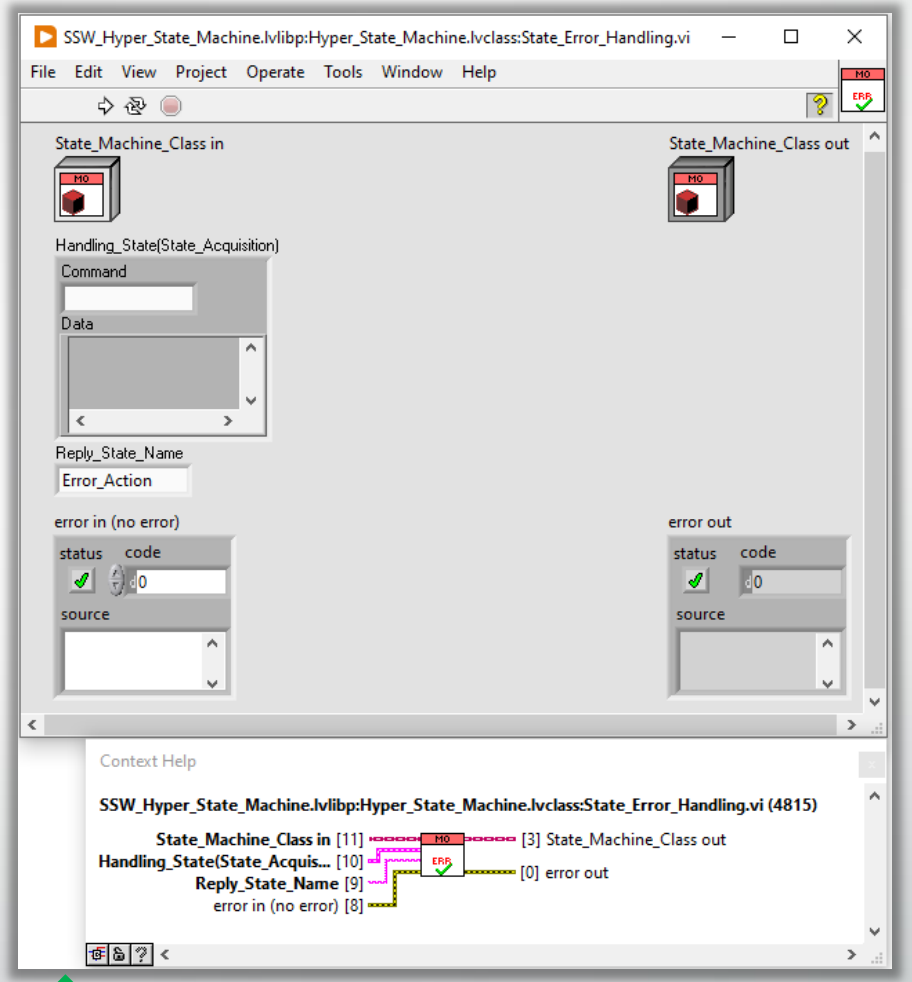


- Method :
  - Get Current HSMO and Action Handling

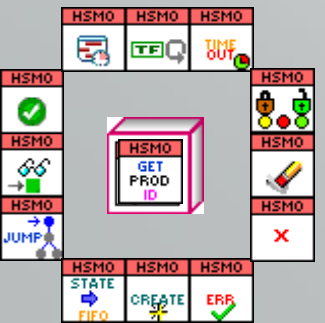


Hyper State Machine Object (HSMO)

- Class Member (12)
  - Get_Product_ID
  - State_Create
  - State_FIFO
  - State_Jump
  - State_Acquisition
  - State_Error_Handling
  - State_Interval
  - State_Repeat
  - State_Time_Out
  - State_Lock_Unlock
  - State_Verify
  - State_Release
  - State_Destroy

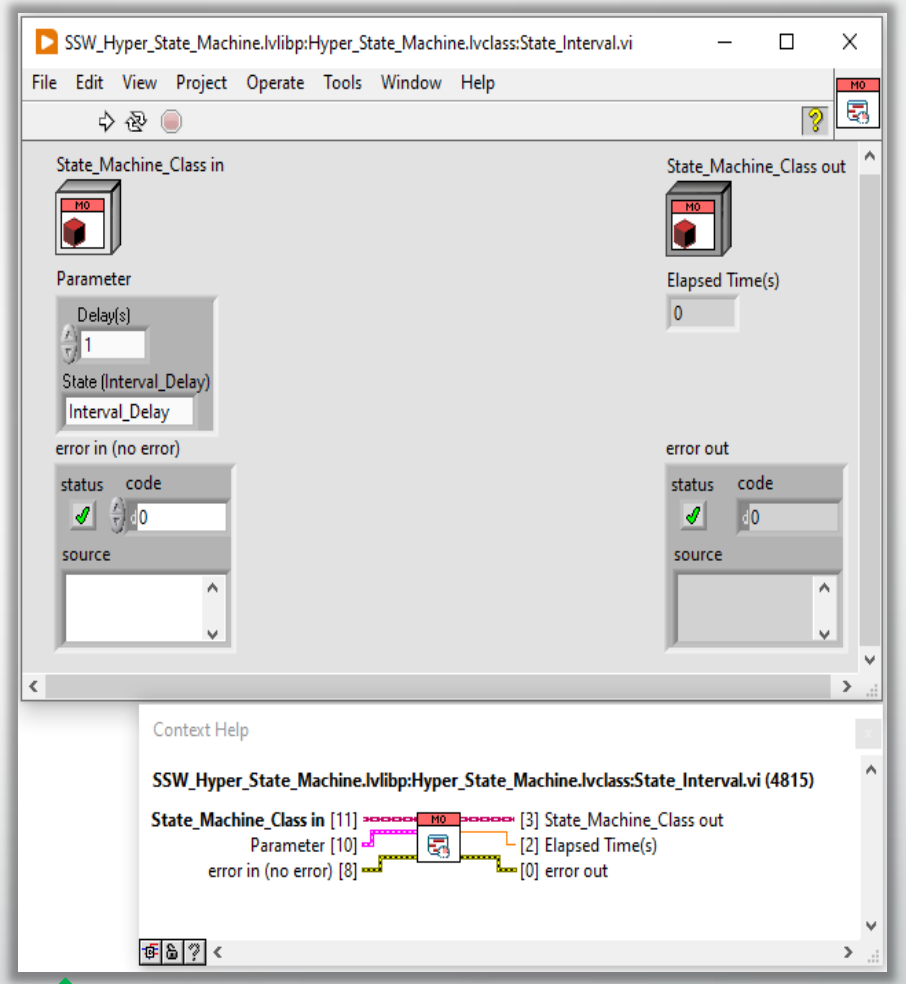


- Method :
  - Error Handling of HSMO Member Class and Un define Method Detection

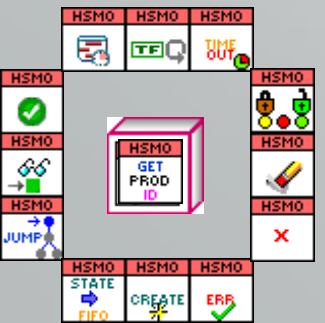


Hyper State Machine Object (HSMO)

- Class Member (12)
  - Get_Product_ID
  - State_Create
  - State_FIFO
  - State_Jump
  - State_Acquisition
  - State_Error_Handling
  - State_Interval
  - State_Repeat
  - State_Time_Out
  - State_Lock_Unlock
  - State_Verify
  - State_Release
  - State_Destroy

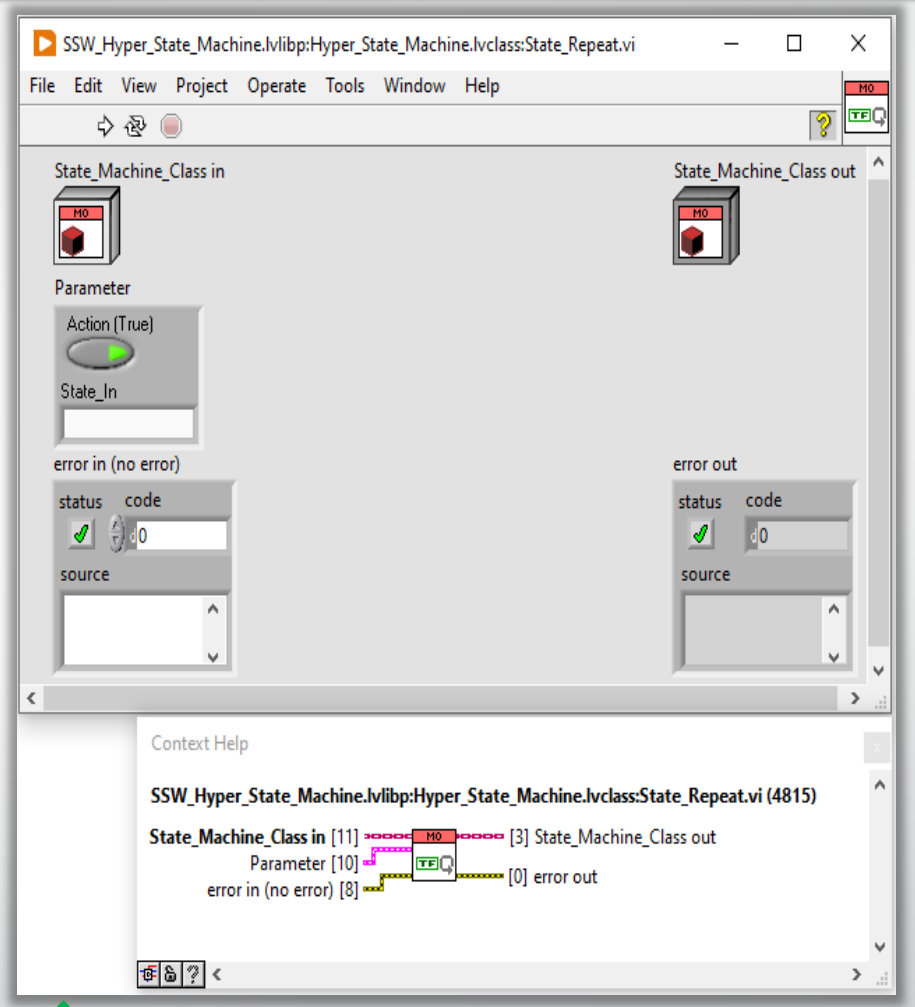


- Method :
  - Repeat HSMO Target State with Specify Timing Target

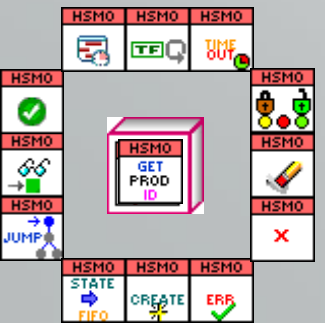


Hyper State Machine Object (HSMO)

- Class Member (12)
  - Get_Product_ID
  - State_Create
  - State_FIFO
  - State_Jump
  - State_Acquisition
  - State_Error_Handling
  - State_Interval
  - State_Repeat
  - State_Time_Out
  - State_Lock_Unlock
  - State_Verify
  - State_Release
  - State_Destroy

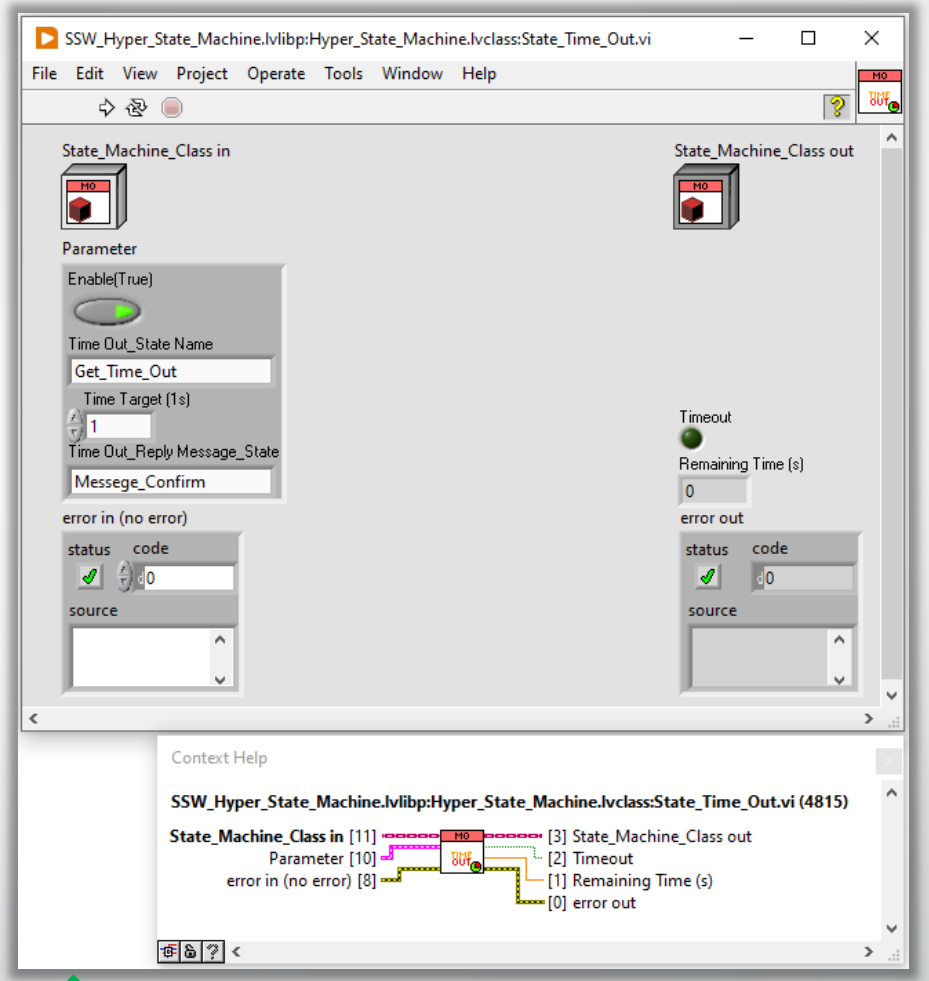


- Method :
  - Repeat HSMO Target State with Condition of Flag (True/False)



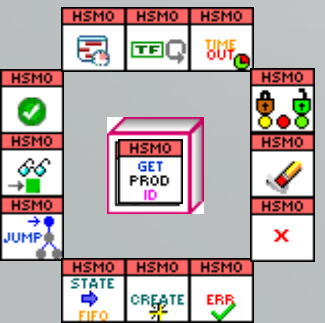
Hyper State Machine Object (HSMO)

- Class Member (12)
  - Get_Product_ID
  - State_Create
  - State_FIFO
  - State_Jump
  - State_Acquisition
  - State_Error_Handling
  - State_Interval
  - State_Repeat
  - **State_Time_Out**
  - State_Lock_Unlock
  - State_Verify
  - State_Release
  - State_Destroy



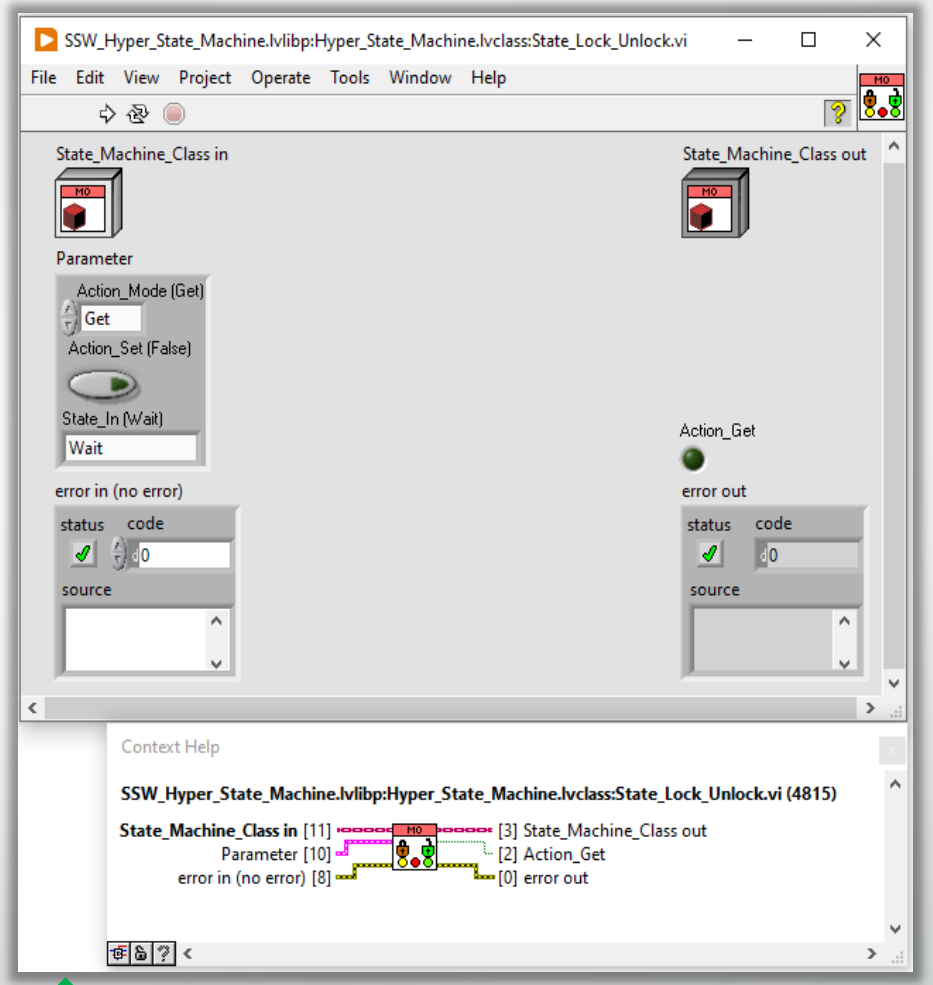
- Method :

- Repeat & Monitoring HSMO Target State with Timeout Target and Condition of Flag (True/False) , if Target State is Timeout Active the Destination Action is activate



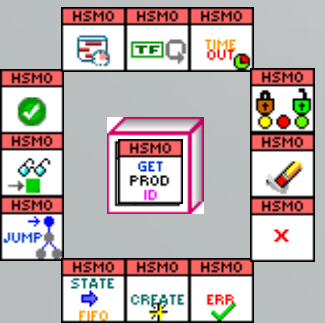
Hyper State Machine Object (HSMO)

- Class Member (12)
  - Get_Product_ID
  - State_Create
  - State_FIFO
  - State_Jump
  - State_Acquisition
  - State_Error_Handling
  - State_Interval
  - State_Repeat
  - State_Time_Out
  - **State_Lock_Unlock**
  - State_Verify
  - State_Release
  - State_Destroy



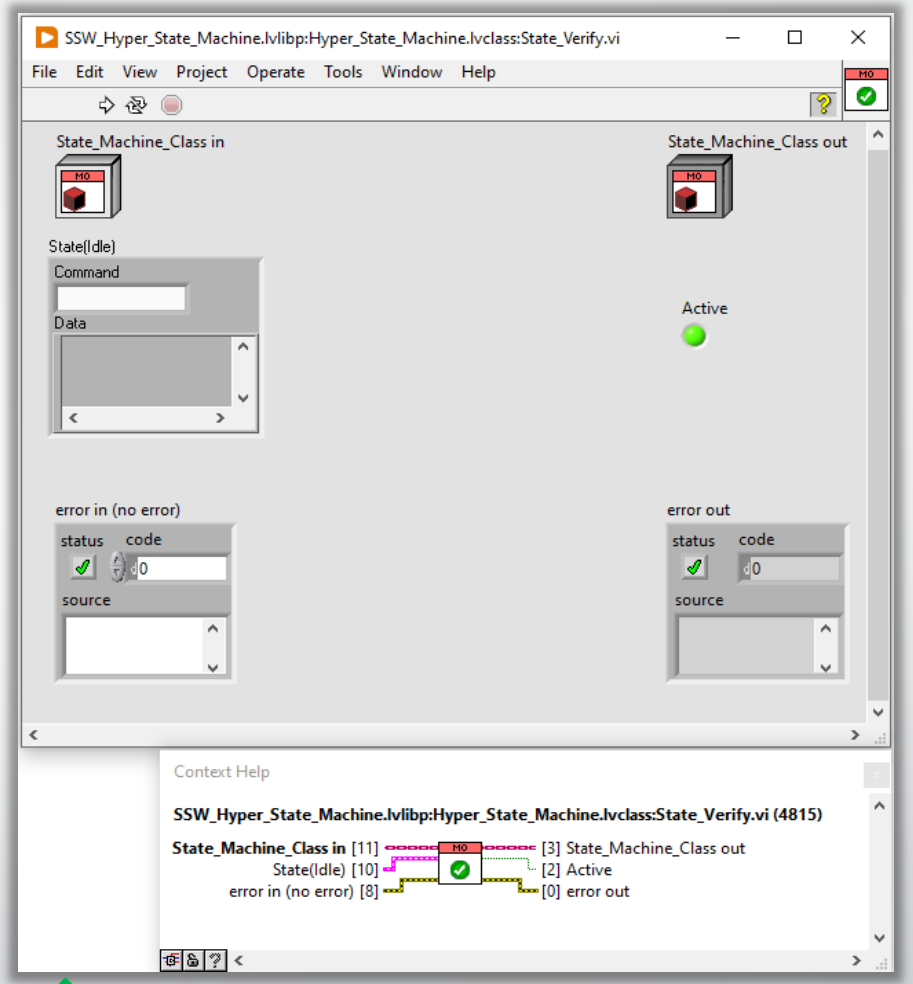
- Method :

- Lock or Unlock HSMO Target State, If Current State is Locking it's Waiting until the Unlock Command active

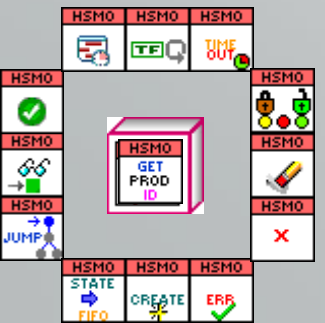


Hyper State Machine Object (HSMO)

- Class Member (12)
  - Get_Product_ID
  - State_Create
  - State_FIFO
  - State_Jump
  - State_Acquisition
  - State_Error_Handling
  - State_Interval
  - State_Repeat
  - State_Time_Out
  - State_Lock_Unlock
  - State_Verify
  - State_Release
  - State_Destroy

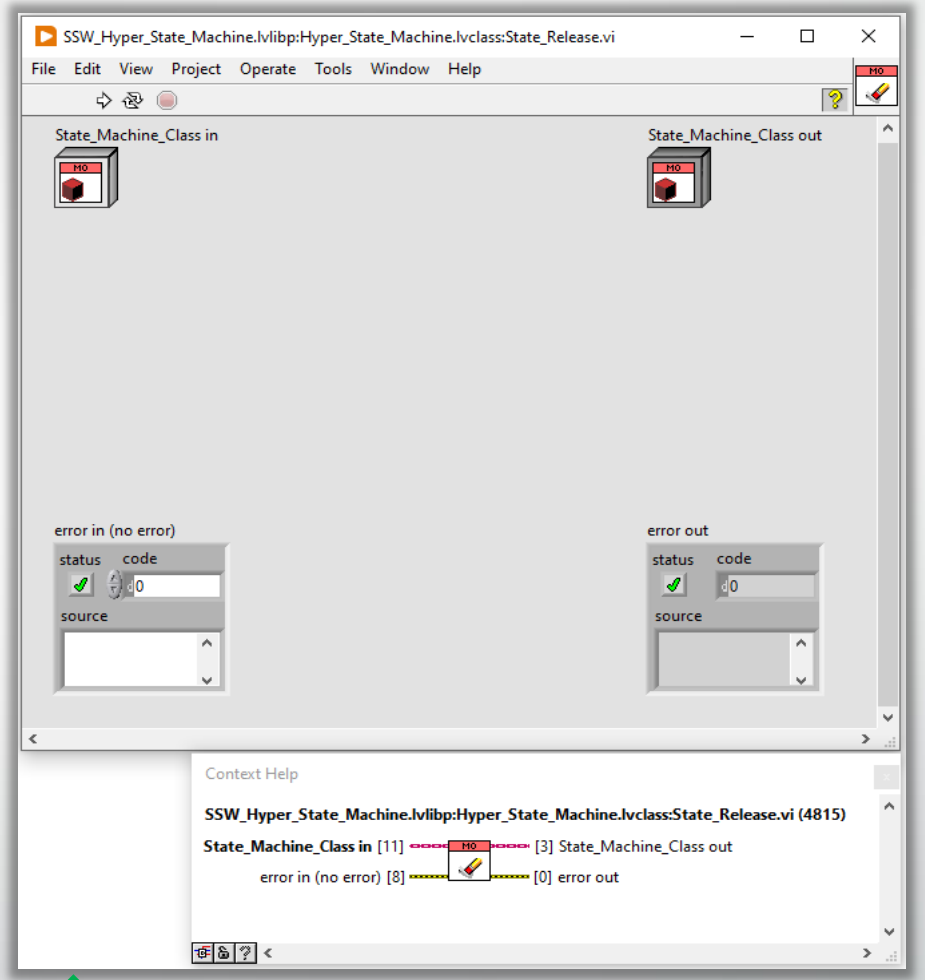


- Method :
  - HSMO Self detection of un define State or Action to prevent malfunction

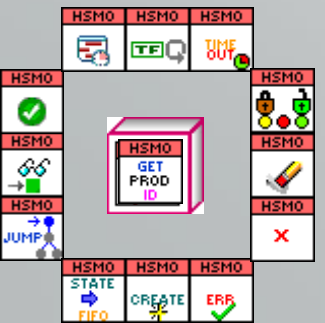


Hyper State Machine Object (HSMO)

- Class Member (12)
  - Get_Product_ID
  - State_Create
  - State_FIFO
  - State_Jump
  - State_Acquisition
  - State_Error_Handling
  - State_Interval
  - State_Repeat
  - State_Time_Out
  - State_Lock_Unlock
  - State_Verify
  - State_Release
  - State_Destroy

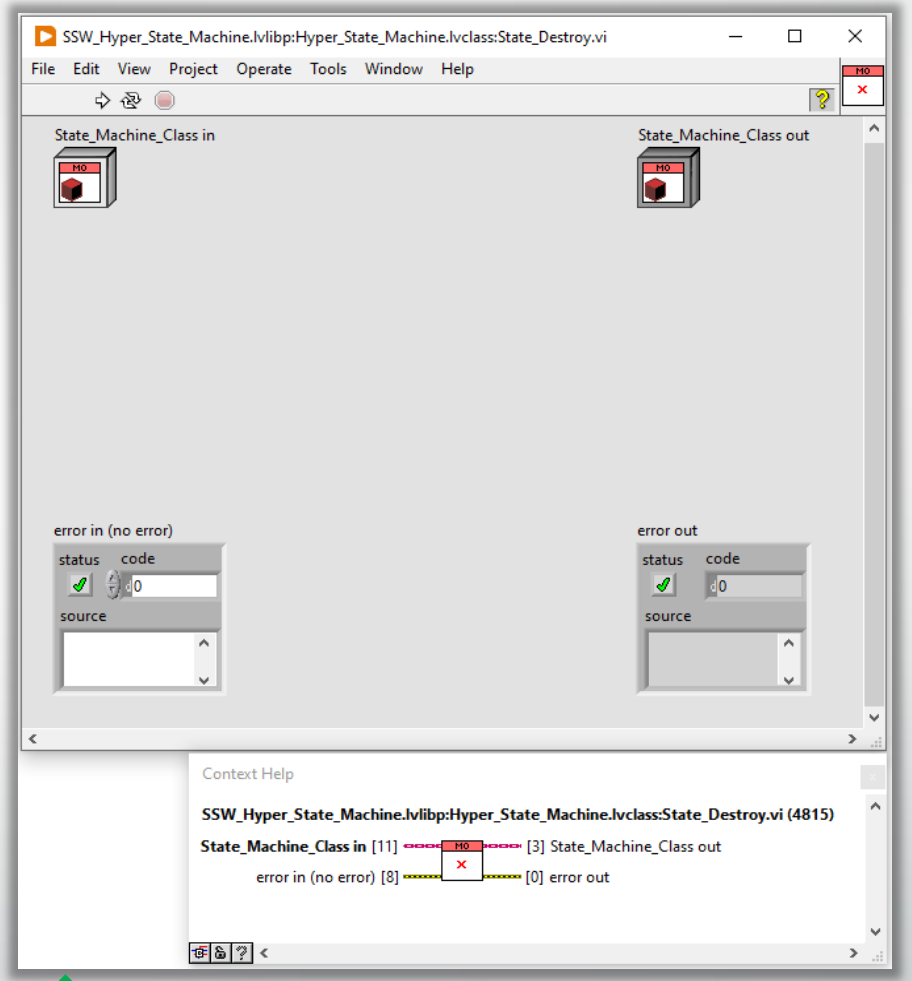


- Method :
  - HSMO Release or Buffer Flushing

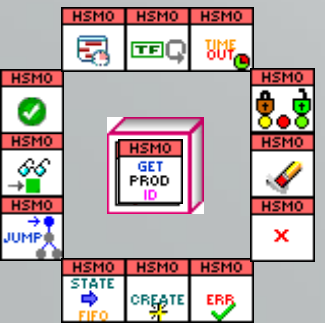


Hyper State Machine Object (HSMO)

- Class Member (12)
  - Get_Product_ID
  - State_Create
  - State_FIFO
  - State_Jump
  - State_Acquisition
  - State_Error_Handling
  - State_Interval
  - State_Repeat
  - State_Time_Out
  - State_Lock_Unlock
  - State_Verify
  - State_Release
  - State_Destroy

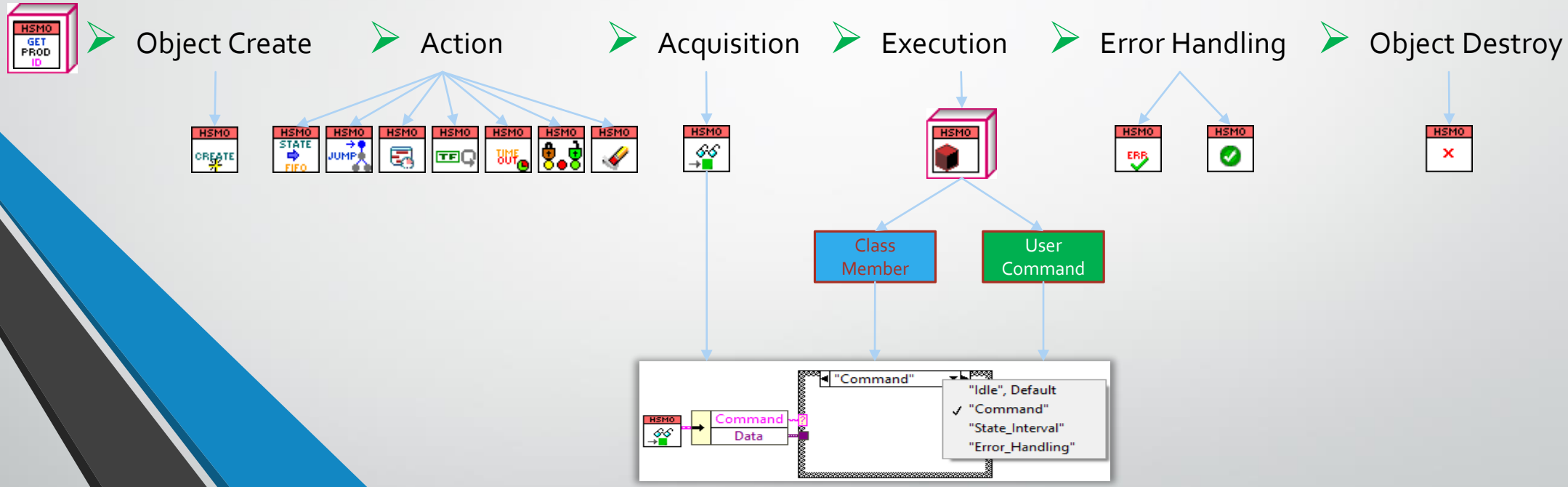


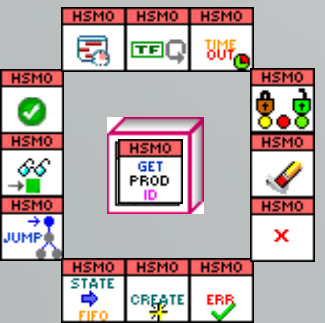
- Method :
  - Delete HSMO



Hyper State Machine Object (HSMO)

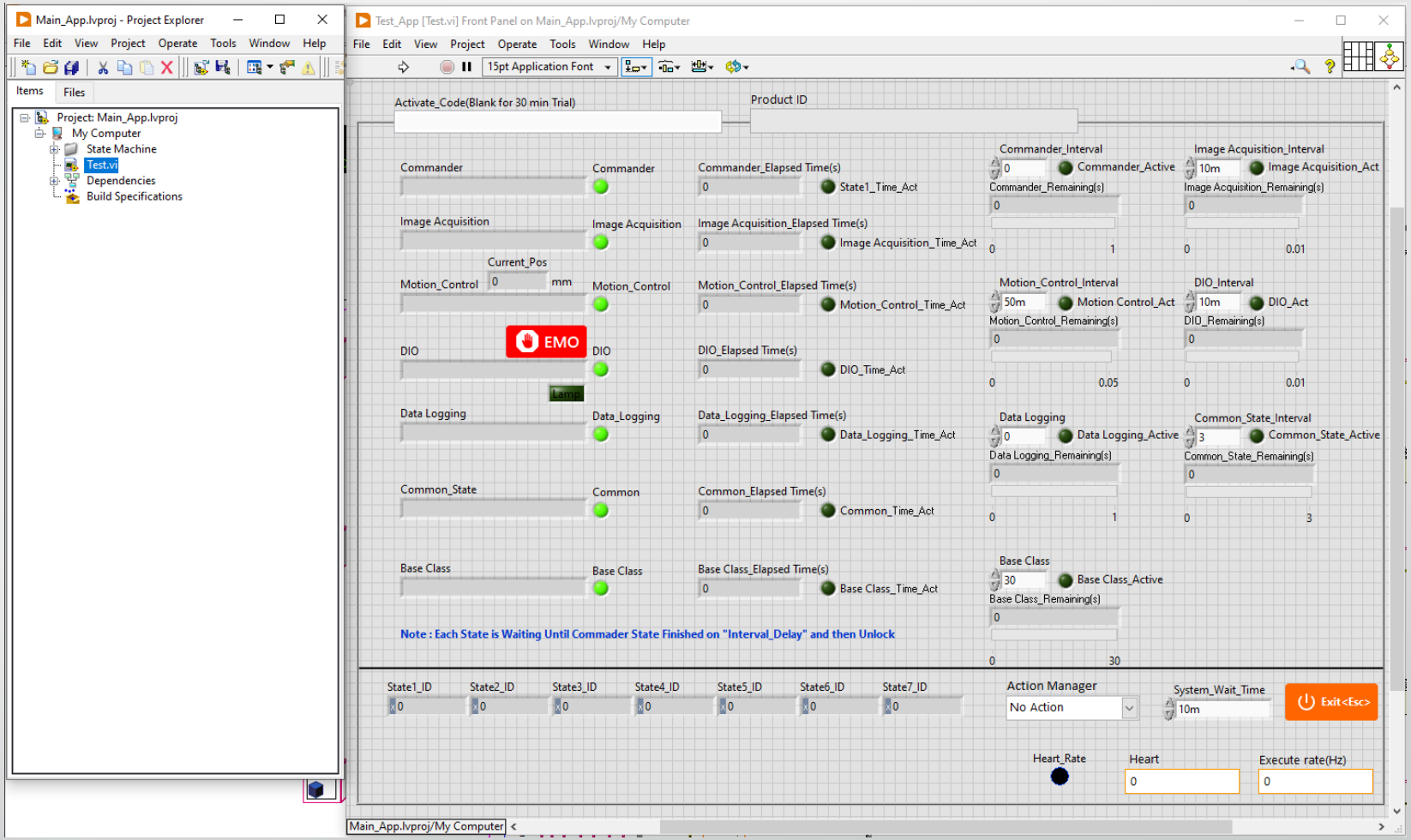
- Design Pattern

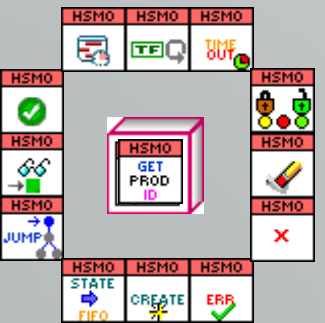




Hyper State Machine Object (HSMO)

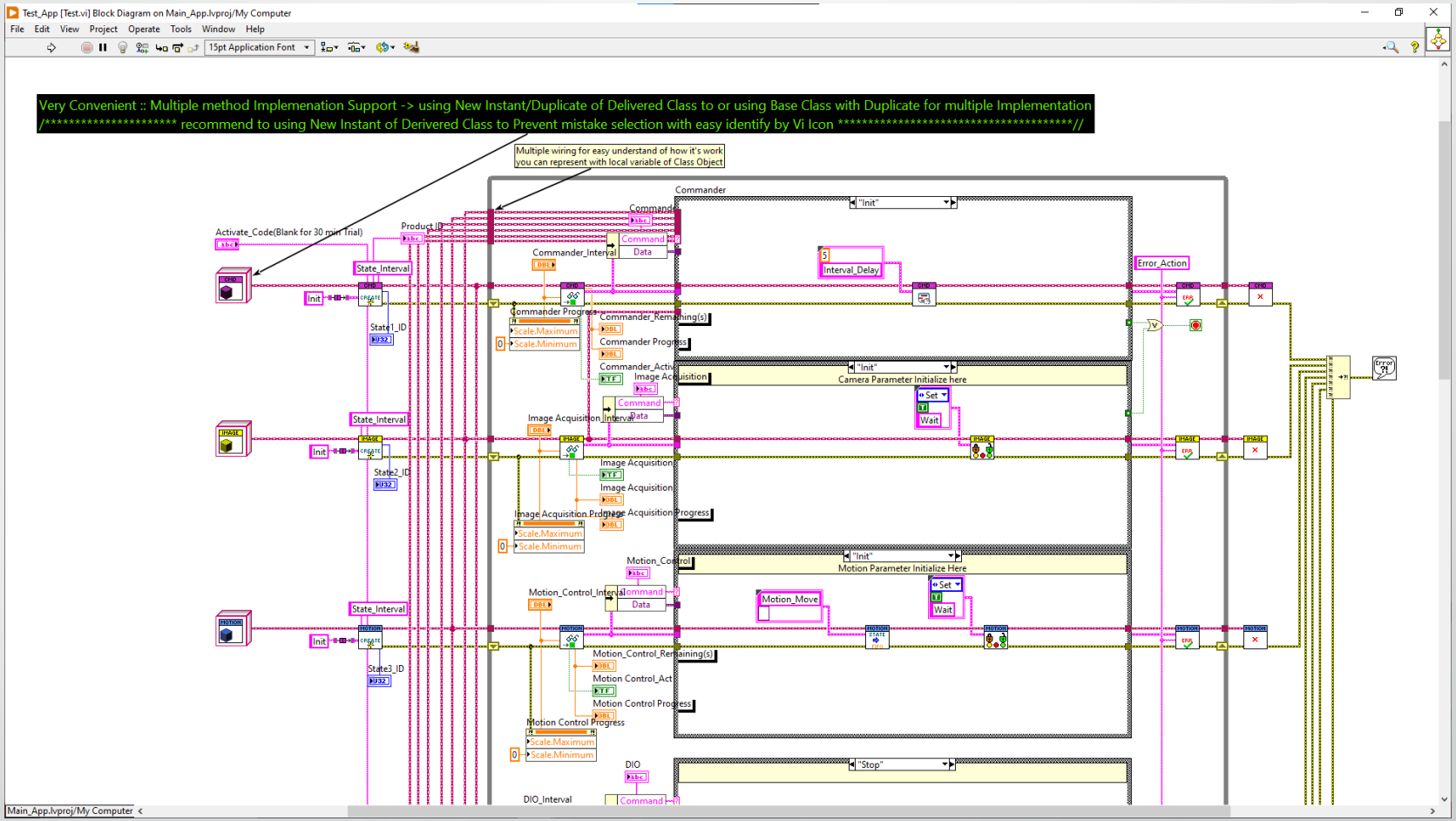
- Implementation

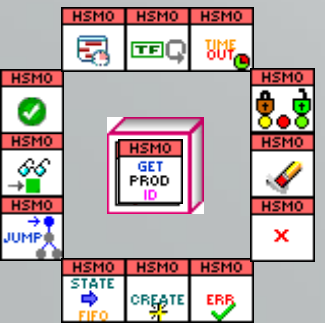




Hyper State Machine Object (HSMO)

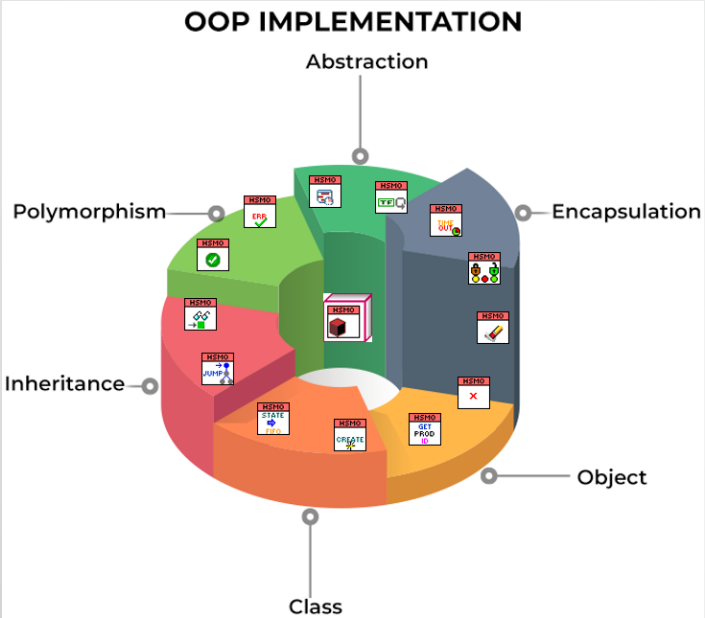
- Implementation(Continue)



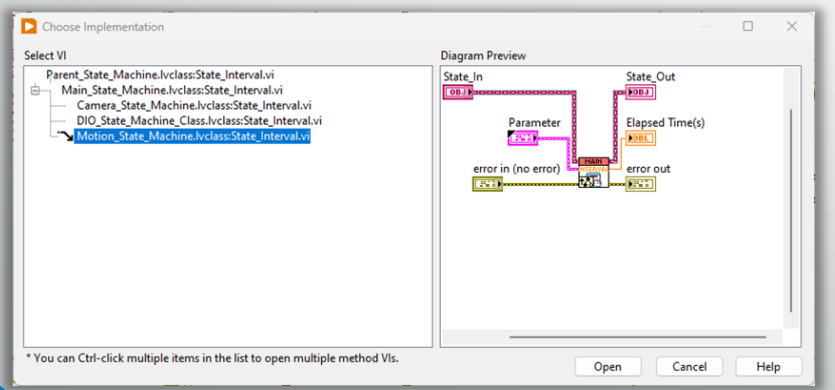


Hyper State Machine Object (HSMO)

Benefit



Dynamic Dispatching Implementation



Reduce Develop Time :

Code reusability, an important feature of Object-Oriented Programming (OOP), is enabled through inheritance, polymorphism, and information hiding. With inheritance, an object can be extended and code from the parent object can be reused or overloaded in the child object. For example we use one (Parent Class) and after that using Inheritance behavior form Parent Class to many Delivered Class and every property and Method of Parent Class is automatic transfer to Delivered class

Note : The Traditional Software Programming using Duplicate Code when we need more function and this is take time to debug or hard to maintain coding.

Automatic Method Handling :

Dynamic Dispatching : Abstract parent class is automatic handle of many delivered class, with this behavior we able to Simulation Software without actual hardware by using Abstract Parent Class and switch to Actual hardware when ready (Hardware Abstraction)

Low Resource and Fast Execution :

- This library is the Precompile code , it's low memory usage require and fast execution

Multi Execution Rate Support :

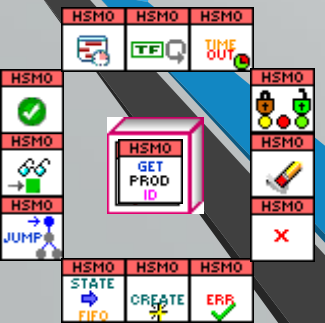
- All state task has the individual clock and mechanism managements, so we can mixed of low transfer rate and hi transfer rate device together.

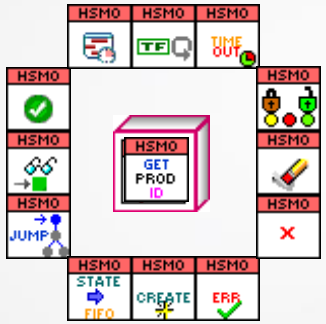
Asset Protection :

- This Framework is protected with standard license activate method , user can applied to protect own asset when machine or system is deliver to customer

Large Scale Application Support :

OOP it's powerful and flexible of Programming Corroborate , Team Member is able to work with same Parent Class or Using Override Method to Make new Method that Inheritance from Parent Class with is not impact to other class and if Parent Class is has new method the Delivered Class is has new method also





Thank you