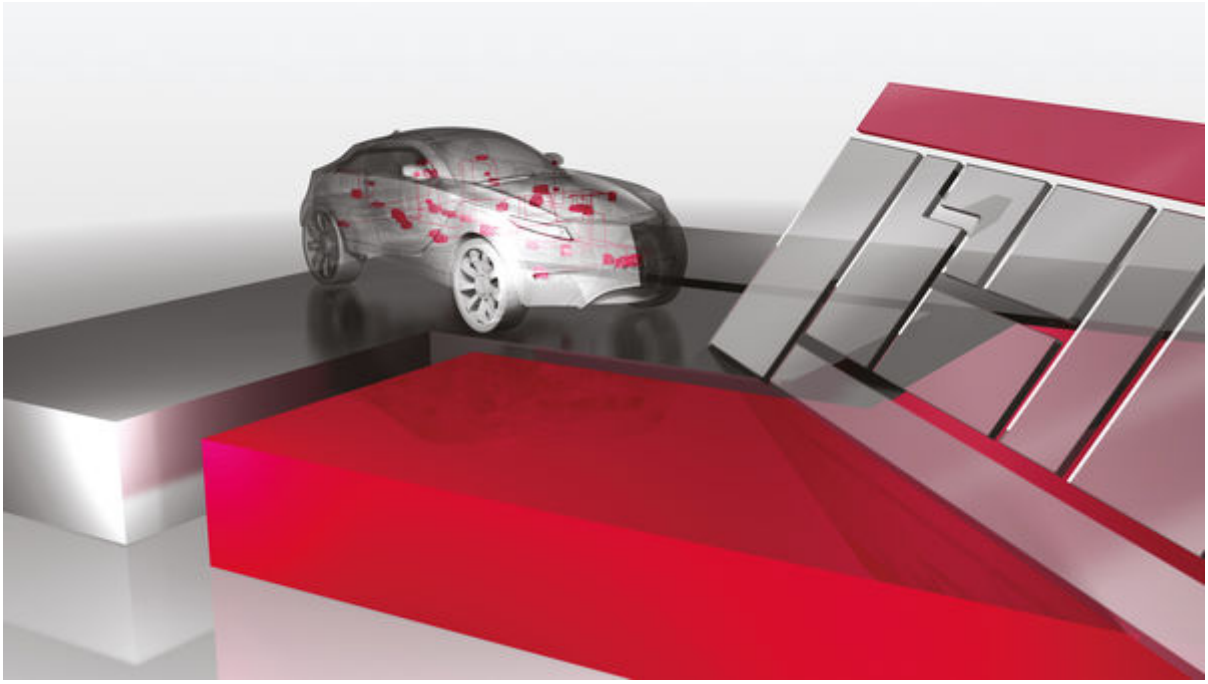


Automotive Software Development

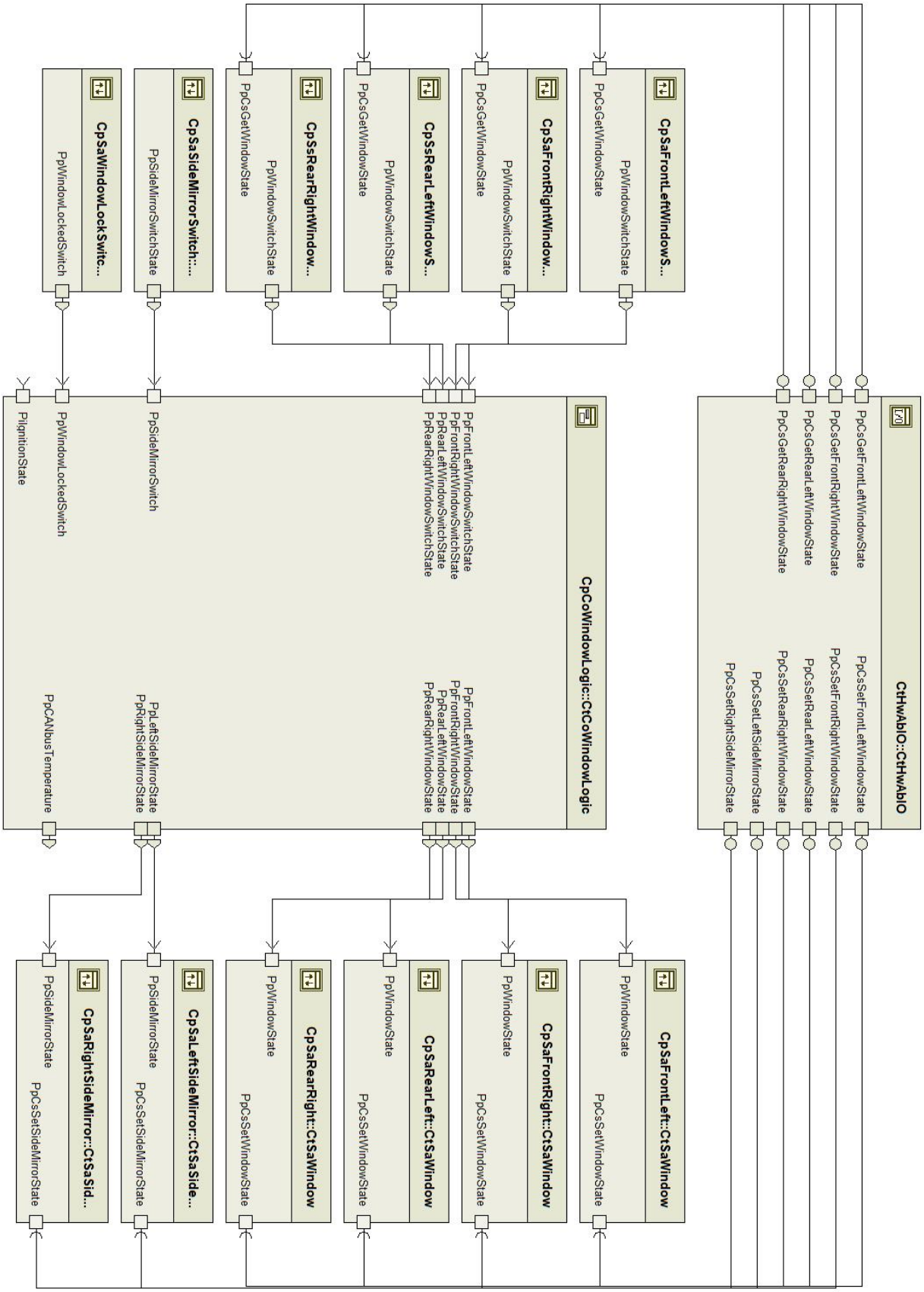
Driver Door Module Control System

Emmanuel Sedicol (20072377)

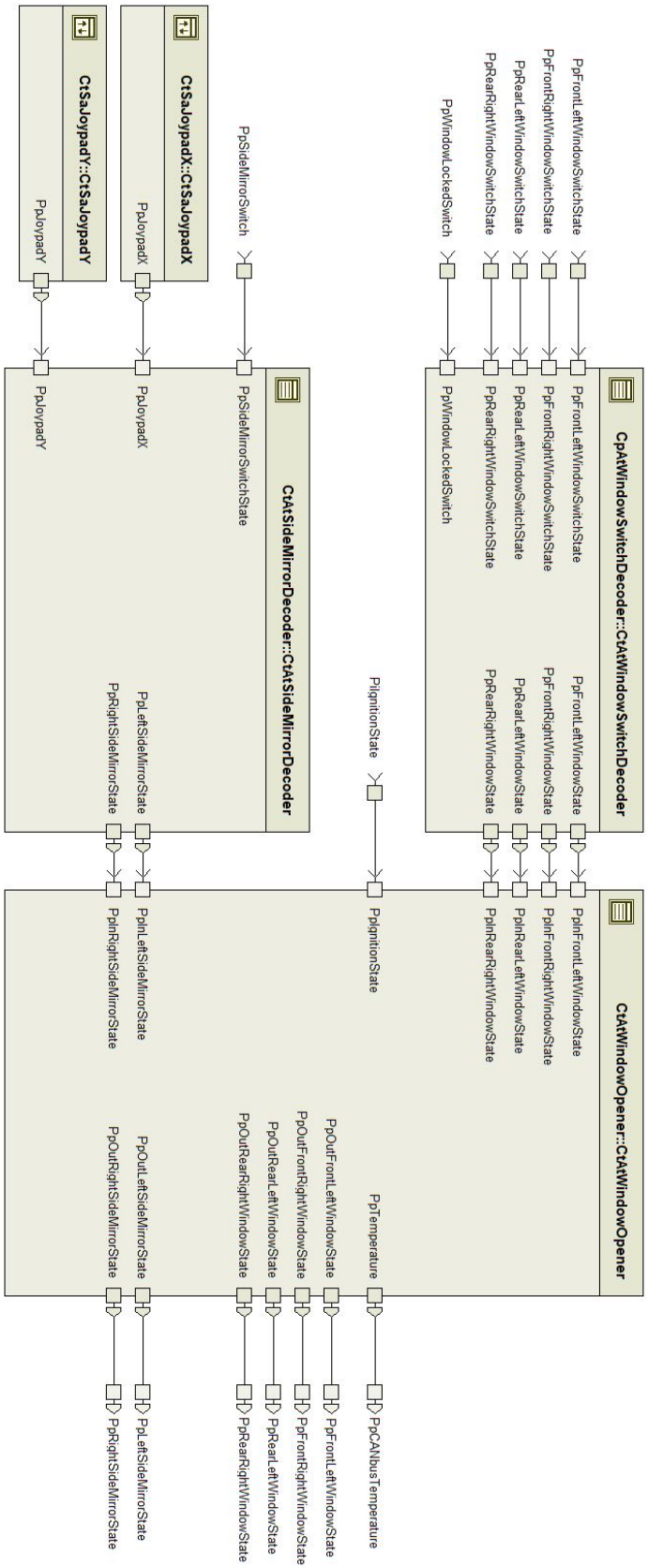
17TH OF NOVEMBER 2019

Table of Contents

1st level SWC's



2nd Level SWC's



Introduction

As part of our Automotive Software Development module we were asked to design and specify a set of AUTOSTAR application components to implement a driver door module. The drive door modules control all four mirrors in a car and controls both the left and right side mirrors. This application will be created in Vectors Davinci Developer where we can auto generate its C code which we can alter or add methods to.

The door module has the following control operations:

- Four independent buttons (1-4) which controls four door mirrors.
- A joypad which controls the selected side mirror.
- A single button when pressed locks all window except the drivers mirror.

Project Activities:

1. Design a set of AUTOSTAR SWCs to implement the door control functions described above.
2. Specify Application data types to use with the SWCs
3. Specify all Port Interface Specifications and port prototypes required.
4. Specify any inter-runnable variables needed.
5. Specify all runnable required, with associated trigger events and port access requirements.

Application Port Interfaces

PiWindowSwitchState: the following sender/ receiver port interface will be use for the four-independent window SWC's. The port interface will transmit data elements in an 8-bit unsigned integer. '0' means button is not pressed and a value of '1' is when button is pressed. We will assign a runnable on each switch SWC that will be triggered when the button is pressed.

PiWindowState: the following sender/ receiver port interface will be used when getting the state of a window. The port accepts or transmits data elements in an 8-bit sint integer. I have used 'sint' because opening a window is positive while closing the window will be negative.

PiWindowLockedSwitch: the following sender/ receiver port interface has a 'Boolean' data type to represent a button's state as on and off. Also, I have assigned a data constraint for this port interface to have a minimum value of '0' and a maximum value of '1'. Each port will have an initial value of 0 or false/ off and this port interface will be used for the Locked Switch SWC.

PiTemperature: the following sender/ receiver port interface will be used for taking temperature readings on the door modules built in sensor. Is has an 8-bit sint data type and its data is constraint to a minimum of -40 to a maximum of 60.

PiSideMirrorSwitchState: the following sender/ receiver port interface will be assigned to the side mirror switch ports which has an 8-bit unsigned integer data type with a constraint of min 0 and max 2. Zero represent the idle state, '1' represent the side mirror switch set to left and '2' mean the switch is set to the right mirror.

PiSideMirrorState: the following sender/ receiver port interface will be used for the horizontal and vertical joypads to set or get the mirror state. This port interface accepts or transmits data elements in an 8-bit sint integer.

PiJoypadX: the following sender/ receiver interface is set to have a 8-sint data type and will be used for the horizontal joypad SWC. It has a constraint called 'JoypadContraint' which has a min value of -50 and a max value of 50, this means the when the max and min values are reached the side mirror can't be move anymore.

PiJoypadY: the following sender/ receiver interface is set to have a 8-sint data type and will be used for the vertical joypad SWC. It has a constraint called 'JoypadContraint' which has a min value of -50 and a max value of 50, this means the when the max and min values are reached the side mirror can't be move anymore.

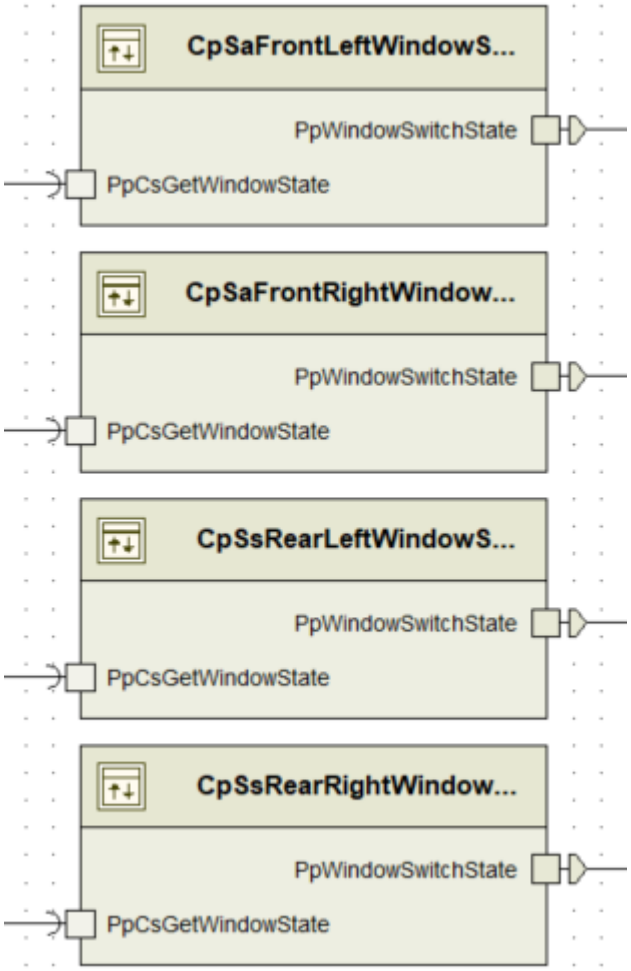
PiIgnitionState: the following sender/ receiver interface has a Boolean data type.

PiCsSetWindowState: the following port interface is a client/ server port which be used for the Get/Set operation on serve ports. The interface has an 8-sint data type.

PiCsGetWindowState: the following port interface is a client/ server port which be used for the Get/Set operation on serve ports. The interface has an 8-sint data type.

Individual Software Components

CpSaWindowSwitch (4)

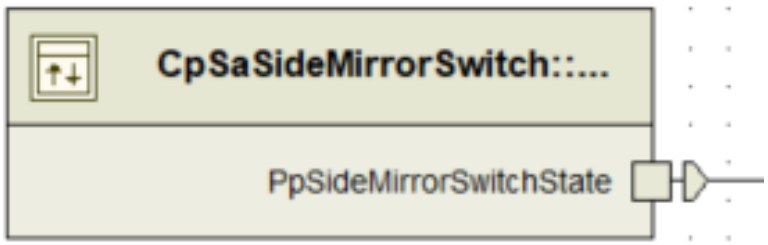


This sensor actuator component contains one sender (PpWindowSwitchState) and one client (PpCsGetWindowState) port. This component has three other instances, one for each window switch. Each button is independent meaning that each switch will have its own sensor actuator component and have its own unique I/O ports.

PpWindowSwitchState: this interface is responsible for transmitting switch values where it values is decoded in the second level. The SWC contains a runnable that triggers when the button is pressed, then the correspond result is sent to the 'CpAtWindowSwitchDecoder' component via the port access 'PpWindowSwitchState.DeWindowSwitchState'. The runnable is triggered by client port after is accesses the I/O port in the ECU.

PpCsGetWindowState: this port invokes a get method to the connected server port. The value receive will be the current state or value of the corresponding window. This is done so that when a user clicks the switch it will start from the last state.

CpSaSideMirrorSwitch

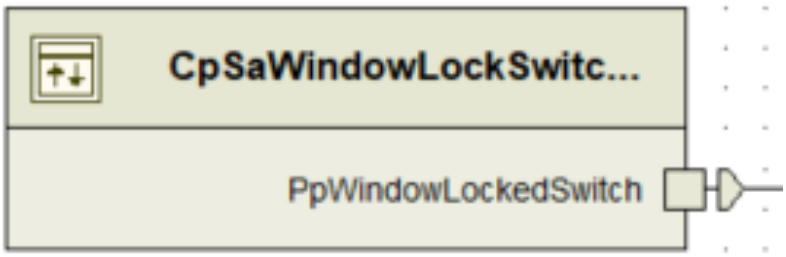


This is another sensor actuator component that has three possible values, ‘0’ for idle state, ‘1’ when left mirror is picked and ‘2’ when the right mirror is picked. The SWC contains one sender port (PpSideMirrorState) that can send an 8-bit unsigned integer values to the window login composition. The value sent by this component will be the deciding factor on the runnable ‘SetLeftRightMirror’.

LeftRight runnable: the runnable for component is triggered periodically (20msec). The runnable will handle whether switch is clicked or not and which side mirror is to be active. The runnable has an explicit write access to the sender port ‘PpWindowLockedSwitch’.

DEP/Operation/Trigger	Access	Name
PpSideMirrorSwitchState.DeSideMirrorSelectionState	Write (explicit)	SEND_PpSideMirrorSwitchState_DeSi...

CpSaWindowLockSwitch

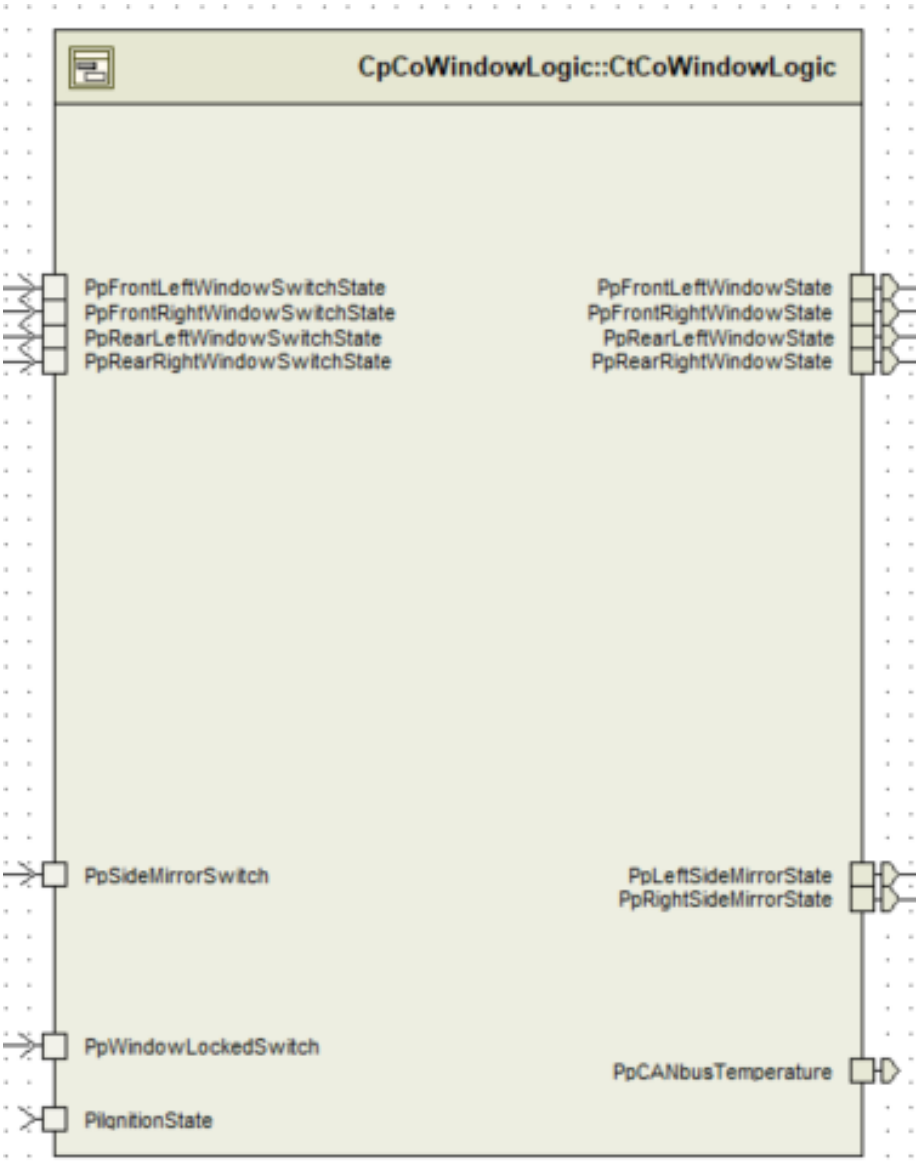


On the project description we are asked to implement a button that once it was pressed it will lock all window operations on the car except the drivers mirror. This sensor actuator component handles that operation, it contains one sender port ‘PpWindowLockedSwitch’ which has a Boolean data type for values on and off.

OnOff runnable: the runnable implemented in this component handles the value to be sent to the next lower level where all the window switch states are decoded. The runnable has explicit write access to its sender port and is triggered periodically (20 msec).

DEP/Operation/Trigger	Access	Name
PpWindowLockedSwitch.DeWindowLock	Write (explicit)	SEND_PpWindowLockedSwitch_DeW...

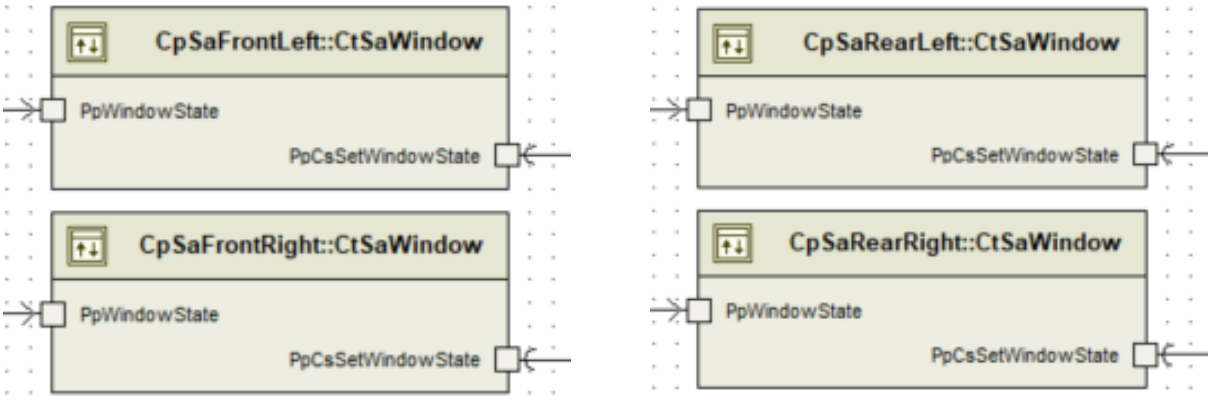
CpCoWindowLogic



The window logic SWC is a composition component that lets us encapsulate atomic application SWC for windows, side mirrors and ignition runnables for higher abstraction levels. The window logic has four receiver port for all four mirrors sides in the car, one receiver port for the side mirror selection switch, one receiver port for the locked switch which is fed on the side mirror decoder to handle the selection of left or right side mirror.

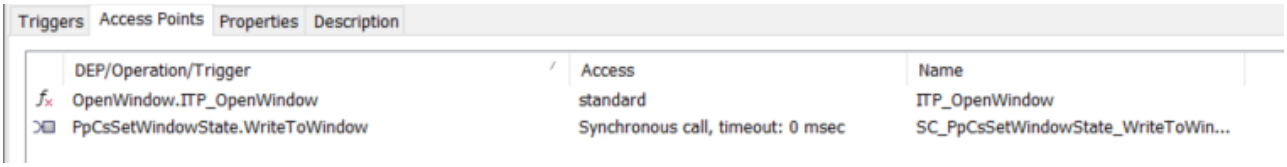
The sender ports are associated with the states of each mirrors and is connected to its corresponding sensor actuator which will send the operation to the mirror Ecu through the virtual functional bus.

CpSaWindow

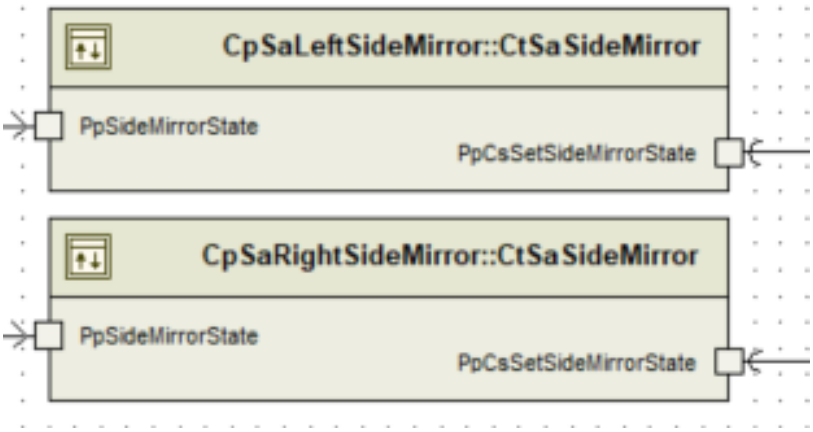


The window component has four other instantiations for the other four mirrors in the car. Each window component is a sensor actuator and is connected to the hardware abstraction component which communicates with the ECU. The hardware abstraction components set the mirror based on the values the window SWC receive. It contains two ports, on receiver port for receiving the window state and one sender port to ask the ECU to invoke set operations.

OpenWindow runnable: this runnable is in charge of the setting the values or states of the mirrors and invokes a method to open, close or adjust the mirrors. The runnable is triggered when a data input is detected from the corresponding port on the window logic SWC. It has access to the 'WriteToWindow' operation.



CpSaSideMirror



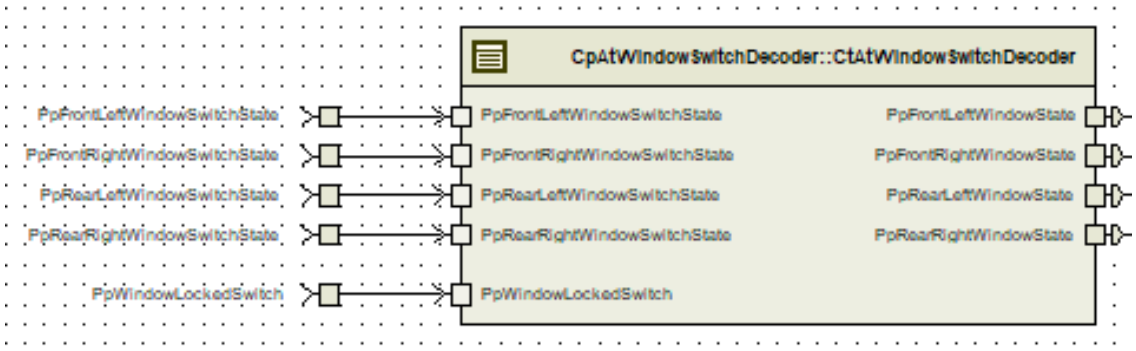
The side mirror sensor actuator component is very similar to the one above (CpSaWindow) because this SWC communicate to the abstraction layer through a server/client port interface and invoking get or set function to the ECU ports. The side mirror SWC has two instantiations for the horizontal joypad and the vertical joypad.

SetSideMirror runnable: The SWC takes in the mirror state from the window logic and a runnable is then activated which will invoke the write method.

Trigger	Disabled in Modes	Activation Reason	Name
PpSideMirrorState.DeSidMirrorState		-	DRT_SetSideMirror_PpSideMirrorStat...

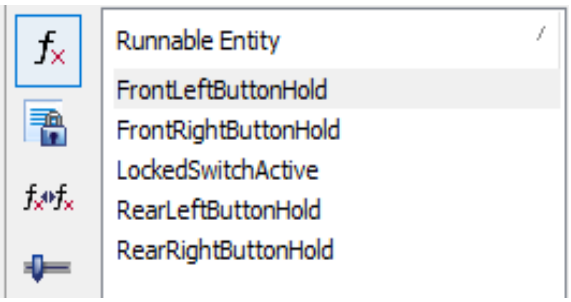
DEP/Operation/Trigger	Access	Name
PpCsSetSideMirrorState.WriteToSideMirror	Synchronous call, timeout: 0 msec	SC_PpCsSetSideMirrorState_WriteT...

CpAtWindowSwitchDecoder



The function for this atomic type SWC is take all the inputs (window state) from the level one sensor actuators and have the runnable do the job to make sure that the state sent to the next SWC is error free and that the vale is real. One of the main function is to detect when a button is being hold for over one second the window will fully go up or down. Since I have set data constraints to only take in values from -50 to 50 we can set values of states to either values.

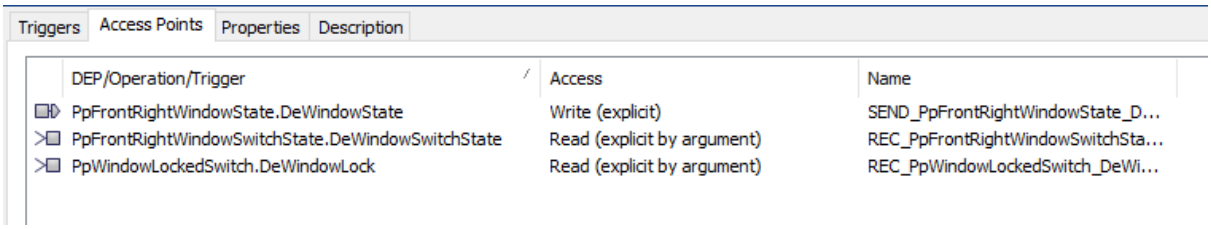
FrontLeftButtonnHold, FrontRightButtonnHold, RearLeftButtonnHold and RearRightButtonnHold runnable:



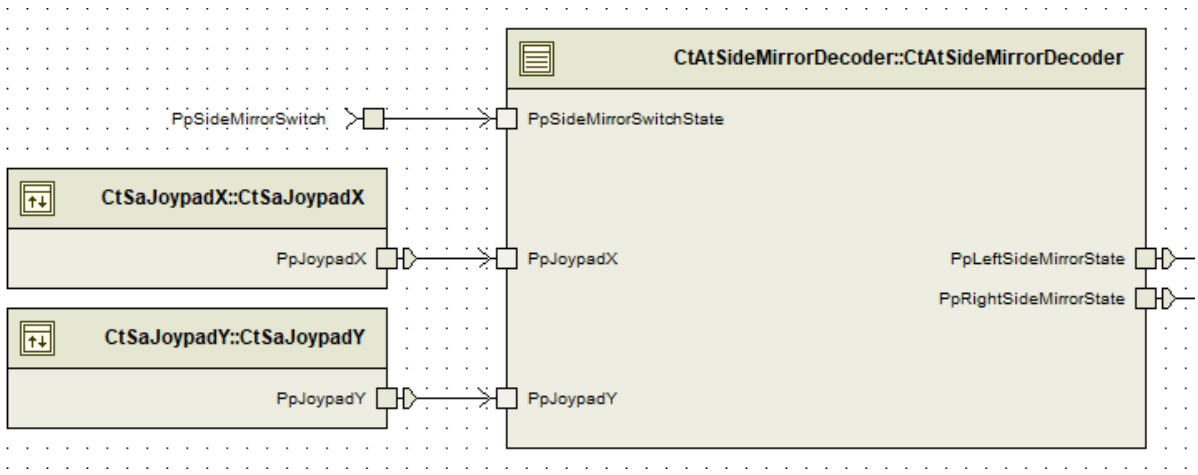
Runnable triggers when we have date input that's sending continuous values on its receiver port and sends a state of fully close or open to its sender port. The runnable has explicit access to its receiver and sender ports.

LockedSwitchActive: In the switch decoder SWC there is an independent runnable that handles data input from the lock switch state port, the runnable is triggered when there is an input from the receiver port and it then locks all other windows. The runnable sends a value of '0' to the rest of the window state sender port and takes the front right window state and send its values to the corresponding port.

Access points of the 'LockedSwitchAvtive' runnable:



CpAtSideMirrorDecoder



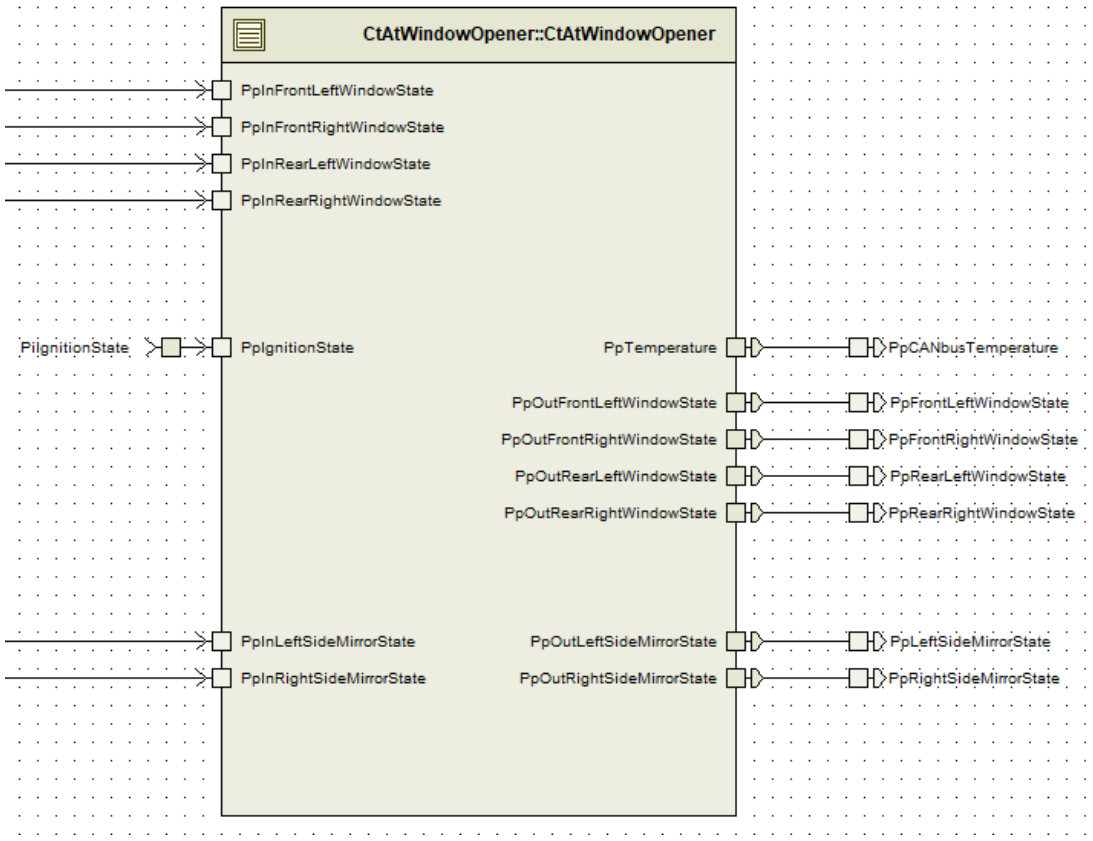
The other decoder in the window logic composition is for the side mirror. It has inputs from two sensor actuator defined as joypad x(horizontal) and joypad y(vertical). The side mirror atomic SWC has two sender ports which send the state of either side mirror one at a time basing on the value we get from the 'PpSideMirrorSwitchState'.

SetLeftRightMirror runnable: this runnable is triggered when we get a value or either '1' or '2' and not '0' from the side mirror switch receiver port. If the value is one we take input from the left sensor actuator and vice versa for the right side mirror. The output will only one side mirror state at a time. The sender port send the calculated and finalise value to the 'windowOpener' SWC.

Runnable access points:

DEP/Operation/Trigger	Access	Name
PpJoypadX.DeJoypadX	Read (explicit by argument)	REC_PpJoypadX_DeJoypadX
PpJoypadY.DeJoypadY	Read (explicit by argument)	REC_PpJoypadY_DeJoypadY
PpLeftSideMirrorState.DeSidMirrorState	Write (explicit)	SEND_PpLeftSideMirrorState_DeSid...
PpRightSideMirrorState.DeSidMirrorState	Write (explicit)	SEND_PpRightSideMirrorState_DeSid...
PpSideMirrorSwitchState.DeSideMirrorSelectionState	Read (explicit by argument)	REC_PpSideMirrorSwitchState_DeSid...

CtAtWindowOpener



The main function of this SWC is to not only have one SWC to do the sending of values but also to pack all the finalise values and have a way to get a record the states so the when the ignition turns off and turns back on he windows go back to their last state. This is done by a runnable that called ‘InigionOnOff’.

IgnitionOnOff runnable: the runnable is triggered when we get an input value of ‘0’, the runnable records the state values. Then when the ignition turns back on it will automatically send the states out to the sender ports.

Access points:

DEP/Operation/Trigger	Access	Name
PpIgnitionState.DeIgnitionState	Read (explicit by argument)	REC_PpIgnitionState_DeIgnitionState
PpInFrontLeftWindowState.DeWindowState	Read (explicit by argument)	REC_PpInFrontLeftWindowState_De...
PpInFrontRightWindowState.DeWindowState	Read (explicit by argument)	REC_PpInFrontRightWindowState_D...
PpInLeftSideMirrorState.DeSidMirrorState	Read (explicit by argument)	REC_PpInLeftSideMirrorState_DeSid...
PpInRearLeftWindowState.DeWindowState	Read (explicit by argument)	REC_PpInRearLeftWindowState_De...
PpInRearRightWindowState.DeWindowState	Read (explicit by argument)	REC_PpInRearRightWindowState_D...
PpInRightSideMirrorState.DeSidMirrorState	Read (explicit by argument)	REC_PpInRightSideMirrorState_DeSi...
PpOutFrontLeftWindowState.DeWindowState	Write (explicit)	SEND_PpOutFrontLeftWindowState...
PpOutFrontRightWindowState.DeWindowState	Write (explicit)	SEND_PpOutFrontRightWindowStat...
PpOutLeftSideMirrorState.DeSidMirrorState	Write (explicit)	SEND_PpOutLeftSideMirrorState_De...
PpOutRearLeftWindowState.DeWindowState	Write (explicit)	SEND_PpOutRearLeftWindowState_...
PpOutRearRightWindowState.DeWindowState	Write (explicit)	SEND_PpOutRearRightWindowState...
PpOutRightSideMirrorState.DeSidMirrorState	Write (explicit)	SEND_PpOutRightSideMirrorState_D...