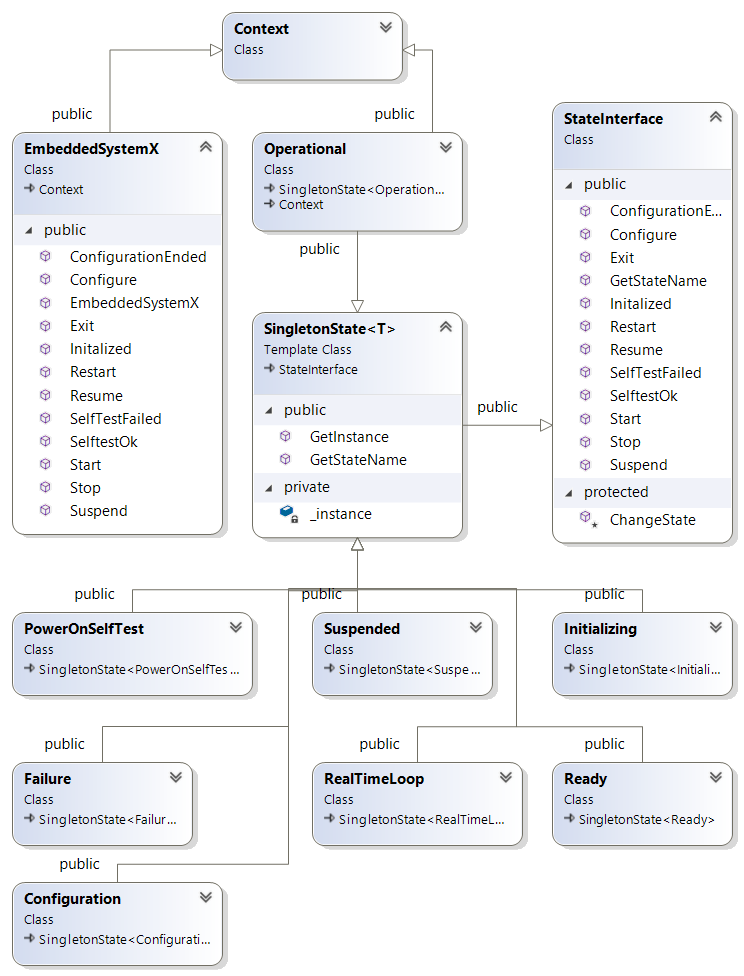
# Solution design for implementing GoF state pattern

Following is the class diagram for a solution implementing GoF state pattern. In this design shared logic of the states is encapsulated in the **SingletonState** and **StateInterface** classes. Moreover, **Operational** class is both a state and a **Context** because it is a super state that can be in different states.



# Implementation in C++

Implementation was done based on the design presented in the previous section. Following are snippets of more important code.

Figure 1 shows state interface. State interface provides a method for each possible event in the system with a default empty implementation. Classes implementing the interface can optionally provide new implementation handling specific events.

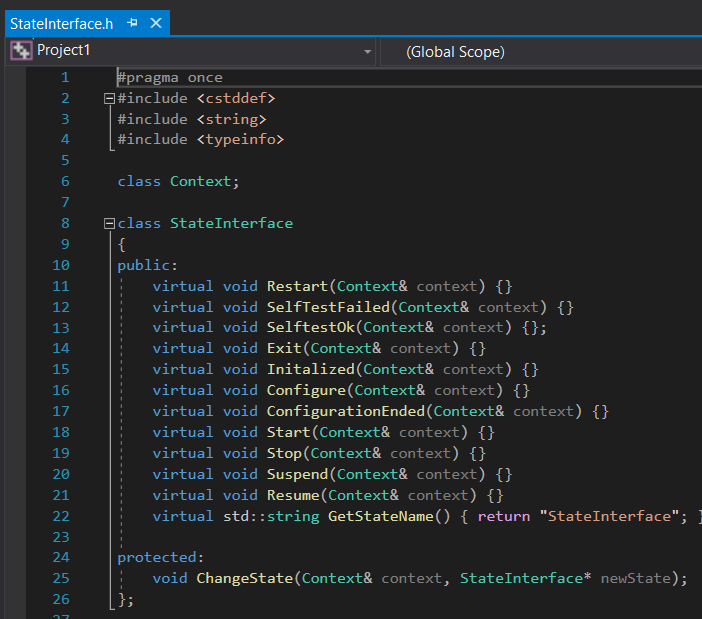


Figure State Interface Definition

There should be only one instance of a state at any point in time. In order to accommodate this requirement **SingletonState** was implemented. As shown in Figure 2, this implementation provides access to the same single instance of the state. Each state extends **SingletonState**.

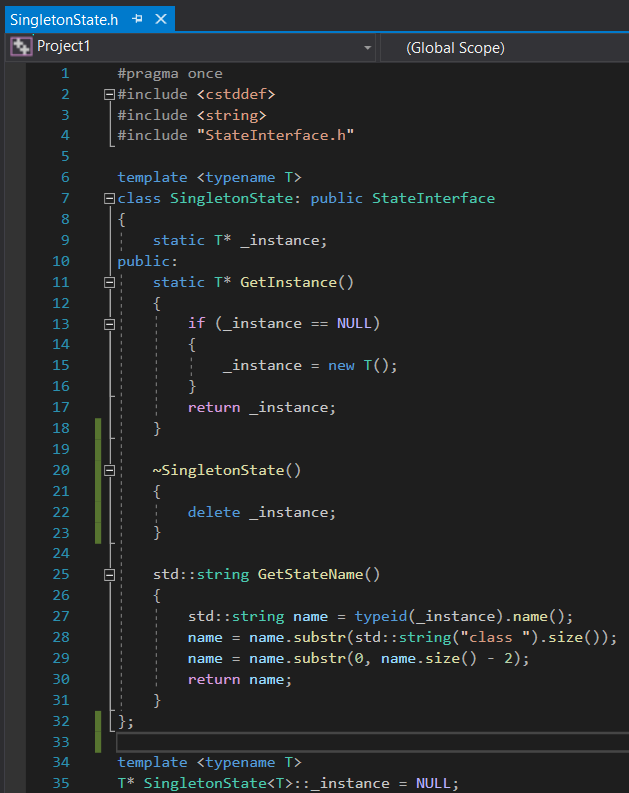


Figure SingletonState template

The required system has two state machines – the global one and the inner one in the **Operational** state. Therefore a **Context** class has been implemented as shown in Figure 3. It provides the functionality of changing states and providing current state name. Changing state implements a dummy delay to simulate a workload associated with changing states.

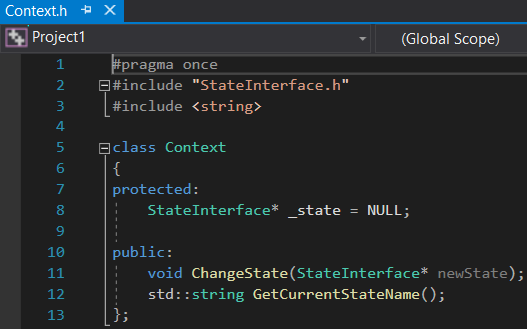


Figure Context Class

The Operational state is a super state (a state and a context at the same time).

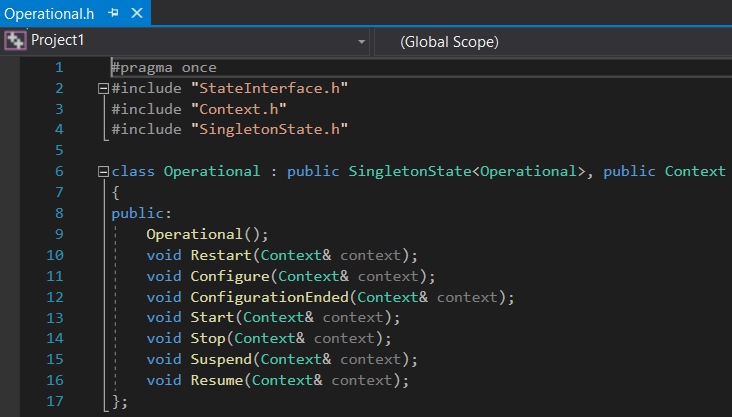


Figure Operational State Declaration

A C++ console application was implemented in order to test the state machine. Part of the implementation of the main loop is shown in the Figure 5. The control loop first accepts instructions from the console invoking specific events. Next, it measures the available time for execution until the state of **EmbeddedSystemX** changes. Once the state change is observed, a new instruction from the user is requested. The time until the state of the system changes can be considered as available for additional computation tasks to be performed in the main control loop.

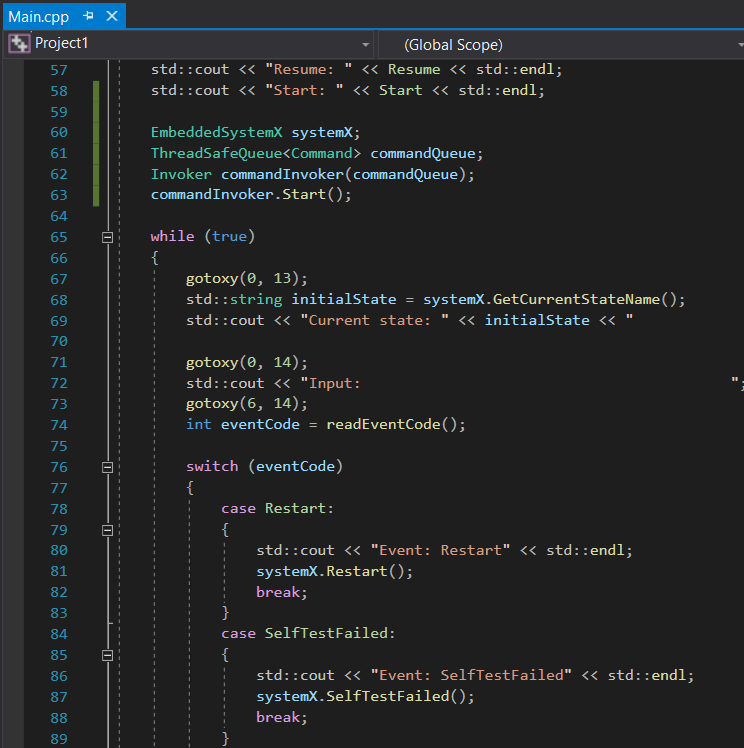


Figure Part of the main loop

# 1.3 Event processing using command pattern

Design was extended with the classes illustrated in the

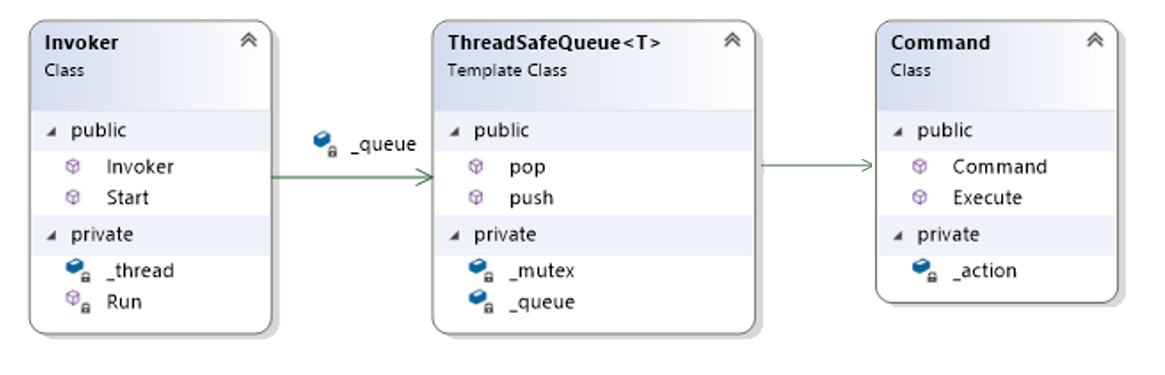


Figure . Command pattern classes

