Main Propulsion System Software Requirements Specification

Volume 2 - Orbiter

Checkout and Launch Control System (CLCS)

84K01450-101

Approval:

Manager, CLCS Application Software	Date	Manager, USA Application Software	Date
USA IPT Co-Lead	Date	NASA IPT Co-Lead	Date
CSCI Lead	Date	System Specialist	Date

Prepared By:

Date

Note: *See "Supporting Document Note" on following page*

REVISION HISTORY

REV	DESCRIPTION	DATE
Draft	Posted to web per author's request. (pbw)	7/5/00
Basic	Signature blocks added, promoted and zipped to route for signature. (pbw)	7/5/00
Basic	Posted signed approved document. (pbw)	7/21/00

Supporting Document Note:
Acronyms and definitions of common CLCS terms may be found in *84K00240, CLCS Acronyms* and *84K00250, CLCS Project Glossary*.

TABLE OF CONTENTS

1. INTRODUCTION 1
1.1. PURPOSE 1
1.2. SCOPE 1
1.3. DEFINITIONS, ACRONYMS, AND ABBREVIATIONS 1
1.4. REFERENCE DOCUMENTS 1
1.5. OVERVIEW 1
2. OVERALL DESCRIPTION 1
2.1. PRODUCT PERSPECTIVE 1
2.1.1. System Interfaces 2
2.1.2. User Interfaces 2
2.1.3. Hardware Interfaces 2
2.1.4. Software Interfaces 2
2.1.5. Communication Interfaces 2
2.2. PRODUCT FUNCTIONS 2
2.3. USER CHARACTERISTICS 3
2.4. CSCI SYSTEM LEVEL SAFETY REQUIREMENTS 3
2.4.1. Response to System Hazards 3
2.4.2. Loss of Remote Monitor and/or Control Capability 3
2.4.3. Redundancy Requirements 3
2.5. CONSTRAINTS 3
2.6. ASSUMPTIONS AND DEPENDENCIES 3
2.7. APPORTIONING OF REQUIREMENTS 3
3. SPECIFIC REQUIREMENTS 4
3.1. MOLECULE CLASSES 4
3.1.1. MPS_Valve_ORB 4
3.1.2. MPS_SolValve_ORB 7
3.1.3. MPS_Valve_ORB_Outboard_F&D 9
3.1.4. MPS_Valve_ORB_Inboard_F&D 30
3.1.5. MPS_Valve_ORB_LH2_PreValve 50
3.1.6. MPS_Valve_ORB_4inchdisc 84
3.1.7. MPS_Valve_ORB_LO2_Prevalve 107
3.1.8. MPS_Valve_ORB_LH2HiPt_Bleed 164
3.1.9. MPS_Valve_ORB_Topping 178
3.1.10. MPS_Valve_ORB_RTLS_Dump 191
3.1.11. MPS_Valve_ORB_LO2OvBrd_Bleed 208
3.1.12. MPS_Valve_ORB_Pogos 234
3.1.13. MPS_Valve_ORB_Recirc_Isolation 246
3.1.14. MPS_Valve_ORB_Relief_Isolation 252
3.1.15. MPS_Valve_ORB_17inch_Disc 265
3.1.16. MPS_Valve_ORB_Latch 296
3.1.17. MPS_SolValve_ORB_FCV 328
3.1.18. MPS_SolValve_ORB_SMERegIsoA 337
3.1.19. MPS_SolValve_ORB_SMERegIsoB 344
3.1.20. MPS_SolValve_ORB_MPSRegIso 352
3.1.21. MPS_SolValve_ORB_InterconnectIn 359
3.1.22. MPS_SolValve_ORB_2Way 372
3.1.23. MPS_SolValve_ORB_2Way_IntOut 378
3.1.24. MPS_SolValve_ORB_RTLSPress 381
3.1.25. MPS_ORB_Pressure_Sensor_wBias 392
3.1.26. MPS_ORB_Temp_Sensor 396
3.1.27. MPS_ORB_SpeedSensor//VAB/PAD Class// 398

3.1.28. MPS_ORB_Analog_wLimits 399
3.2. COMPOSITE CLASSES 399
3.2.1. MPS_Recirc 399
3.2.2. MPS_ORB_PneumaticRegCntl 410
3.2.3. MPS_ORB_SMERegCntl 415
3.2.4. MPS_ORB_17inch_Disconnect_Assembly 422
3.2.5. MPS_ORB_ETOI 424
3.2.6. MPS_LH2_Control_Logic_Manager 470
3.2.7. MPS_LO2_Control_Logic_Manager 484
3.2.8. MPS_System 494
3.2.9. MPS_Hydrogen_Subsystem 498
3.2.10. MPS_Oxygen_Subsystem 508
3.2.11. MPS_Helium_Subsystem 516
3.3. REACTIVE CONTROL LOGIC CLASSES 523
3.3.1. MPS_RCL_GHe_Bottle_Press 523
3.3.2. MPS_RCL_Pneu_Reg_Press 529
3.3.3. MPS_RCL_E1_Reg_Press 534
3.3.4. MPS_RCL_E2_Reg_Press 539
3.3.5. MPS_RCL_E3_Reg_Press 544
3.3.6. MPS_RCL_GHe_Bottle_Temp 549
3.3.7. MPS_RCL_LO2_Manifold_Press 556
3.3.8. MPS_RCL_LH2_Manifold_Press 561
3.3.9. MPS_RCL_LO2_PV9_OP_Ind 566
3.3.10. MPS_RCL_LO2_PV10_OP_Ind 571
3.3.11. MPS_RCL_LO2_PD1_OP_Ind 575
3.3.12. MPS_RCL_LH2_PV11_OP_Ind 582
3.3.13. MPS_RCL_LH2_PV12_OP_Ind 586
3.3.14. MPS_RCL_LH2_PV13_OP_Ind 591
3.3.15. MPS_RCL_LH2_PD3_OP_Ind 596
3.3.16. MPS_RCL_LH2_PD2_OP_Ind 601
3.4. PREREQUISITE CONTROL LOGIC CLASSES 606
3.4.1. MPS_PCL_PV9_CL_CMD 606
3.4.2. MPS_PCL_PV9_OP_CMD 622
3.4.3. MPS_PCL_PV10_CL_CMD 638
3.4.4. MPS_PCL_PV10_OP_CMD 656
3.4.5. MPS_PCL_PD1_CL_CMD 674
3.4.6. MPS_PCL_PD1_OP_CMD 686
3.4.7. MPS_PCL_PV19_CL_CMD 699
3.4.8. MPS_PCL_PV7_CL_CMD 705
3.4.9. MPS_PCL_PD1_LATCH_LK_CMD 713
3.4.10. MPS_PCL_PD1_LATCH_UL_CMD 725
3.4.11. MPS_PCL_LO2_PREVALVES_CL_CMD 737
3.4.12. MPS_PCL_LO2_PREVALVES_OP_CMD 745
3.4.13. MPS_PCL_LO2_POGO_CL_CMD 752
3.4.14. MPS_PCL_PV11_CL_CMD 758
3.4.15. MPS_PCL_PV11_OP_CMD 775
3.4.16. MPS_PCL_PV12_CL_CMD 792
3.4.17. MPS_PCL_PV12_OP_CMD 811
3.4.18. MPS_PCL_PV13_OP_CMD 830
3.4.19. MPS_PCL_PD2_CL_CMD 845
3.4.20. MPS_PCL_PD2_OP_CMD 858
3.4.21. MPS_PCL_PD2_LATCH_LK_CMD 871
3.4.22. MPS_PCL_PD2_LATCH_UL_CMD 883
3.4.23. MPS_PCL_PD3_CL_CMD 895

3.4.24. MPS_PCL_PD3_OP_CMD....900
3.4.25. MPS_PCL_PV8_CL_CMD....906
3.4.26. MPS_PCL_LV74_OP_CMD....914
3.4.27. MPS_PCL_LV75_OP_CMD....927
3.4.28. MPS_PCL_RECIRC_OP_CMD....940
3.4.29. MPS_PCL_RTLS_DUMP_OP_CMD....949
3.4.30. MPS_PCL_PT_SENS_PWR_CMD....956
3.4.31. MPS_PCL_FCV1_CO_CMD....962
3.4.32. MPS_PCL_FCV2_CO_CMD....969
3.4.33. MPS_PCL_FCV3_CO_CMD....977
3.4.34. MPS_PCL_PNEU_OP_CMD....984
3.4.35. MPS_PCL_LV10_OP_CMD....999
3.4.36. MPS_PCL_BLOWDOWN_OP_CMD....1012
3.5. DATA FUSION SEQUENCES....1017
3.6. TEST CLASSES....1017
3.6.1. MPS Bus Redundancy....1017
3.6.2. LOX MDM Command Redundancy....1024
3.6.3. LH2 MDM Command Redundancy....1053
3.6.4. MPS 750 Pneumatic Decay....1057
4. DISPLAYS....1103
4.1. GENERAL....1103
4.2. MPS LH2 SYSTEM TEST AND CHECK-OUT DISPLAY - SCHEMATIC....1104
4.2.1. Description....1104
4.2.2. Control Points....1104
4.2.3. Objects....1104
4.3. MPS LH2 SYSTEM TEST AND CHECK-OUT DISPLAY - TABULAR....1107
4.3.1. Description....1107
4.3.2. Control Points....1107
4.3.3. Objects....1107
4.4. MPS LH2 SYSTEM TEST AND CHECK-OUT DISPLAY - ENUMERATED STATE....1108
4.4.1. Description....1108
4.4.2. Control Points....1108
4.4.3. Objects....1108
4.5. MPS LO2 SYSTEM TEST AND CHECK-OUT DISPLAY - SCHEMATIC....1108
4.5.1. Description....1108
4.5.2. Control Points....1108
4.5.3. Objects....1108
4.6. MPS LO2 SYSTEM TEST AND CHECK-OUT DISPLAY - TABULAR....1111
4.6.1. Description....1111
4.6.2. Control Points....1111
4.6.3. Objects....1111
4.7. MPS LO2 SYSTEM TEST AND CHECK-OUT DISPLAY - ENUMERATED STATE....1111
4.7.1. Description....1111
4.7.2. Control Points....1111
4.7.3. Objects....1111
4.8. MPS GHE SYSTEM TEST AND CHECK-OUT DISPLAY - SCHEMATIC....1112
4.8.1. Description....1112
4.8.2. Control Points....1112
4.8.3. Objects....1112
4.9. MPS GHE SYSTEM TEST AND CHECK-OUT DISPLAY - TABULAR....1112
4.9.1. Description....1112
4.9.2. Control Points....1112
4.9.3. Objects....1112
4.10. MPS GHE SYSTEM TEST AND CHECK-OUT DISPLAY - ENUMERATED STATE....1112

4.10.1. Description 1112
4.10.2. Control Points 1112
4.10.3. Objects 1112
4.11. MPS SYSTEM MESSAGE DISPLAY 1113
4.11.1. Description 1113
4.11.2. Control Points 1113
4.11.3. Objects 1113
5. INSTANCES 1113

1. INTRODUCTION

1.1. PURPOSE

This document will establish the requirements for the Main Propulsion System Orbiter Systems Application Software set and will reflect the needs of the users. The requirements will be structured to allow an Object Oriented approach to the software development process.

1.2. SCOPE

The requirements will be stated in sufficient detail to allow the complete recode of the MPS software set and will define the following:
OBJECTS
OPERATIONS
USER INTERFACES
DISPLAYS
The requirements will cover MPS Orbiter Systems test and checkout operations from landing through liftoff.

1.3. DEFINITIONS, ACRONYMS, AND ABBREVIATIONS

Refer to the STS Glossary, Acronyms and Abbreviations (NASA reference publication).

1.4. REFERENCE DOCUMENTS

KLO-82-0014 LO2/LH2/MPS/SSME LPS Applications Software Requirements Document
KLO-82-0015 LO2/LH2/MPS/SSME LPS Applications Software Specifications Document

1.5. OVERVIEW

2. OVERALL DESCRIPTION

2.1. PRODUCT PERSPECTIVE

This SRS provides the requirements for the Main Propulsion System (MPS) orbiter hardware related software set. Associated software sets include SSME, CCME, LO2 and LH2 systems. The software set developed and operated using the information contained in this document will be able to provide monitor and control over the MPS system components. The MPS checkout includes standalone checkout mode through integrated cryogenic loading and launch operations. This SRS includes all MPS requirements for OPF/MLP(VAB/PAD) associated components. Associated Ground Support Equipment (GSE, including LOX/LH2 Systems GSE) requirements are contained in sections where the GSE and vehicle are integrally related in function (Integrated Load/Launch Operations, MPS recirc pump operation, Helium Bottle Fill, etc.,) .

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

2.1.1. System Interfaces

2.1.1.1. Class Diagram

2.1.2. User Interfaces

The operator shall be able to monitor the state of the MPS system using displays that follow the format of the orbiter pneumatic (GHe), liquid hydrogen (LH2), liquid oxygen (LO2), gaseous hydrogen (GH2), gaseous oxygen (GOX) and electrical schematics. Details of all associated displays are contained in section 4.0.

The displays shall be selectable from a over-all menu of all MPS displays provided on the Console Navigation System (CNS) drop down Display Menu.

Included on each display will be a pull-down menu. The menu will allow the operator to select from operations which pertain to that display functionality as defined in section 4.

Operations available for a particular object will be accessible from a dropdown menu on the associated display when that item is selected.

The schematic displays will include an EMERGENCY SAFING button in the lower left hand corner. This button, when selected and executed, will perform the emergency safing operations required for that panel/system as defined in the applicable section of this SRS.

Multiple displays shall be able to be viewed simultaneously with the capability to window (move / resize / overlap) and iconify displays.

There shall be a monitor indicating that the display is cyclically updating or a method employed to signal the operator that a display is not updating due to a hardware/software problem.

2.1.3. Hardware Interfaces

2.1.4. Software Interfaces

2.1.5. Communication Interfaces

In an attempt to prototype applications software tolerant of sporadic system failures which result in failure to read data from the CVT, for specific classes the return code for data reads should be monitored and a message output to the operator noting the function and the measurement for which the error was generated. In functions that determine state (overall/indicator/command), if a read on any of the data fails, "ERROR" should be appended to the overall state which was previously calculated with valid data. For other software functions (turnOn/open/etc) the software shall continue processing (after outputting a message) for fails on data read, if the fail to read is at a operator imput point the software shall treat a fail to read as a fail of the required statet and prompt the operator prior to continuing. The scope of implementation of the prototyping will be up to the individual programmers and will be implemented only without adversely impacting the schedule, testing, or overall complexity of the modules. This prototyping (output of the messages) will be a good way to characterize the level of system reliability and do a first cut at detection and processing for these failures should they be present in subsequent releases.

2.2. PRODUCT FUNCTIONS

The MPS Orbiter Systems Application Set provides all end item command and control capability necessary to support stand-alone and integrated systems test and check-out, including load and landing.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

2.3. USER CHARACTERISTICS

All users of the MPS Orbiter Application Set are experienced system engineers who are thoroughly familiar with the system hardware. These system engineers are certified console operators with years of experience performing systems test and check-out. Although trainees sometimes use the system, they do so only while under the direct supervision of certified console operators. Users should be assumed to be fully knowledgeable of the system hardware. Little or no C++ or Control Shell expertise should be assumed since most operators have little programming experience.

2.4. CSCI SYSTEM LEVEL SAFETY REQUIREMENTS

2.4.1. Response to System Hazards

MPS Reactive Control Logic provides the necessary controls to prevent inadvertent potentially hazardous command issuance.

2.4.2. Loss of Remote Monitor and/or Control Capability

If during system test and check-out, visibility or control of any end item is lost, the system will fail "as-is". No specific requirements exist for MPS Orbiter Systems systems safing upon loss of command and control or visibility. The operator should immediately be enunciated with a message as to the cause/source of the failure.

2.4.3. Redundancy Requirements

It is assumed that CLCS system redundancy will be provided in the Command and Control Processor, the Data Distribution Processor, and all hardware links. Command and Control Workstation redundancy is provided by multiple stations running simultaneously which are capable of being expeditiously configured to serve as an MPS workstation.

2.5. CONSTRAINTS

2.6. ASSUMPTIONS AND DEPENDENCIES

Messages to be received by console operator will be displayed to one of two locations: System messages and program generated detailed health messages will be displayed to a scratchpad window, a <PAGE-A> (from LPS) type screen. Emergency, Safing and top level control related information messages will be provided on a System/Test Specific sub window, a <PAGE-B> (from LPS) type screen. The general destination of each message will be enumerated in the Requirements.
For all classes, Attribute and Aggregation Tables define client acces read and write privileges. "R" allows read only access (e.g., "getValueOf()"), "W" allows for access/implementation of listed methods (e.g., "setCmdOn()").

2.7. APPORTIONING OF REQUIREMENTS

The Requirements shall be appropriately apportioned according to the requirements contained thein.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3. SPECIFIC REQUIREMENTS

3.1. MOLECULE CLASSES

3.1.1. MPS_Valve_ORB

3.1.1.1. Description

MPS_Valve_ORB is an abstract base class for the different valve types used by the MPS systems. This base class includes several types of MPS valves: electrical (solenoid) valves, pneumatically actuated monostable valves, and pneumatically actuated bistable valves. Monostable valves (which include all solenoid valves) maintain a normally open or closed state and are commanded (solenoid is energized) to the opposite position. Monostable valves then return (by spring force) to the normal state when commands are removed (solenoid is de-energized). MPS bistable valves (Fill and drains, prevalves, ORB/ET disconnects etc.,) maintain the last commanded position until actuator driven to the opposite position by energizing the opposite state solenoid. Valves will be identified by an ID number (LV or PV) and a description. Valves will have the capability to publish their state and be cycled to either the open or closed state by operator input or sequencer command.

3.1.1.1.1. Instance Index

Instance	Other Users

3.1.1.1.2. Class Diagram

<insert class diagram here>

3.1.1.1.3. Attributes and Aggregations

Attribute	Data Type	Initial Value	Publish / Persistence	Client Access	Description
valveID	String				
valveDescription	String				
state	MPS_ORB_VALVE_STATE		PUB		
indState	MPS_ORB_VALVE_IND_STATE		PUB		
cmdState	MPS_ORB_VALVE_CMD_STATE		PUB		
swSwOpenCycleTime	PA_SECS		PUB		
swSwCloseCycleTime	PA_SECS		PUB		
sigSwOpenCycleTime	PA_SECS		PUB		
sigSwCloseCycleTime	PA_SECS		PUB		

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

cycleTimeSuspect	ENUM_ON_OFF		PUB		

3.1.1.1.4. Methods Index

Operation	Description
autoCycle()	This method will perform valve open/close autocycle operations.
sendOrbValveMsg(int message no., str FDname):char*	This method displays generic MPS valve messages. The message displayed is determined by the integer passed parameter.
calculateCycleTime(sig, sw1, sw2 cycletype)	This method calculates the switch-to-switch and signal-to-switch cycle times and displays time to page A.

3.1.1.1.5. Associations

Association	Type	Class	Description

3.1.1.2. State Description

3.1.1.2.1. State Diagram

<insert state diagram here>

3.1.1.2.2. State Table

3.1.1.3. Operations

3.1.1.3.1. sendOrbValveMsg(int MsgNum, str FDname)

3.1.1.3.1.1. Descriptive Text

This method returns generic MPS valve messages to the MPS valve software..

3.1.1.3.1.2. Sequence

```
sendOrbValveMsg(int MsgNum, str FDname)
        {

        if (MsgNum == 1){msgString =   "Valve is Closed, cycling Open."}
        if (MsgNum == 2){msgString =   "Valve is Open, cycling Closed."}
        if (MsgNum == 3){msgString =  "Valve is already Open, issuing open commands."}
        if (MsgNum == 4){msgString =  "Valve is already Closed, issuing open commands."}
        if (MsgNum == 5){msgString =   "Valve state must be determinate to run this method with MPS 750
active."}
        if (MsgNum == 6){msgString =   "Energizing Solenoid with 750 pneumatics secured."}
        if (MsgNum == 7){msgString =   "De-energizing Solenoid with 750 pneumatics secured."}
        if (MsgNum == 8){msgString =   "Valve cycle time is suspect due to failed indicators."}
        if (MsgNum == 9){msgString =   "Open power" & FDname & "is ON, should be OFF."}
        if (MsgNum == 10){msgString =  "Open power " & FDname & "is OFF, should be ON."}
        if (MsgNum == 11){msgString =  "Open indicator " & FDname & " is ON, should be OFF."}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
        if (MsgNum == 12){msgString =  "Open indicator " & FDname & " is OFF, should be ON."}
        if (MsgNum == 13){msgString =  "Closed indicator " & FDname & " is ON, should be OFF."}
        if (MsgNum == 14){msgString =  "Closed indicator " & FDname & " is OFF, should be ON."}
        if (MsgNum == 15){msgString =  "Closed power " & FDname & " is ON, should be OFF."}
        if (MsgNum == 16){msgString =  "Closed power " & FDname & " is OFF, should be ON."}
        if (MsgNum == 17){msgString =  "Open command(s) removed."}
        if (MsgNum == 18){msgString =  "Closed command(s) removed."}
        if (MsgNum == 19){msgString =  "Open and Closed commands removed."}
        if (MsgNum == 20){msgString =  "Valve cycle complete.  No errors."}
        if (MsgNum == 21){msgString =  "Valve cycle errors.  See page A for details."}
        if (MsgNum == 22){msgString =   "Removing close power "}
        if (MsgNum == 23){msgString =   "Removing open power "}
        if (MsgNum == 24){msgString =   "Applying open power. "}
        if (MsgNum == 25){msgString =   "Applying Close power. "}
        if (MsgNum == 26){msgString =   "Cannot run this method with 750 pneumatics secured."}
        if (MsgNum == 27){msgString =   "Aborting selected method."}
        if (MsgNum == 28){msgString =   "Activating SSME Reg A 750 PSI Pneumatics"}
        if (MsgNum == 29){msgString =   "Activating SSME Reg B 750 PSI Pneumatics"}
        if (MsgNum == 30){msgString =   "The location of the orbiter must be determinate to perform this
method."}
        if (MsgNum == 31){msgString =   "This component cannot be cycled in the OPF.
                                         The solenoids can be energized and deenergized using the appropriate
                                         methods.  Verify that the disconnect GSE is NOT installed and select
                                         the appropriate energize or deenergize solenoid method."}
        if (MsgNum == 32){msgString =   ""
        if (MsgNum == 33){msgString =   "Valve power " & FDname & " has been removed with bypassed
indicators."
        if (MsgNum == 34){msgString =   "Valve power " & FDname & " has been applied with bypassed
indicators."
        if (MsgNum == 35){msgString =   "Open RPC " & FDname & " is OFF, should be ON."
        if (MsgNum == 36){msgString =   "Open RPC " & FDname & " is ON, should be OFF."
        if (MsgNum == 37){msgString =   "Close RPC " & FDname & " is OFF, should be ON."
        if (MsgNum == 38){msgString =   "Close RPC " & FDname & " is ON, should be OFF."
        if (MsgNum == 39){msgString =   "Securing SSME Reg A isolation solenoid."
        if (MsgNum == 40){msgString =   "Securing SSME Reg B isolation solenoid."

return msgString;

        }
```

3.1.1.3.2. calculateCycleTimes(valveID, GMT_SIG, GMT_SW1, GMT_SW2, cycleType)

3.1.1.3.2.1. Descriptive Text

This method will be called from the derived class open/close valve cycle methods. The calling method will pass in GMT_SIG, GMT_SW1, GMT_SW2, and cycle type. The method will then calculate the sig to switch cycle time and print the result to the MPS Message display.

3.1.1.3.2.2. Sequence

```
calculateCycleTimes(GMT_SIG, GMT_SW1, GMT_SW2, CYCLE TYPE)
{

        swSwCycleTime = GMT_SW2 - GMT_SIG;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
        sigSwCycleTime = GMT_SW2-GMT_SW1;
}
{
        Print Message to MPS Message Page:
        Valve ID; Cycle Type; "switch to switch cycle time was" swSwCycleTime;"seconds"
        Valve ID; Cycle Type; "signal to switch cycle time was" sigSwCycleTime;"seconds"

}
return;
```

3.1.1.4. User Interfaces

3.1.1.5. Safety Requirements

3.1.2. MPS_SolValve_ORB

3.1.2.1. Description

MPS_SolValve_ORB is an abstract base class for the two-way and three-way solenoid valve types used by the MPS systems. This base class includes several types of MPS electrical solenoid valves (but does not apply to propellant valve actuation solenoids). Valves will be identified by an ID number (LV) and a description. Valves will have the capability to publish their state and be cycled to either the open (energized) or closed (deenergized) state by operator input or sequencer command.

3.1.2.1.1. Instance Index

Instance	Display Users	Other Users
	MPS, MPL	

3.1.2.1.2. Class Diagram

<insert class diagram here>

3.1.2.1.3. Attributes and Aggregations

Attribute	Data Type	Initial Value	Publish / Persistence	Client Access	Description
valveID	String				
valveDescription	String				
cmdState	MPS_ORB_SOL_STATE		PUB		
state	MPS_ORB_SOL_STATE		PUB		
RPCdelaytime	AM_SECS		PUB		

3.1.2.1.4. Methods Index

Operation	Description

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

sendOrbSolValveMsg(int MsgNum, str FDName):char *	This method displays generic MPS solenoid valve messages. The message displayed is determined by the integer passed parameter.

3.1.2.1.5. Associations

Association	Type	Class	Description

3.1.2.2. State Description

3.1.2.2.1. State Diagram

<insert state diagram here>

3.1.2.2.2. State Table

3.1.2.3. Operations

3.1.2.3.1. sendOrbSolValveMsg(int MsgNum, str FDName):char *

3.1.2.3.1.1. Sequence

```
sendOrbValveMsg(int MsgNum, str FDname)
        {

        if (MsgNum == 1){msgString =   "Valve is Closed, cycling Open."}
        if (MsgNum == 2){msgString =   "Valve is Open, cycling Closed."}
        if (MsgNum == 3){msgString =  "Valve is already Open, issuing open commands."}
        if (MsgNum == 4){msgString =  "Valve is already Closed, issuing open commands."}
        if (MsgNum == 5){msgString =   "Energizing Solenoid with 750 pneumatics secured."}
        if (MsgNum == 6){msgString =   "De-energizing Solenoid with 750 pneumatics secured."}
        if (MsgNum == 7){msgString =   "Valve cycle time is suspect due to failed indicators."}
        if (MsgNum == 8){msgString =   "Open power" & FDname & "is ON, should be OFF."}
        if (MsgNum == 9){msgString =  "Open power " & FDname & "is OFF, should be ON."}
        if (MsgNum == 10){msgString =  "Closed power " & FDname & " is ON, should be OFF."}
        if (MsgNum == 11){msgString =  "Closed power " & FDname & " is OFF, should be ON."}
        if (MsgNum == 12){msgString =  "Open command(s) removed."}
        if (MsgNum == 13){msgString =  "Closed command(s) removed."}
        if (MsgNum == 14){msgString =  "Valve cycle complete.  No errors."}
        if (MsgNum == 15){msgString =  "Valve cycle errors.  See page A for details."}
        if (MsgNum == 16){msgString =   "Removing close power "}
        if (MsgNum == 17){msgString =   "Removing open power "}
        if (MsgNum == 18){msgString =   "Applying open power. "}
        if (MsgNum == 19){msgString =   "Applying Close power. "}
        if (MsgNum == 20){msgString =   "Aborting selected method."}
        if (MsgNum == 21){msgString =   "Activating SSME Reg A 750 PSI Pneumatics"}
        if (MsgNum == 22){msgString =   "Activating SSME Reg B 750 PSI Pneumatics"}
        if (MsgNum == 23){msgString =   "Securing SSME Reg A isolation solenoid."
        if (MsgNum == 24){msgString =   "Securing SSME Reg B isolation solenoid."
        if (MsgNum == 25){msgString =   "Open RPC " & FDname & " is OFF, should be ON."
        if (MsgNum == 26){msgString =   "Open RPC " & FDname & " is ON, should be OFF."
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
        if (MsgNum == 27){msgString =   "Close RPC " & FDname & " is OFF, should be ON."
        if (MsgNum == 28){msgString =   "Close RPC " & FDname & " is ON, should be OFF."

return msgString;

        }
```

3.1.2.4. User Interfaces

3.1.2.5. Safety Requirements

3.1.3. MPS_Valve_ORB_Outboard_F&D

3.1.3.1. Description

MPS_Valve_ORB_Outboard_F&D is a bi-stable valve with 1 open and 1 close commands, 1 open and 1 close position indications, 1 open and 1 close power, and 1 close inhibit (ground open) command. The valve class has a close inhibit On pseudo which provides a status of this command for INS Cockpit Switch scan position determination. Functionally, the valve provides redundancy (with the inboard fill and drain valve) against inadvertent loss of propellant overboard (through the fill and drain line) during ascent. The Valve class inherits from the MPS_Valve_ORB Class. The valve has an association with the MPS_Helium_System class for status of the MPS 750 pneumatic system. For integrated operations, this valve class has an association with the MPS_System for determination of the cryo load stautus via a Cryo Flag pseudo.

3.1.3.1.1. Instance Index

Instance	Display User	Other Users
MPS_PV9	MPL	
MPS_PV11	MPS	

3.1.3.1.2. Class Diagram

<insert class diagram here>

3.1.3.1.3. Attributes and Aggregations

Attribute	Data Type	Initial Value	Publish / Persistence	Client Access	Description
openPosInd	DM_ON_OFF			R	
closePosInd	DM_ON_OFF			R	
openCmd	DS_ON_OFF			R,W	
closeCmd	DS_ON_OFF			R,W	
openPwr	DM_ON_OFF			R	
closePwr	DM_ON_OFF			R	
closeInhibit	DS_ON_OFF			R,W	

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

closeInhibitOnPseudo	PD_ON_OFF			R,W	
openSolCmdState	MPS_ORB_SOL_STATE		PUB	R	
closeSolCmdState	MPS_ORB_SOL_STATE		PUB	R	

3.1.3.1.4. Methods Index

Operation	**Description**
open()	This method performs necessary functions to cycle valve to the open position.
close()	This method performs necessary functions to cycle valve to the closed position.
energizeOpenSolenoid()	This method energizes open solenoid. Method includes a check of MPS pneumatics status, and present valve indicator state to determine consequence of energizing valve. The method then runs the appropriate open() method if the operator chooses to continue via prompt response.
deenergizeOpenSolenoid()	This method de-energizes opening solenoid. Method includes a check of MPS pneumatics status, and present valve indicator state to determine consequence of energizing valve. The method then runs the appropriate close() method if the operator chooses to continue via prompt response.
energizeCloseSolenoid()	This method energizes closing solenoid. Method includes a check of MPS pneumatics status, and present valve indicator state to determine consequence of energizing valve. The method then runs the appropriate close() method if the operator chooses to continue via prompt response..
deenergizeCloseSolenoid()	This method de-energizes closing solenoid. Method includes a check of MPS pneumatics status, and present valve indicator state to determine consequence of energizing valve. The method then runs the appropriate open() method if the operator chooses to continue via prompt response.
determineIndState()	This method will determine the position state of the valve based on all-of-all unbypassed open/close indicators and will return either: OPEN CLOSED TRANSITION ERROR
getIndState()	This method will return valve enumerated indicator state.
determineOpenSolCmdState()	This method will determine the command state of the opening solenoid valve based on valve power indicators and will return either: ENERGIZED DEENERGIZED ERROR

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

determineCloseSolCmdState()	This method will determine the command state of the opening solenoid valve based on valve power indicators and will return either: ENERGIZED DEENERGIZED ERROR
getOpenSolCmdState()	This method will return valve opening solenoid enumerated command state.
getCloseSolCmdState()	This method will return valve closing solenoid enumerated command state.
verifyCmdsOff()	This method will verify no commands issued.
setCloseConstraints()	This method will set-up Constraints for a closed valve.
setClosePosConstraints	This method will set-up constraints for a uncommanded closed bistable valve.
setOpenConstraints()	This method will set-up Constraints for an open valve.
setOpenPosConstraints	This method will set-up constraints for a uncommanded open bistable valve.
activateConstraints()	This method will activate Constraints for either the close or open positions when in standby mode (uncommanded present state).
inhibitConstraints()	This method will inhibit Constraint notification during valve cycle operations.
autoCycle()	This method will perform valve open/close autocycle operations.
verifyTminus3BT4()//	This method does a byte test 4 of valve commands to verify T-3 hour configuration.
verifyTminus6BT4()//	This method does a byte test 4 of valve commands to verify T-6 hour configuration.
setCloseInhibitOn()//	Phase 3 method required to support load.
setCloseInhibitOff()//	Phase 3 method required to support load.
monitorForInitialCloseTime()//	Phase 3 method required to support LCC preplanned contingency.
stopMonitorForInitialCloseTime()//	Phase 3 method required to support LCC preplanned contingency.
getInitialCloseTime()//	Phase 3 method required to support LCC preplanned contingency.
determineState()//	Phase 3 method to determine over-all state of valve based on command state and indicator state (for use in load displays). Possible states are: OPEN CLOSED OPEN_ERROR

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

	CLOSED_ERROR TRANSITION
setValveCloseCOnstraints()//	Phase 3 method for constraint initiation for valve cycle at approx T-45 seconds.
getState()//	This method will return valve over-all state. Phase 3 method.
determineCmdState()//	Phase 3 method to determine over-all cmd state of valve based on open/close solenoid command states (for use in load displays). Possible states are: COMMANDED_OPEN COMMANDED_CLOSED UNCOMMANDED COMMANDED_SNUBBED ERROR
getCmdState()//	This method will return valve over-all cmd state. Phase 3 method.

3.1.3.1.5. Associations

Association	Type	Class	Description
MPS_Hydrogen_SubSystem			LH2CryoFlag status.
MPS_Oxygen_Subsystem			LO2CryoFlag status.
MPS_Helium_SubSystem			MPS 750 pneumatic status.

3.1.3.2. State Description

3.1.3.2.1. State Diagram

<insert state diagram here>

3.1.3.2.2. State Table

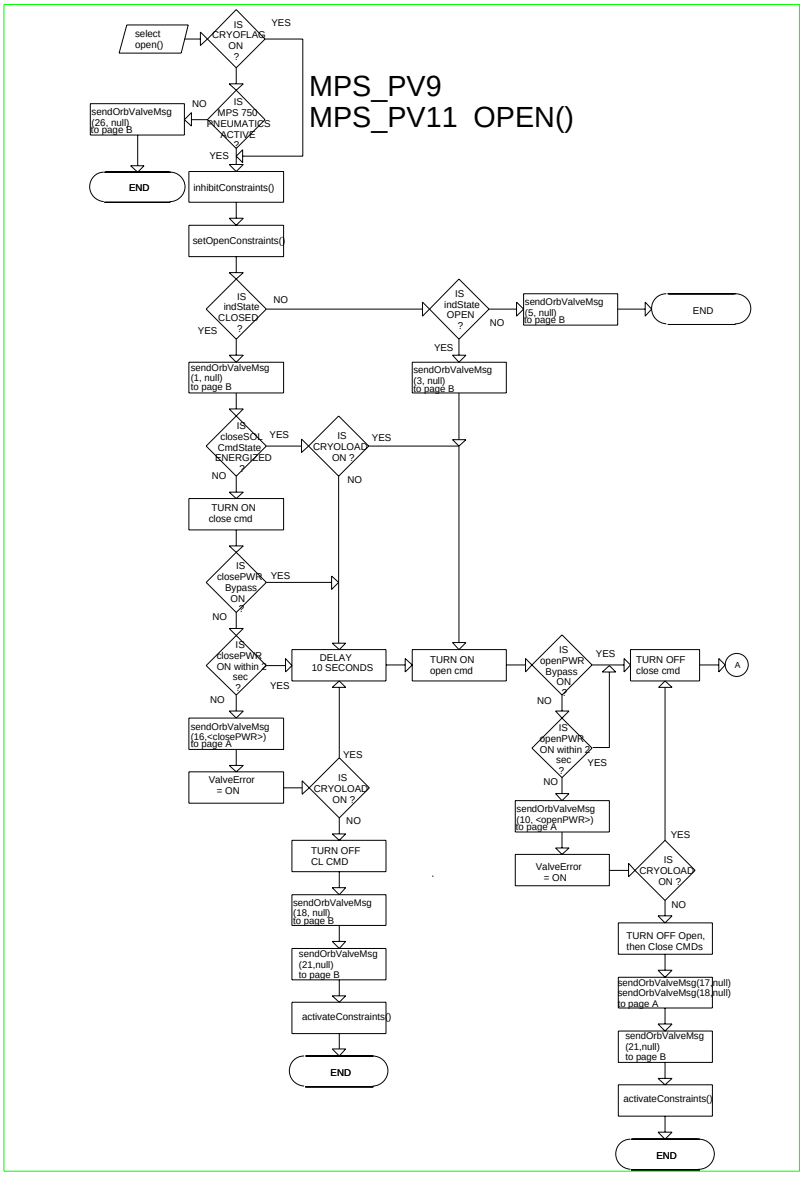
3.1.3.3. Operations

3.1.3.3.1. open()

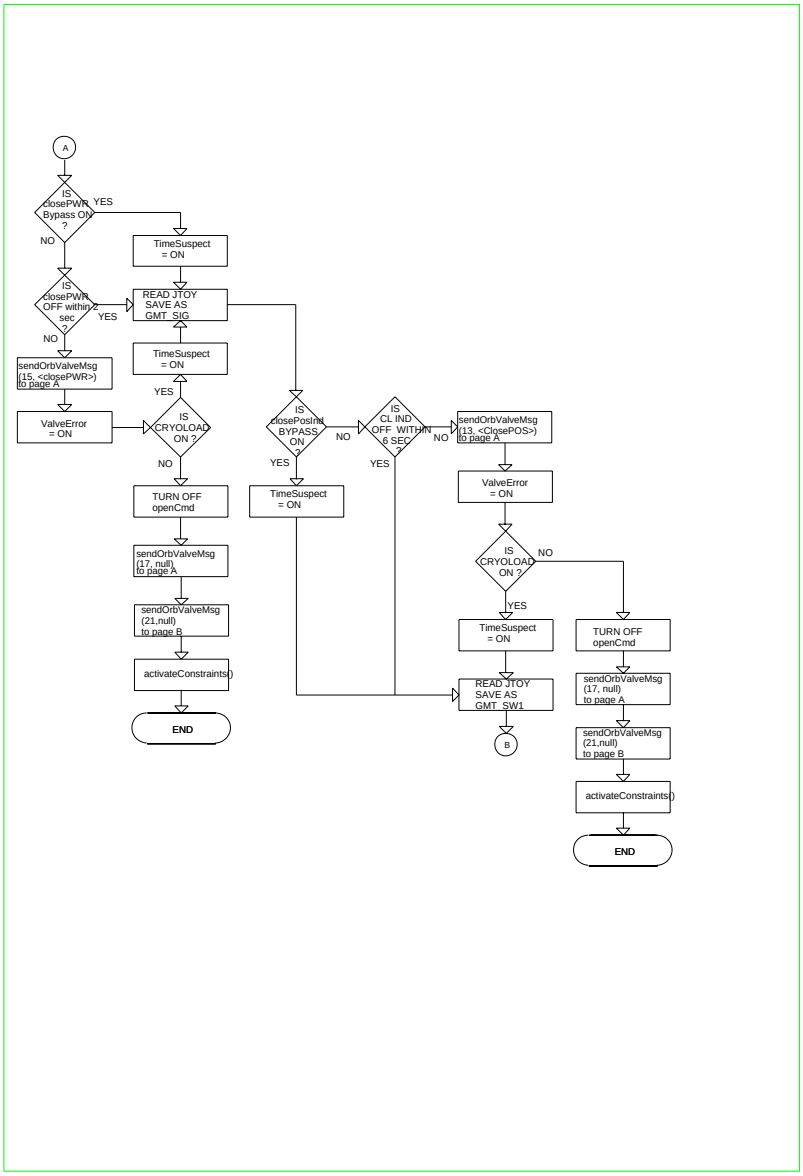
3.1.3.3.1.1. Descriptive Text

This method performs cycling of the bistable LO2 and LH2 outboard fill and drain valves to the open position. The flow chart provides the necessary logic for valve cycles, including checks for snubbing pressure, proper response of power indications, and proper response of position indications. The valve cycles in a fail safe mode and 'wimps-out' (all command removed) at the first indication of hardware failure, unless the orbiter is in cryogenic load. If in cryo load, the valve cycles in a 'damn-the-torpedos' mode and completes all steps necessary to achieve the valve cycle, regardless of failures which are detected during the cycle. Valve signal to switch and switch to switch timing is also calculated. ValveError and TimeSuspect are initialized to OFF and Valve cycle times are set to 0 prior to starting valve cycle methods. If ValveError gets set ON during method run, cycle time FD data health should be set to invalid to aid in identifying suspect cycle data in trend analysis.

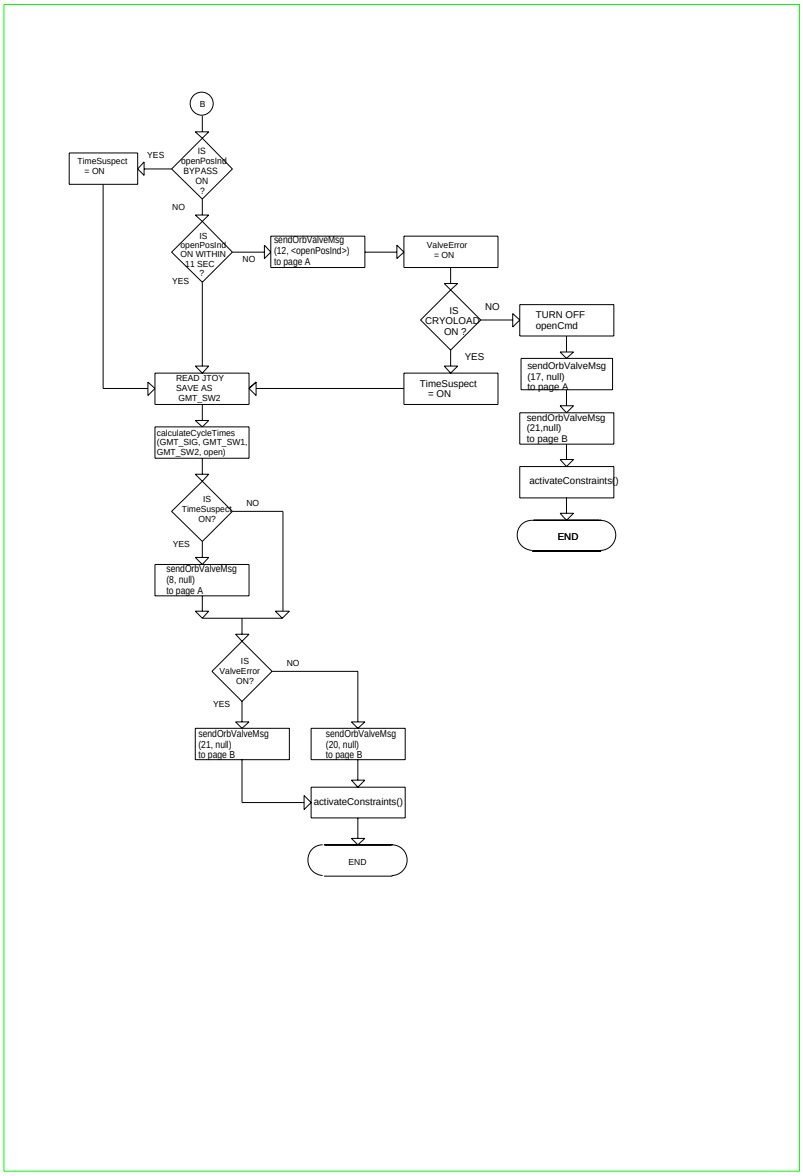
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

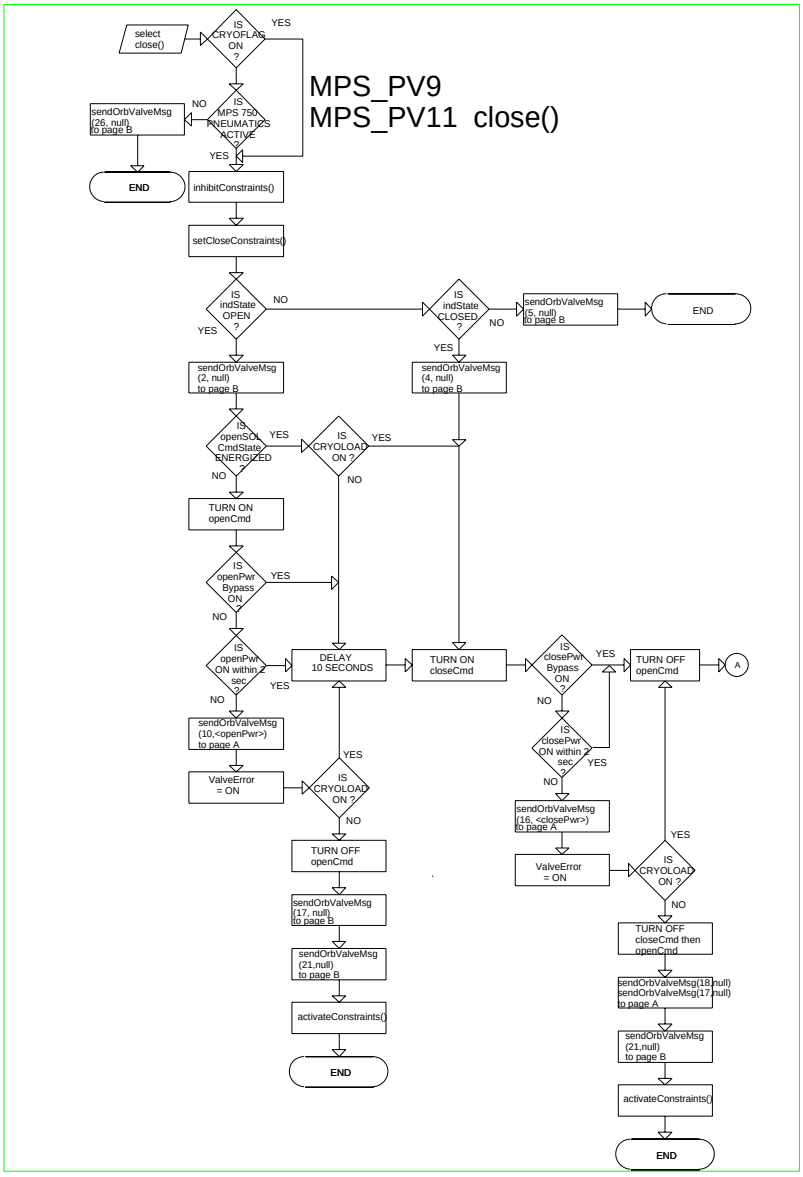


3.1.3.3.2. close()

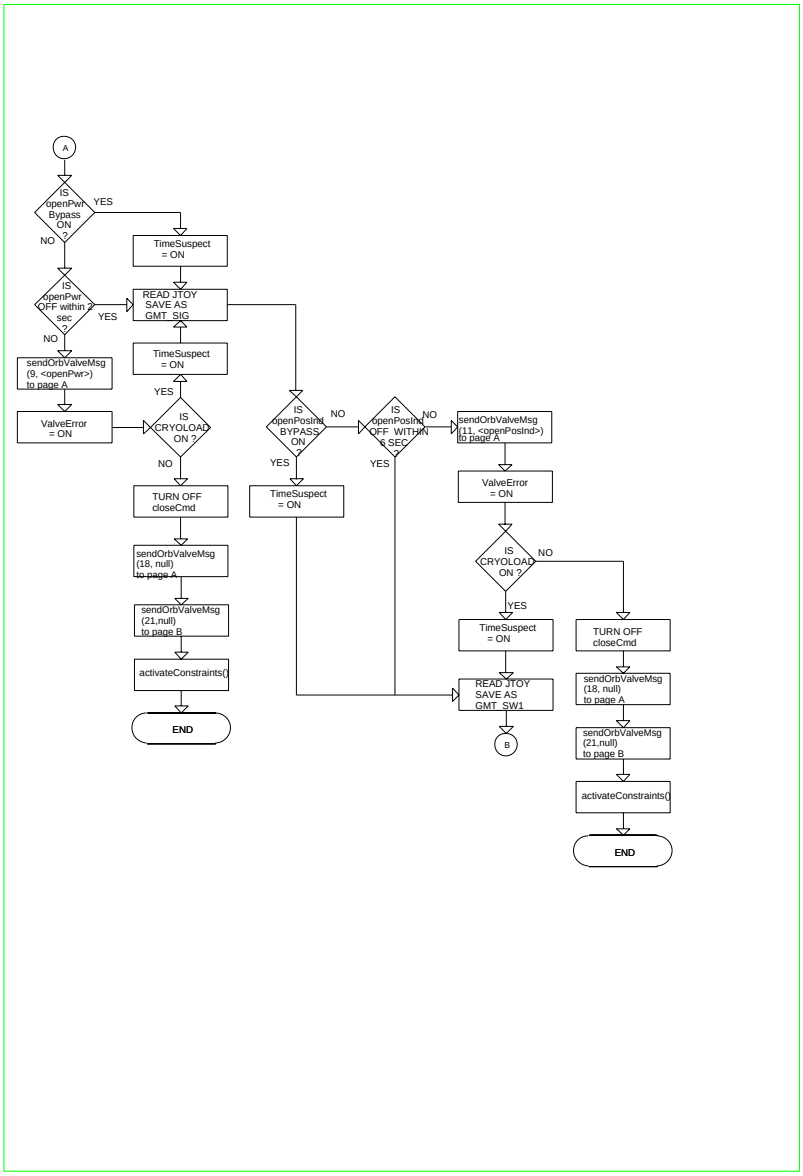
3.1.3.3.2.1. Descriptive Text

This method performs cycling of the bistable LO2 and LH2 outboard fill and drain valves to the closed position. The flow chart provides the necessary logic for valve cycles, including checks for snubbing pressure, proper response of power indications, and proper response of position indications. The valve cycles in a fail safe mode and 'wimps-out' (all command removed) at the first indication of hardware failure, unless the orbiter is in cryogenic load. If in cryo load, the valve cycles in a 'damn-the-torpedoes' mode and completes all steps necessary to achieve the valve cycle, regardless of failures which are detected during the cycle. Valve signal to switch and switch to switch timing is also calculated. ValveError and TimeSuspect are initialized to OFF and Valve cycle times are set to 0 prior to starting valve cycle methods. If ValveError gets set ON during method run, cycle time FD data health should be set to invalid to aid in identifying suspect cycle data in trend analysis.

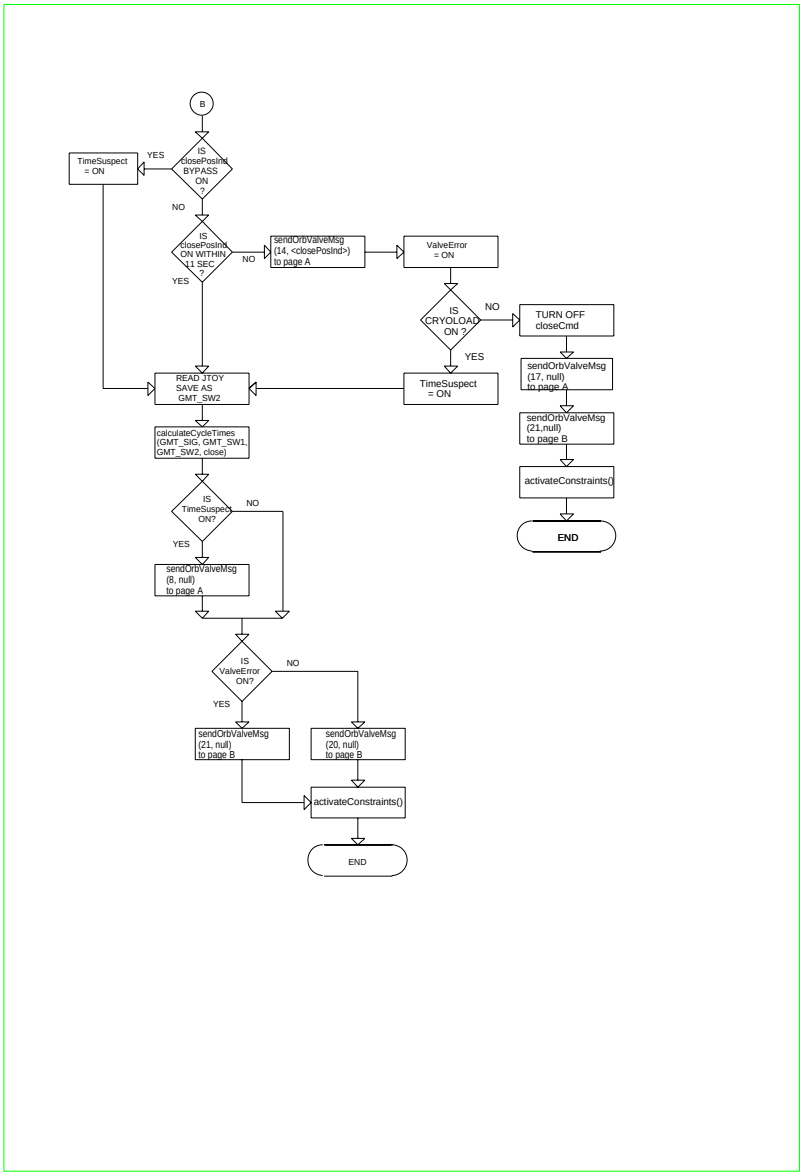
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

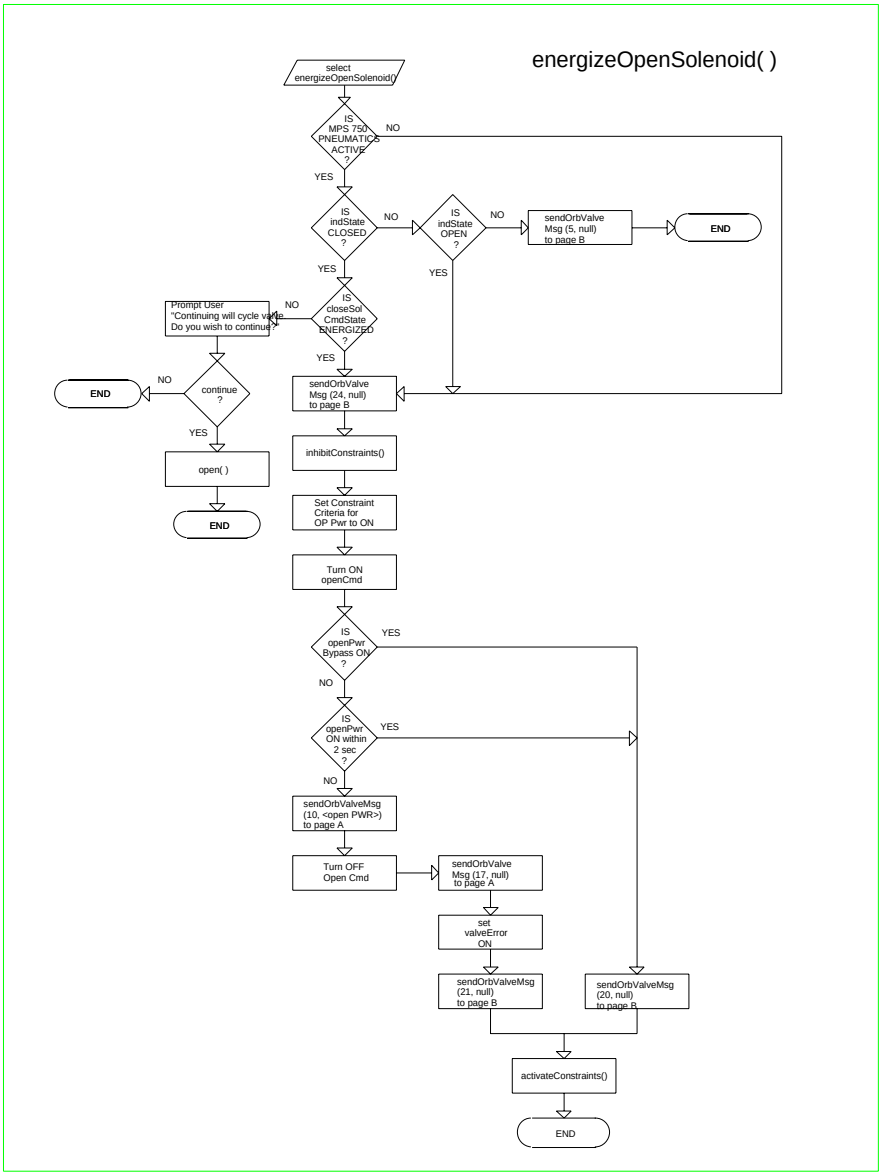


3.1.3.3.3. energizeOpenSolenoid()

3.1.3.3.3.1. Descriptive Text

This method performs the necessary checks to energize the outboard fill and drain valve opening solenoid. The sequence as identified on the following flow chart checks for valve present position, status of MPS 750 pneumatics, and result of energizing the opening solenoid if pneumatics is active. If a valve cycle will occur, the operator is prompted to provide input to continue (Yes/No option), and if a cycle is intended, the close() method is initiated. Constraints are maintained during this operation, with failure messages being generated on valve power indications.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

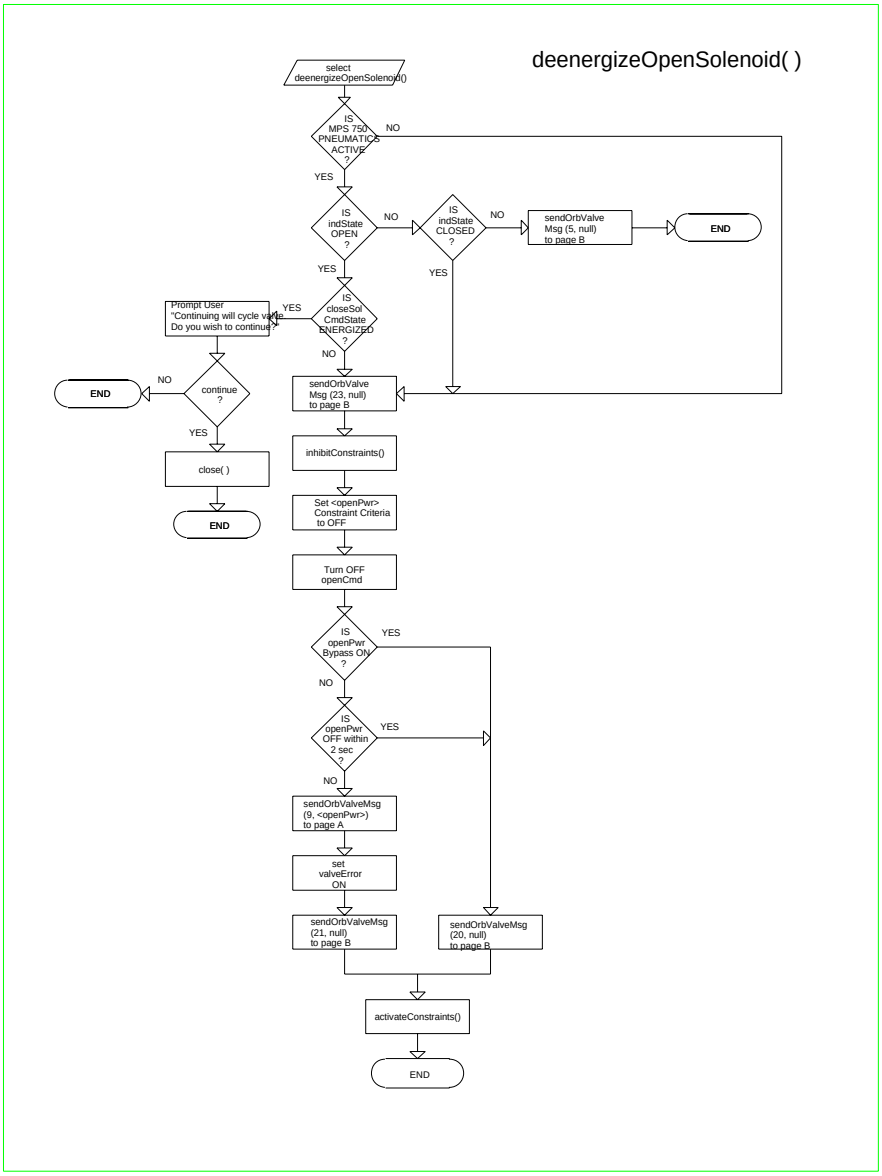


3.1.3.3.4. deenergizeOpenSolenoid()

3.1.3.3.4.1. Descriptive Text

This method performs the necessary checks to deenergize the outboard fill and drain valve opening solenoid. The sequence as identified on the following flow chart checks for valve present position, status of MPS 750 pneumatics, and result of deenergizing the opening solenoid if pneumatics is active. If a valve cycle will occur, the operator is prompted to provide input to continue (Yes/No option), and if a cycle is intended, the close() method is initiated. Constraints are maintained during this operation, with failure messages being generated on valve power indications.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

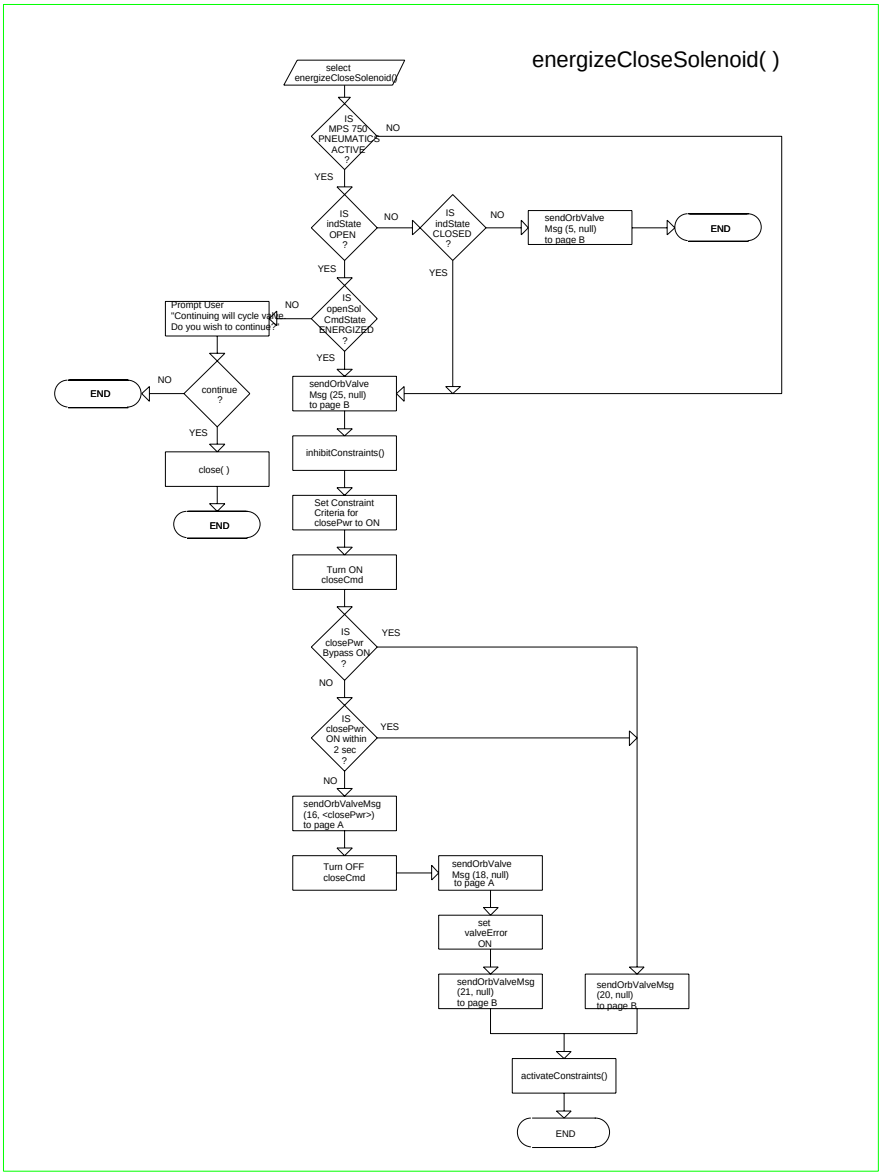


3.1.3.3.5. energizeCloseSolenoid()

3.1.3.3.5.1. Descriptive Text

This method performs the necessary checks to energize the outboard fill and drain valve closing solenoid. The sequence as identified on the following flow chart checks for valve present position, status of MPS 750 pneumatics, and result of energizing the closing solenoid if pneumatics is active. If a valve cycle will occur, the operator is prompted to provide input to continue (Yes/No option), and if a cycle is intended, the close() method is initiated. Constraints are maintained during this operation, with failure messages being generated on valve power indications.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

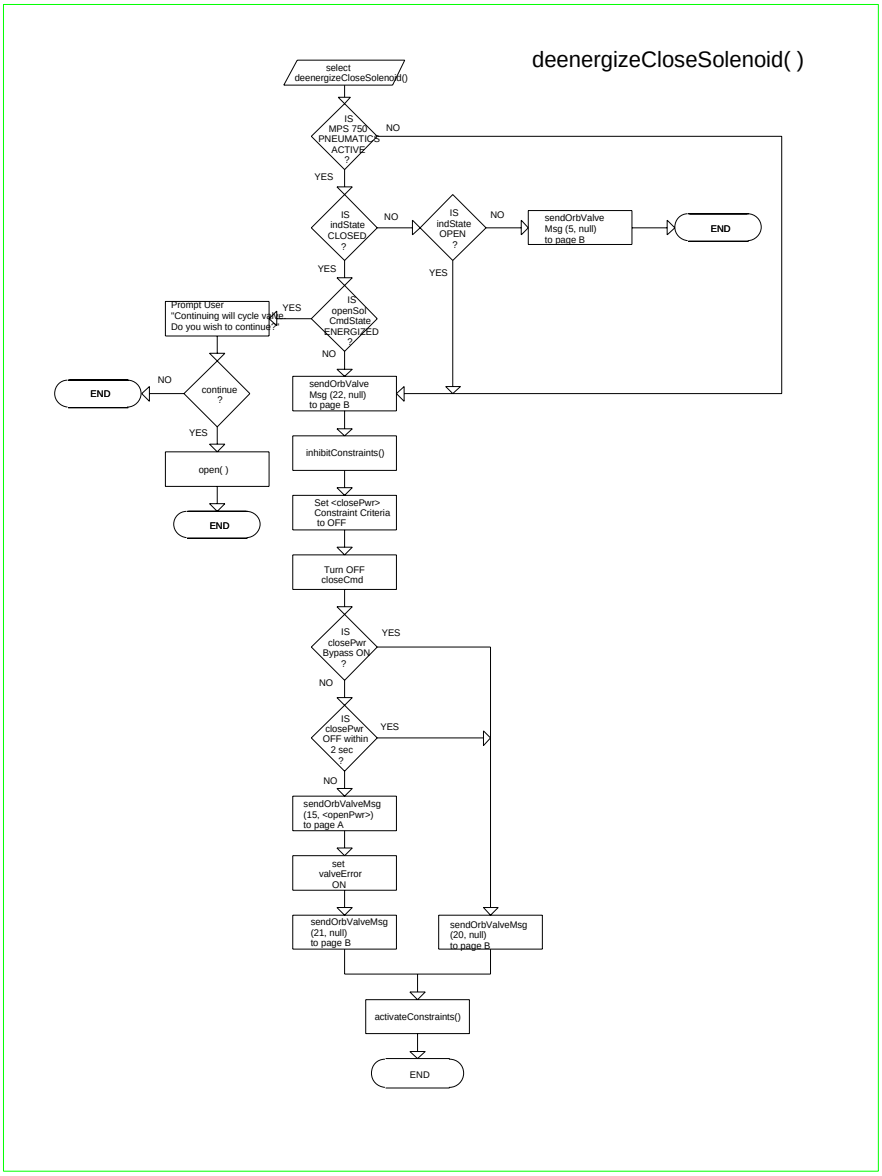


3.1.3.3.6. deenergizeCloseSolenoid()

3.1.3.3.6.1. Descriptive Text

This method performs the necessary checks to deenergize the outboard fill and drain valve closing solenoid. The sequence as identified on the following flow chart checks for valve present position, status of MPS 750 pneumatics, and result of deenergizing the closing solenoid if pneumatics is active. If a valve cycle will occur, the operator is prompted to provide input to continue (Yes/No option), and if a cycle is intended, the open() method is initiated. Constraints are maintained during this operation, with failure messages being generated on valve power indications.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



3.1.3.3.7. determineOpenSolCmdState()

3.1.3.3.7.1. Descriptive Text

This operation is used to determine the open solenoid's commanded state based on the open power indicator and its associated bypass. The return value will be an enumerated type with possible values of ENERGIZED, DEENERGIZED and ERROR.

3.1.3.3.7.2. Effectivity

Operates on change data.

3.1.3.3.7.3. Sequence

Case:

<openPwr> == OFF

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

<openPwr>Byp == OFF
openSolCmdState = DEENERGIZED;

<openPwr> == ON
<openPwr>Byp == OFF
openSolCmdState = ENERGIZED;

<openPwr>Byp==On
openSolCmdState = ERROR;

End Case:

3.1.3.3.8. getOpenSolCmdState()

3.1.3.3.8.1. Descriptive Text

This method will return the Enumerated Command State of the valve Opening Solenoid.

3.1.3.3.8.2. Sequence

```
getOpenSolCmdState()
{
    return OpenSolCmdState;
}
```

3.1.3.3.9. determineCloseSolCmdState()

3.1.3.3.9.1. Descriptive Text

This method will determine the close solenoid's commanded state based on the two close RPCs, the close power, and the associated bypasses. The return value will be an enumerated
type with possible values of ENERGIZED, DEENERGIZED and ERROR.

3.1.3.3.9.2. Effectivity

Operates on change data.

3.1.3.3.9.3. Sequence

Case:

<closePwr> == OFF
<closePwr>Byp == OFF
closeSolCmdState = DEENERGIZED;

<closePwr> == ON
<closePwr>Byp == OFF
closeSolCmdState = ENERGIZED;

<closePwr>Byp == ON
closeSolCmdState = ERROR;

End Case:

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.3.3.10. getCloseSolCmdState()

3.1.3.3.10.1. Descriptive Text

This method will return the Enumerated Command State of the valve Closing Solenoid.

3.1.3.3.10.2. Sequence

```
getCloseSolCmdState()
{
        return CloseSolCmdState;
}
```

3.1.3.3.11. determineIndState()

3.1.3.3.11.1. Descriptive Text

This method will determine the valve's position based on its position indicators and associated indicator bypasses. The return value will be an enumerated type with possible values of OPEN, CLOSED, TRANSITION, and ERROR.

3.1.3.3.11.2. Effectivity

3.1.3.3.11.3. Sequence

Case:

```
        <openPosInd> == OFF
        <openPosInd>Byp == OFF
        <closePosInd> == ON
        <closePosInd>Byp == OFF
                        indState = CLOSED;

        <openPosInd> == OFF
        <openPosInd>Byp == OFF
        <closePosInd>Byp == ON
                        indState = CLOSED;

        <openPosInd>Byp == ON
        <closePosIndB> == ON
        <closePosInd>Byp == OFF
                        indState = CLOSED;

        <openPosInd> == ON
        <openPosInd>Byp == OFF
        <closePosInd> == OFF
        <closePosInd>Byp == OFF
                        indState = OPEN;

        <openPosInd> == ON
        <openPosInd>Byp == OFF
        <closePosInd>Byp == ON
                        indState = OPEN;

        <openPosInd>Byp == ON
        <closePosInd> == OFF
        <closePosInd>Byp == OFF
                        indState = OPEN;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
        <openPosInd> == OFF
        <openPosInd>Byp == OFF
        <closePosInd> == OFF
        <closePosInd>Byp == OFF
                        indState = TRANSITION;


  CASE ELSE
                        indState = ERROR;

End Case:
```

3.1.3.3.12. getIndState()

3.1.3.3.12.1. Descriptive Text

This method will return the Enumerated Indicator State of the valve

3.1.3.3.12.2. Sequence

```
getIndState()
{
        return IndState;
}
```

3.1.3.3.13. verifyCmdsOff()

3.1.3.3.13.1. Descriptive Text

This function will verify that both the open and close commands are OFF.

3.1.3.3.13.2. Sequence

```
verifyCommandsOff()
{
        open command status = byteTest4 <Open Command>
                verify  open command status == OFF
                else send message ",valveID, open command;<openCMD>; " is ON, should be OFF" to Page A

        close command status = byteTest4 <Close Command>
                verify close command status == OFF
                else print message ",valveID, close command;<closeCMD>; " is ON, should be OFF" to Page A

        {
        verify  open command status && close command status == OFF

        return TRUE
        }
        ELSE return FALSE
}
```

3.1.3.3.14. setCloseConstraints()

3.1.3.3.14.1. Descriptive Text

This method changes the constraint criteria of the measurements associated with this valve to those expected with the valve in the closed position.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.1.3.3.14.2. Sequence

setCloseConstraints()
{
CHANGE <openPwr> CONSTRAINT CRITERIA TO OFF;

CHANGE <openPosInd> CONSTRAINT CRITERIA TO OFF;

CHANGE <closePwr> CONSTRAINT CRITERIA TO ON;

CHANGE <closePosInd> CONSTRAINT CRITERIA TO ON;
}

3.1.3.3.15. setClosePosConstraints()

3.1.3.3.15.1. Descriptive Text

This method changes the constraint criteria of the measurements associated with this valve to those expected with the valve in the closed position with no power (or pneumatics) applied.

3.1.3.3.15.2. Sequence

setClosePosConstraints()
{
CHANGE <openPosInd> CONSTRAINT CRITERIA TO OFF;

CHANGE <closePosInd> CONSTRAINT CRITERIA TO ON;

CHANGE <openPwr> CONSTRAINT CRITERIA to OFF;

CHANGE <closePwr> CONSTRAINT CRITERIA to OFF;

}

3.1.3.3.16. setOpenConstraints()

3.1.3.3.16.1. Descriptive Text

This method changes the constraint criteria of the measurements associated with this valve to those expected with the valve in the open position.

3.1.3.3.16.2. Sequence

setOpenConstraints()
{
CHANGE <openPwr> CONSTRAINT CRITERIA TO ON;

CHANGE <openPosInd> CONSTRAINT CRITERIA TO ON;

CHANGE <closePwr> CONSTRAINT CRITERIA TO OFF;

CHANGE <closePosInd> CONSTRAINT CRITERIA TO OFF;

}

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.3.3.17. setOpenPosConstraints()

3.1.3.3.17.1. Description

This method changes the constraint criteria of the measurements associated with this valve to those expected with the valve in the open position with no power (or pneumatics) applied.

3.1.3.3.17.2. Sequence

```
setOpenPosConstraints()
        {
                CHANGE <openPosInd>  CONSTRAINT CRITERIA TO ON;

                  CHANGE <closePosInd> CONSTRAINT CRITERIA TO OFF;

                  CHANGE <openPwr> CONSTRAINT CRITERIA to OFF;

                  CHANGE <closePwr> CONSTRAINT CRITERIA to OFF;

        }
```

3.1.3.3.18. activateConstraints()

3.1.3.3.18.1. Descriptive Text

This operation activates constraint monitoring on valve position and power indicators.

3.1.3.3.18.2. Sequence

```
activateConstraints()
{
        If (<openPwr>.Byp == OFF)
        Initiate Constraint Monitoring on <openPwr>
        }
        {
        If (<openPosInd>.Byp == OFF)
        Initiate Constraint Monitoring on <openPosInd>
        }
        {
        If (<closePosInd>.Byp == OFF)
        Initiate Constraint Monitoring on <closePosInd>
        {
        If (<closePwr>.Byp == OFF)
        Initiate Constraint Monitoring on <closePwr>

}
```

3.1.3.3.19. inhibitConstraints()

3.1.3.3.19.1. Descriptive Text

This operation de-activates constraint monitoring on valve position and power indicators.

3.1.3.3.19.2. Sequence

```
inhibitConstraints()
{
        Inhibit Constraint Monitoring on
                <openPwr>
                <closePwr>
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

<openPosInd>
<closePosInd>
}

3.1.3.3.20. autoCycle()

3.1.3.3.20.1. Descriptive Text

This method will check the present state of the valve, then cycle the valve to the opposite state and back to the original state using the open() and close() methods to perform an autocycle check

3.1.3.3.20.2. Sequence

{

If instance IndState == CLOSED
open();
If valveCycleError is ON, set autoValveCheckoutError = ON
close();
If valveCycleError is ON, set autoValveCheckoutError = ON

If instance IndState == OPEN
close();
If valveCycleError is ON, set autoValveCheckoutError = ON
open();
If valveCycleError is ON, set autoValveCheckoutError = ON

Return autoValveCheckoutError;

}

3.1.3.3.21. determineState()//Phase 3 Method

3.1.3.3.21.1. Description

This method determines over-all valve state of type MPS_ORB_VALVE_STATE based on command state and indicator state with possible values of OPEN, CLOSED, OPEN_ERROR, CLOSED_ERROR and TRANSITION.

3.1.3.3.22. getState()//Phase 3 Method

3.1.3.3.22.1. Description

Phase 3 method returns over-all state of valvefrom determineState(). Possible states are:
OPEN
CLOSED
OPEN_ERROR
CLOSED_ERROR
TRANSITION

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.3.3.23. determineCmdState()//Phase 3 Method

3.1.3.3.23.1. Description

3.1.3.3.24. getCmdState()//Phase 3 Method

3.1.3.3.24.1. Description

3.1.3.3.25. verifyTminus3BT4()//Phase 3 Method

3.1.3.3.25.1. Description

This method performs a byte test 4 of fill and drain valve commands to verify they are in a proper T-3 hour configuration. For both instances of this class, open commands should be ON, close commands should be OFF.

3.1.3.3.25.2. Sequence

begin sequence;

```
set commanderrorcheck OFF
{
perform ByteTest4 of <openCMD>
        verify ByteTest4 = ON
        else send message to page A
                "Valve" <openCMD> "is not in T-3 hour configuration."
                and set commanderrorcheck ON
}
{
perform ByteTest4 of <closeCMD>
        verify ByteTest4 = OFF
        else send message to page A
                "Valve" <closeCMD> "is not in T-3 hour configuration."
                and set commanderrorcheck ON
}
return commanderrorcheck

end sequence;
```

3.1.3.3.26. verifyTminus6BT4()//Phase 3

3.1.3.3.26.1. Description

This method performs a byte test 4 of fill and drain valve commands to verify they are in a proper T-6 hour configuration. For both instances of this class, open commands should be OFF, close commands should be ON.

3.1.3.3.26.2. Sequence

begin sequence;

```
set commanderrorcheck OFF
{
perform ByteTest4 of <openCMD>
        verify ByteTest4 = OFF
        else send message to page A
                "Valve" <openCMD> "is not in T-6 hour configuration."
                and set commanderrorcheck ON
}
{
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
perform ByteTest4 of <closeCMD>
        verify ByteTest4 = ON
        else send message to page A
                "Valve" <closeCMD> "is not in T-6 hour configuration."
                and set commanderrorcheck ON
}
return commanderrorcheck

end sequence;
```

3.1.3.3.27. setCloseInhibitOn()//Phase 3

3.1.3.3.28. setCloseInhibitOff()//Phase 3

3.1.3.3.29. setValveCloseConstraints()//Phase 3

3.1.3.3.30. monitorForInitialCloseTime()//Phase 3

3.1.3.3.31. stopMonitorForInitialCloseTime()//Phase 3

3.1.3.3.32. getInitialCloseTime()//Phase 3

3.1.3.4. Safety Requirements

3.1.4. MPS_Valve_ORB_Inboard_F&D

3.1.4.1. Description

MPS_Valve_ORB_Inboard_F&D is a bi-stable valve with 2 open and 1 close commands, 1 open and 1 close position indications, 1 open and 1 close power, and 1 close inhibit (ground open) command. The valve class has a close inhibit On pseudo which provides a status of this command for INS Cockpit Switch scan position determination. Functionally, the valve provides redundancy (with the outboard fill and drain valve) against inadvertent loss of propellant overboard (through the fill and drain line) during ascent. The Valve class inherits from the MPS_Valve_ORB Class. The valve has an association with the MPS_Helium_System class for status of the MPS 750 pneumatic system. For integrated operations, this valve class has an association with the MPS_System for determination of the cryo load stautus via a Cryo Flag pseudo. One instance of this valve class exists for both the MPS orbiter liquid hydrogen and oxygen systems.

3.1.4.1.1. Instance Index

Instance	Display User	**Other Users**
MPS_PV10	MPL	
MPS_PV12	MPS	

3.1.4.1.2. Class Diagram

<insert class diagram here>

3.1.4.1.3. Attributes and Aggregations

Attribute	**Data Type**	**Initial Value**	**Publish / Persistence**	**Client Access**	**Description**

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

openPosInd	DM_ON_OFF			R	
closePosInd	DM_ON_OFF			R	
openCmdA	DS_ON_OFF			R,W	
openCmdB	DS_ON_OFF			R,W	
closeCmd	DS_ON_OFF			R,W	
openPwr	DM_ON_OFF			R	
closePwr	DM_ON_OFF			R	
closeInhibit	DS_ON_OFF			R,W	
closeInhibitOnPseudo	ENUM		PUB	R	
openSolCmdState	MPS_SOL_VALVE_CMD_STATE		PUB	R	
closeSolCmdState	MPS_SOL_VALVE_CMD_STATE		PUB	R	

3.1.4.1.4. Methods Index

Operation	**Description**
open()	This method performs necessary functions to cycle valve to the open position.
close()	This method performs necessary functions to cycle valve to the closed position.
energizeOpenSolenoid()	This method energizes open solenoid. Method includes a check of MPS pneumatics status, and present valve indicator state to determine consequence of energizing valve. The method then runs the appropriate open() method if the operator chooses to continue via prompt response.
deenergizeOpenSolenoid()	This method de-energizes opening solenoid. Method includes a check of MPS pneumatics status, and present valve indicator state to determine consequence of energizing valve. The method then runs the appropriate close() method if the operator chooses to continue via prompt response.
energizeCloseSolenoid()	This method energizes closing solenoid. Method includes a check of MPS pneumatics status, and present valve indicator state to determine consequence of energizing valve. The method then runs the appropriate close() method if the operator chooses to continue via prompt response.
deenergizeCloseSolenoid()	This method de-energizes closing solenoid. Method includes a check of MPS pneumatics status, and present valve indicator state to determine

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

	consequence of energizing valve. The method then runs the appropriate open() method if the operator chooses to continue via prompt response.
determineIndState()	This method will determine the position state of the valve based on all-of-all unbypassed open/close indicators and will return either: OPEN CLOSED TRANSITION ERROR
determineOpenSolCmdState()	This method will determine the command state of the opening solenoid valve based on valve power indicators and will return either: ENERGIZED DEENERGIZED ERROR
determineCloseSolCmdState()	This method will determine the command state of the opening solenoid valve based on valve power indicators and will return either: ENERGIZED DEENERGIZED ERROR
getIndState()	This method will return valve enumerated indicator state.
getOpenSolCmdState()	This method will return the valve opening solenoid Enumerated command state.
getCloseSolCmdState()	This method will return the valve closing solenoid Enumerated command state.
verifyCmdsOff()	This method will verify no command issued (for 750 activation).
setCloseConstraints()	This method will set-up Constraints for a closed valve.
setClosePosConstraints()	This method will set-up Constraints for a closed bistable uncommanded valve.
setOpenConstraints()	This method will set-up Constraints for an open valve.
setOpenPosConstraints	This method will set-up Constraints for a open bistable uncommanded valve.
activateConstraints()	This method will activate Constraints for either the close or open positions when in standby mode (uncommanded present state).
inhibitConstraints()	This method will inhibit Constraint notification during valve cycle operations.
autoCycle()	This method will perform valve open/close autocycle operations.
verifyTminus3BT4//Phase 3	This method does a byte test 4 of valve commands to verify T-3 hour configuration.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

verifyTminus6BT4//Phase 3	This method does a byte test 4 of valve commands to verify T-6 hour configuration.
setCloseInhibitOn()//Phase 3	VAB/PAD method required to support load.
setCloseInhibitOff()//Phase 3	VAB/PAD method required to support load.
setValveCloseConstraints()//Phase 3	VAB/PAD method required to support load.
monitorForInitialCloseTime()//Phase 3	VAB/PAD method required to support LCC preplanned contingency.
stopMonitorForInitialCloseTime()//Phase 3	VAB/PAD method required to support LCC preplanned contingency.
getInitialCloseTime()//Phase 3	VAB/PAD method required to support LCC preplanned contingency.

3.1.4.1.5. Associations

Association	Type	Class	Description
MPS_Hydrogen_SubSystem			LH2CryoFlag status.
MPS_Oxygen_Subsystem			LO2CryoFlag status.
MPS_Helium_SubSystem			MPS 750 pneumatic status.

3.1.4.2. State Description

3.1.4.2.1. State Diagram

<insert state diagram here>

3.1.4.2.2. State Table

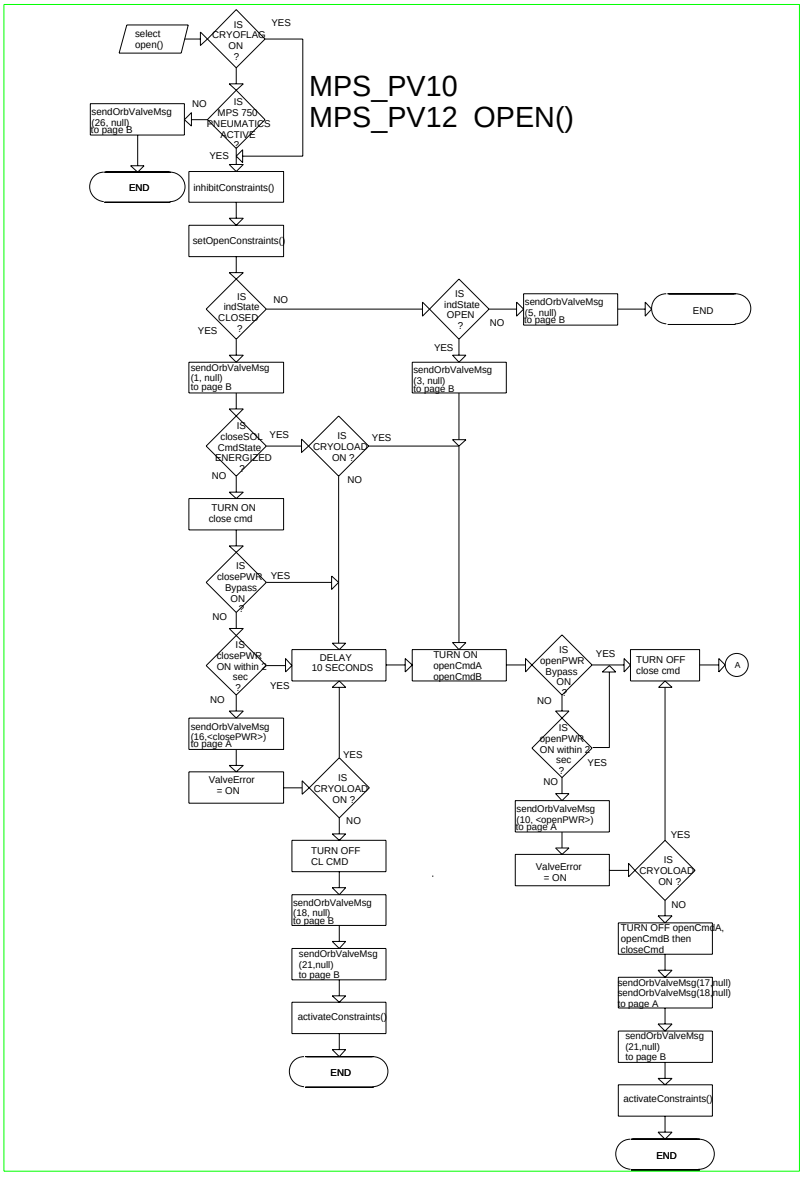
3.1.4.3. Operations

3.1.4.3.1. open()

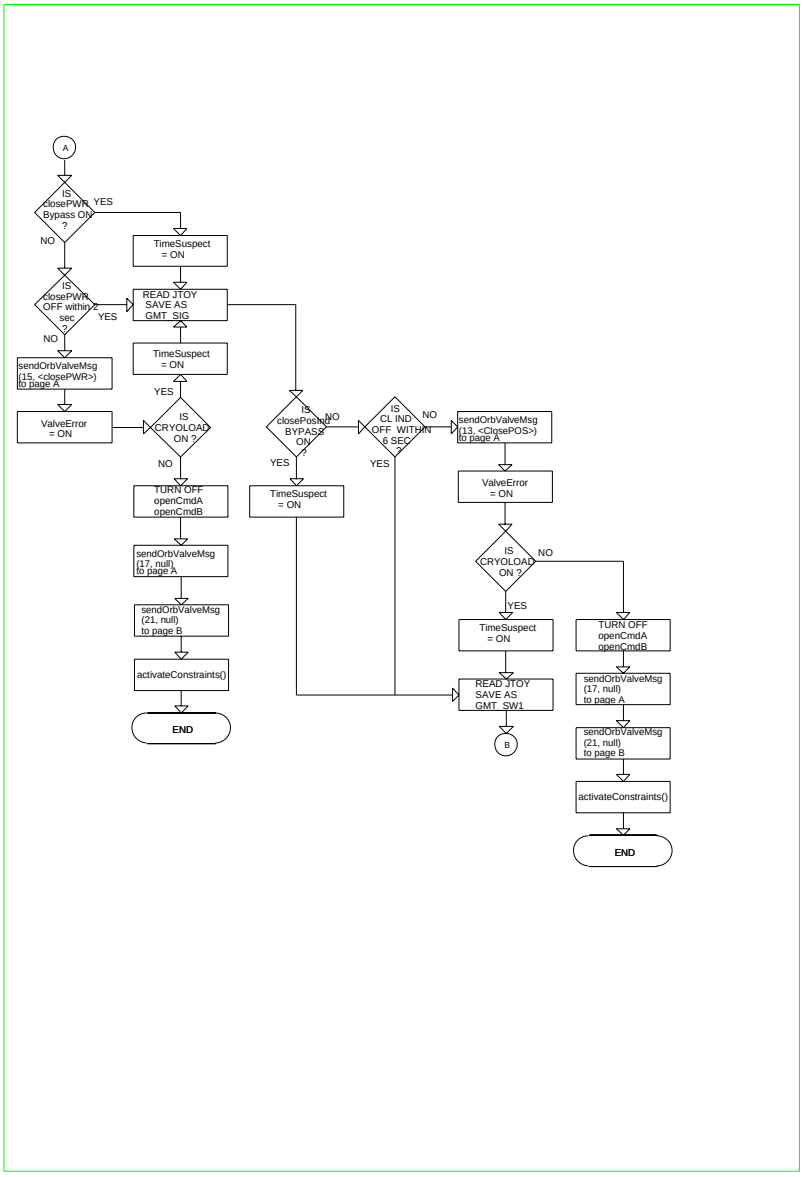
3.1.4.3.1.1. Descriptive Text

This method performs cycling of the bistable LO2 and LH2 inboard fill and drain valves to the open position. The flow chart provides the necessary logic for valve cycles, including checks for snubbing pressure, proper response of power indications, and proper response of position indications. The valve cycles in a fail safe mode and 'wimps-out' (all command removed) at the first indication of hardware failure, unless the orbiter is in cryogenic load. If in cryo load, the valve cycles in a 'damn-the-torpedos' mode and completes all steps necessary to achieve the valve cycle, regardless of failures which are detected during the cycle. Valve signal to switch and switch to switch timing is also calculated. ValveError and TimeSuspect are initialized to OFF and Valve cycle times are set to 0 prior to starting valve cycle methods. If ValveError gets set ON during method run, cycle time FD data health should be set to invalid to aid in identifying suspect cycle data in trend analysis.

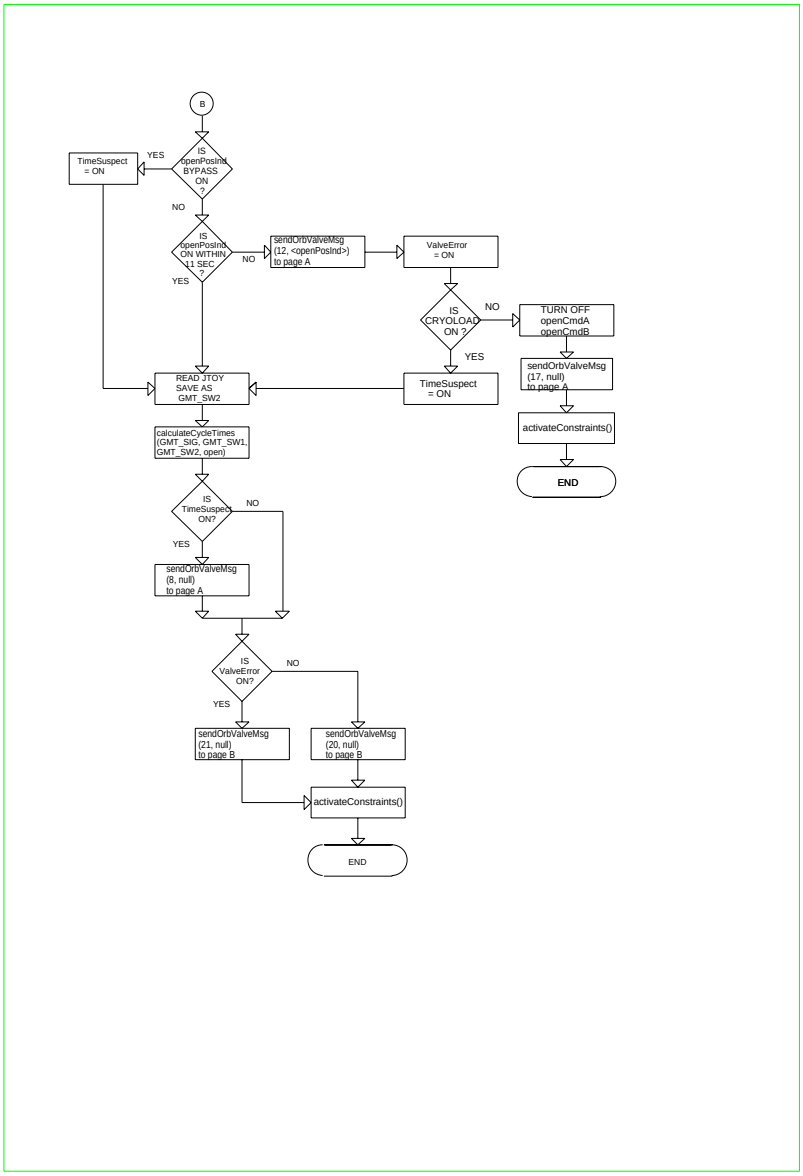
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

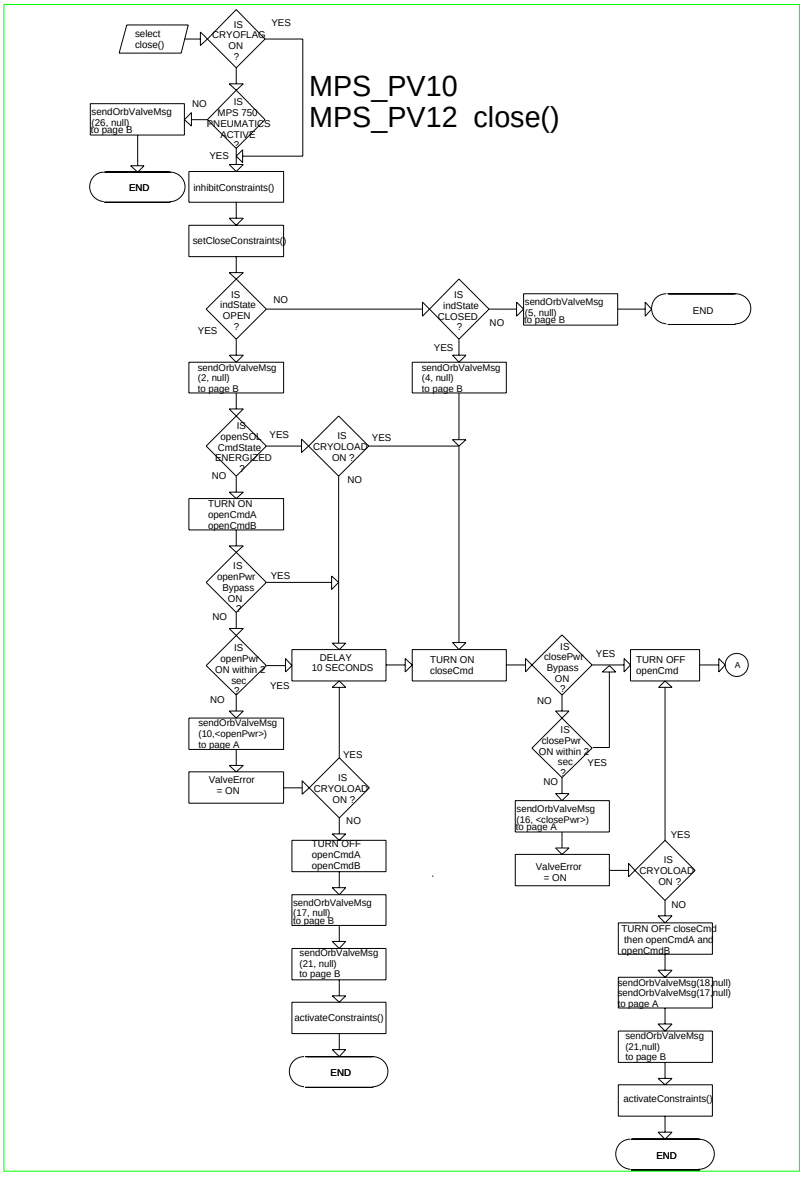


3.1.4.3.2. close()

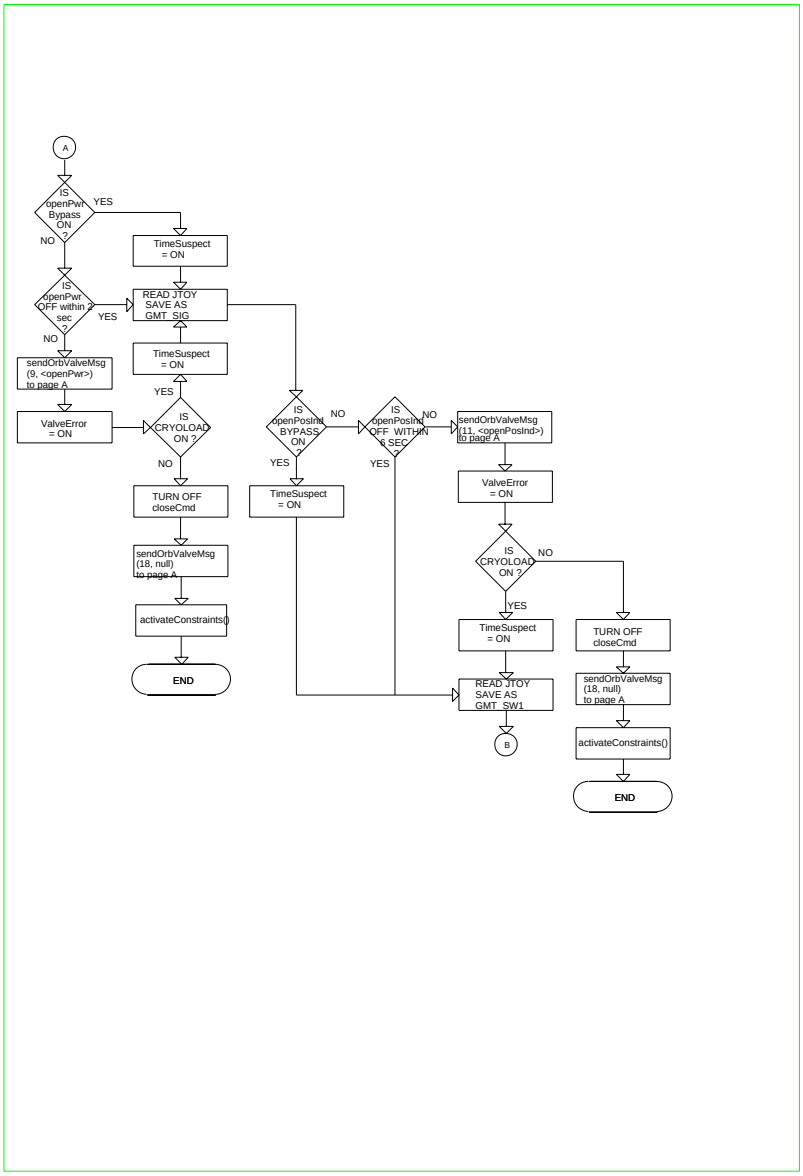
3.1.4.3.2.1. Descriptive Text

This method performs cycling of the bistable LO2 and LH2 inboard fill and drain valves to the closed position. The flow chart provides the necessary logic for valve cycles, including checks for snubbing pressure, proper response of power indications, and proper response of position indications. The valve cycles in a fail safe mode and 'wimps-out' (all command removed) at the first indication of hardware failure, unless the orbiter is in cryogenic load. If in cryo load, the valve cycles in a 'damn-the-torpedos' mode and completes all steps necessary to achieve the valve cycle, regardless of failures which are detected during the cycle. Valve signal to switch and switch to switch timing is also calculated. ValveError and TimeSuspect are initialized to OFF and Valve cycle times are set to 0 prior to starting valve cycle methods. If ValveError gets set ON during method run, cycle time FD data health should be set to invalid to aid in identifying suspect cycle data in trend analysis.

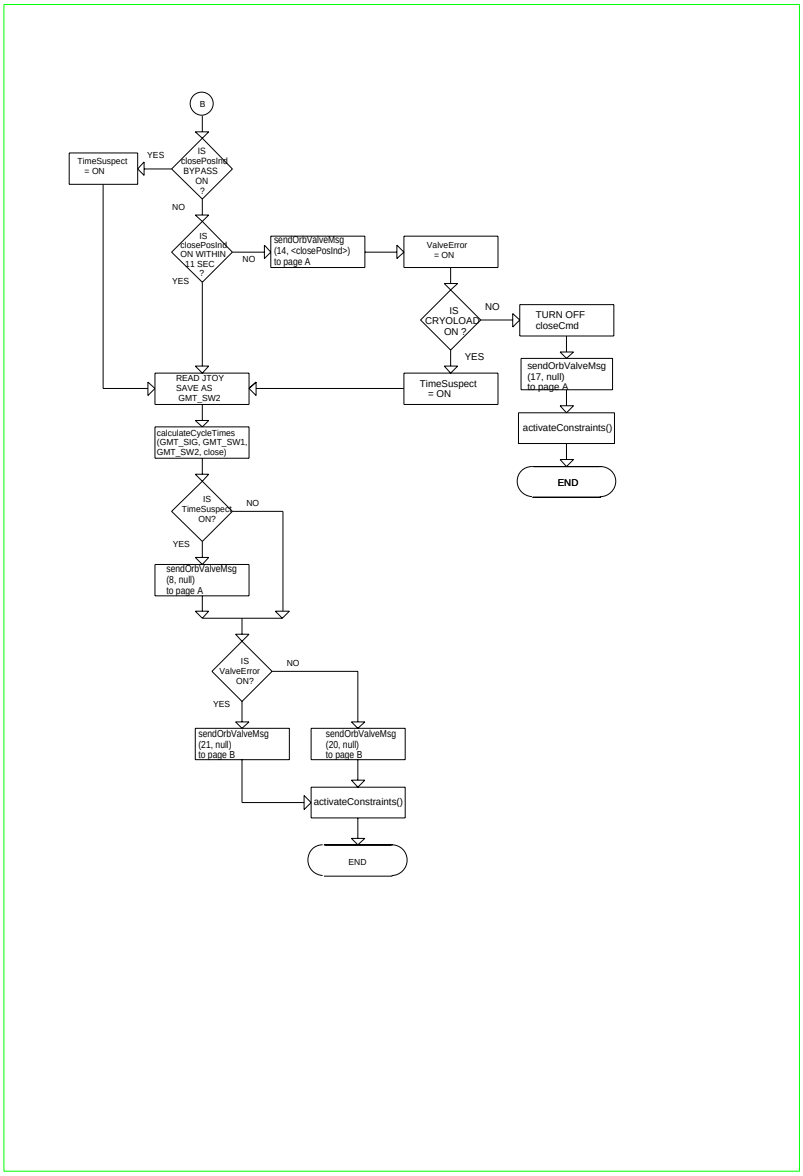
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

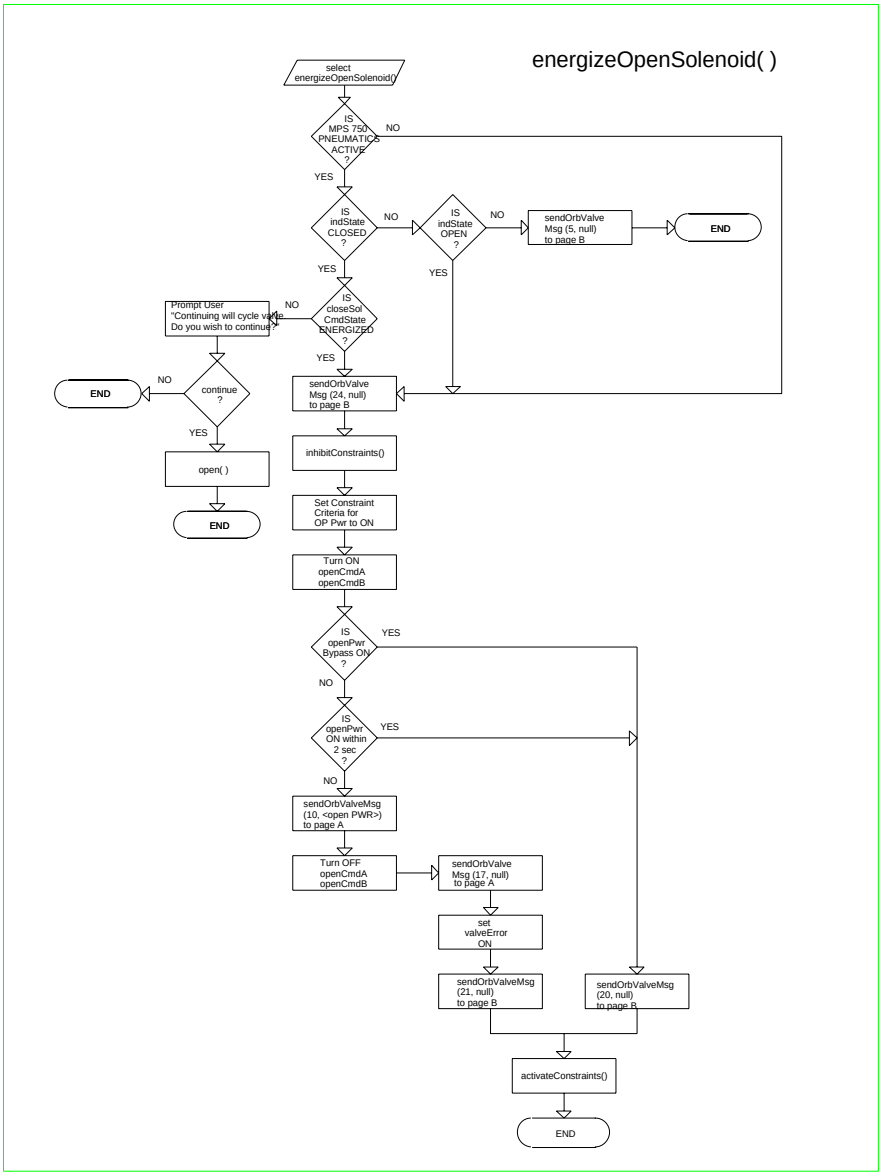


3.1.4.3.3. energizeOpenSolenoid()

3.1.4.3.3.1. Descriptive Text

This method performs the necessary checks to energize the inboard fill and drain valve opening solenoid. The sequence as identified on the following flow chart checks for valve present position, status of MPS 750 pneumatics, and result of energizing the opening solenoid if pneumatics is active. If a valve cycle will occur, the operator is prompted to provide input to continue (Yes/No option), and if a cycle is intended, the close() method is initiated. Constraints are maintained during this operation, with failure messages being generated on valve power indications.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

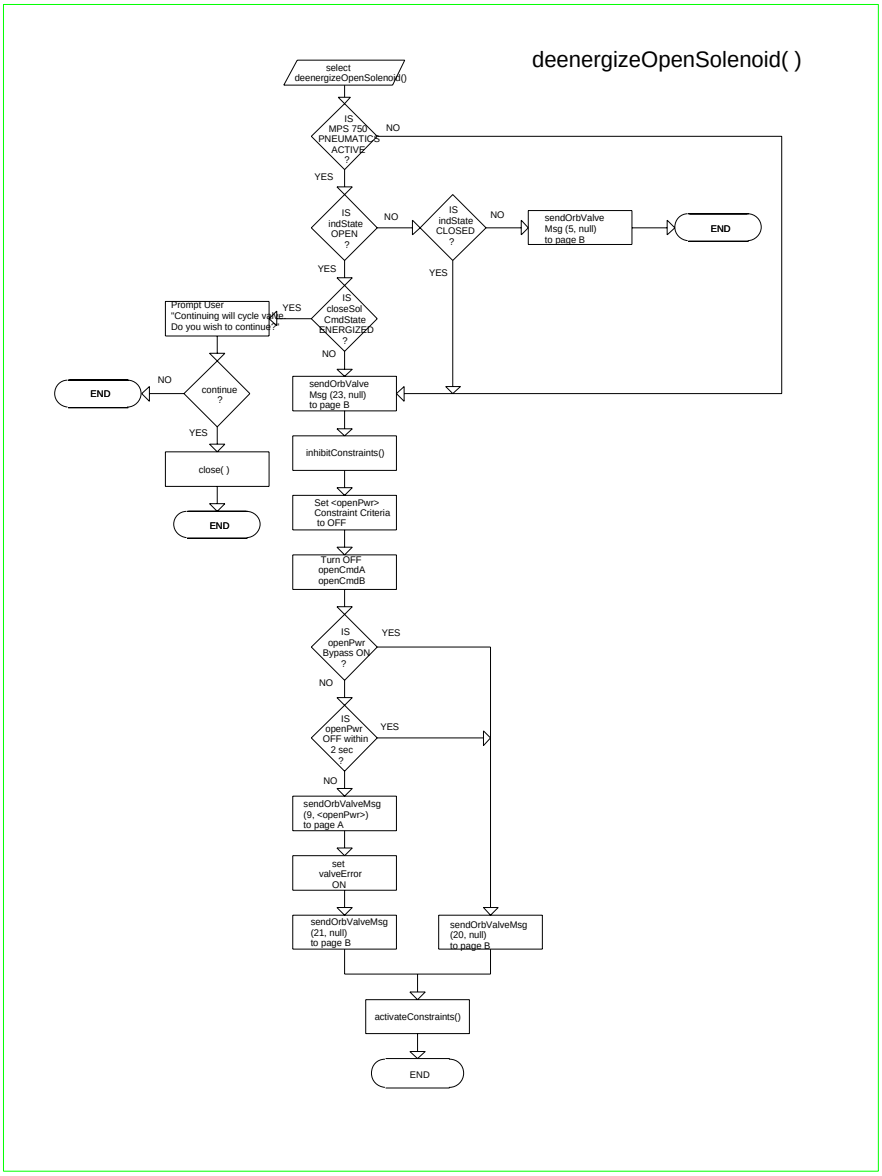


3.1.4.3.4. deenergizeOpenSolenoid()

3.1.4.3.4.1. Descriptive Text

This method performs the necessary checks to deenergize the inboard fill and drain valve opening solenoid. The sequence as identified on the following flow chart checks for valve present position, status of MPS 750 pneumatics, and result of deenergizing the opening solenoid if pneumatics is active. If a valve cycle will occur, the operator is prompted to provide input to continue (Yes/No option), and if a cycle is intended, the close() method is initiated. Constraints are maintained during this operation, with failure messages being generated on valve power indications.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

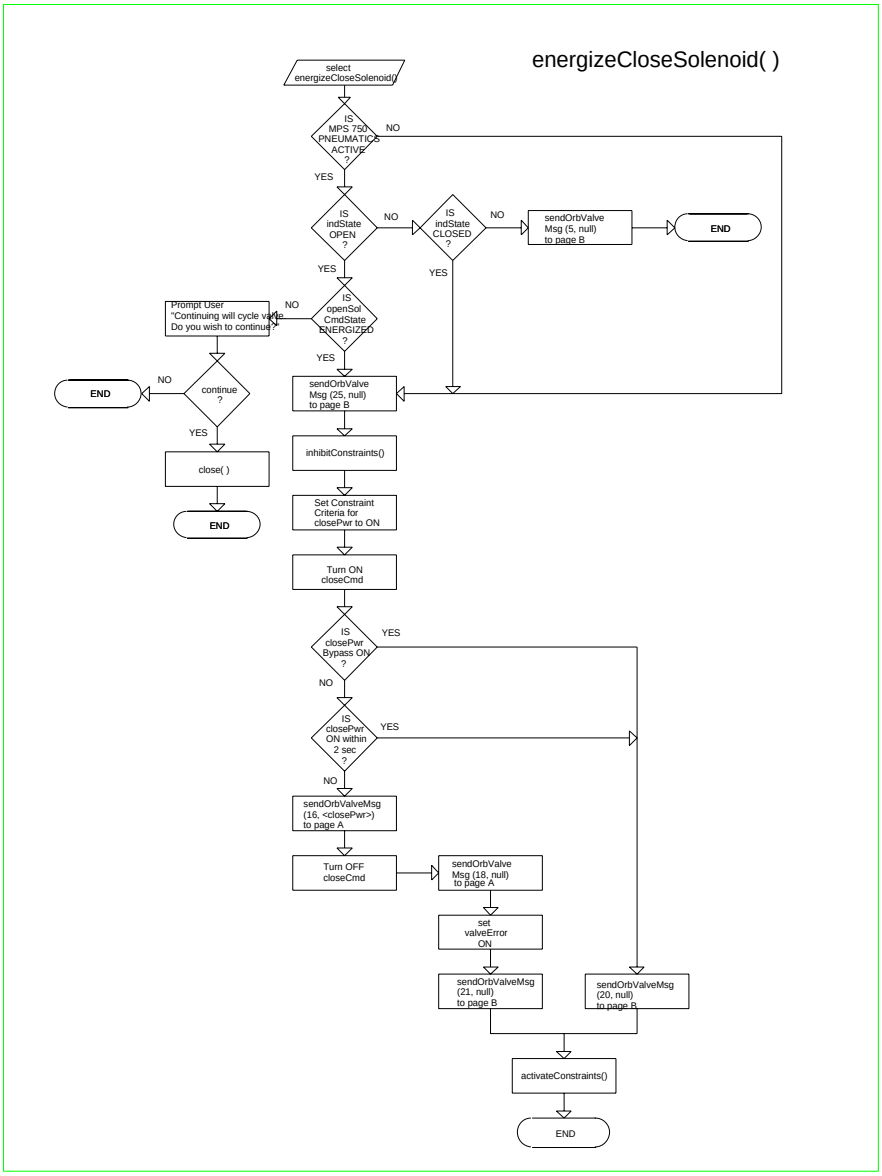


3.1.4.3.5. energizeCloseSolenoid()

3.1.4.3.5.1. Descriptive Text

This method performs the necessary checks to energize the inboard fill and drain valve closing solenoid. The sequence as identified on the following flow chart checks for valve present position, status of MPS 750 pneumatics, and result of energizing the closing solenoid if pneumatics is active. If a valve cycle will occur, the operator is prompted to provide input to continue (Yes/No option), and if a cycle is intended, the close() method is initiated. Constraints are maintained during this operation, with failure messages being generated on valve power indications.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

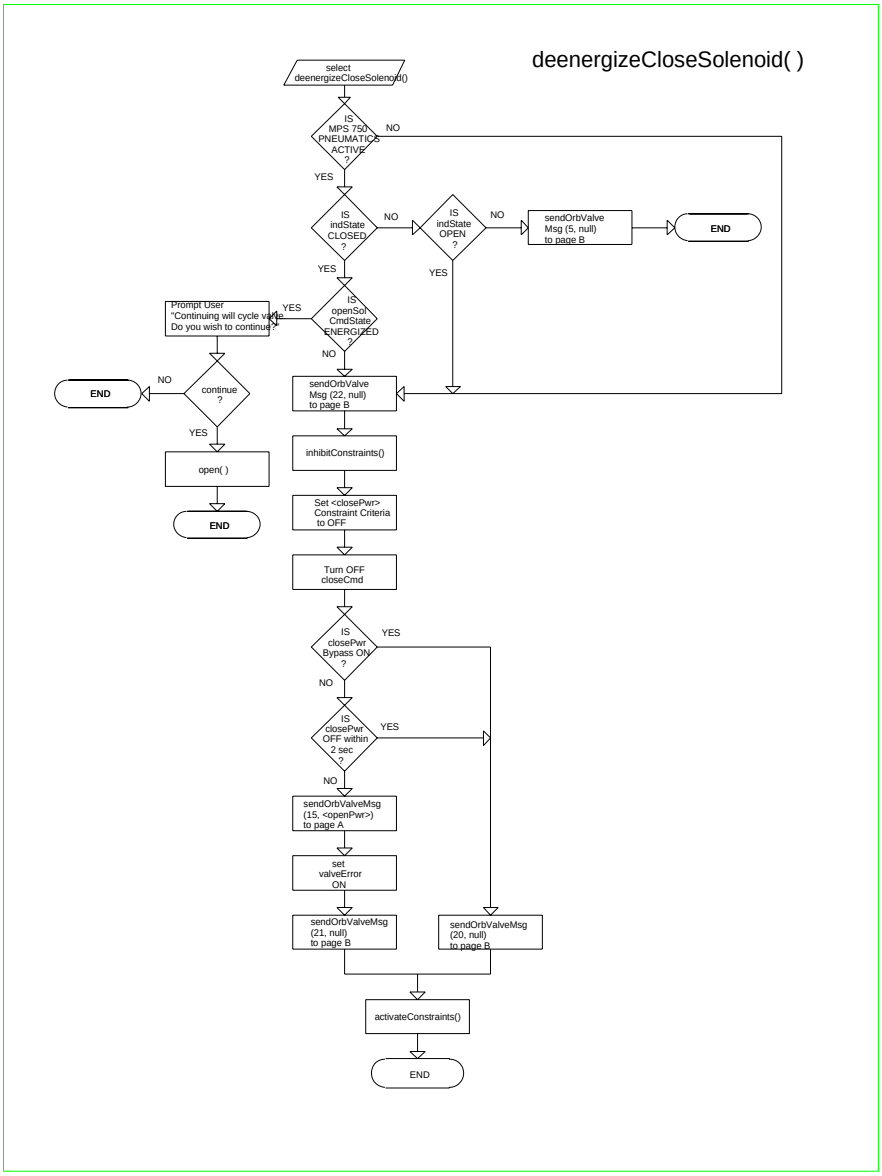


3.1.4.3.6. deenergizeCloseSolenoid()

3.1.4.3.6.1. Descriptive Text

This method performs the necessary checks to deenergize the inboard fill and drain valve closing solenoid. The sequence as identified on the following flow chart checks for valve present position, status of MPS 750 pneumatics, and result of deenergizing the closing solenoid if pneumatics is active. If a valve cycle will occur, the operator is prompted to provide input to continue (Yes/No option), and if a cycle is intended, the open() method is initiated. Constraints are maintained during this operation, with failure messages being generated on valve power indications.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



3.1.4.3.7. determineIndState()

3.1.4.3.7.1. Descriptive Text

This method will determine the valve's position based on its position indicators and associated indicator bypasses. The return value will be an enumerated type with possible values of OPEN, CLOSED, TRANSITION, or ERROR.

3.1.4.3.7.2. Effectivity

Operates on change data.

3.1.4.3.7.3. Sequence

Case:

<openPosInd> == OFF

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

<openPosInd>Byp == OFF
<closePosInd> == ON
<closePosInd>Byp == OFF
indState = CLOSED;

<openPosInd> == OFF
<openPosInd>Byp == OFF
<closePosInd>Byp == ON
indState = CLOSED;

<openPosInd>Byp == ON
<closePosIndB> == ON
<closePosInd>Byp == OFF
indState = CLOSED;

<openPosInd> == ON
<openPosInd>Byp == OFF
<closePosInd> == OFF
<closePosInd>Byp == OFF
indState = OPEN;

<openPosInd> == ON
<openPosInd>Byp == OFF
<closePosInd>Byp == ON
indState = OPEN;

<openPosInd>Byp == ON
<closePosInd> == OFF
<closePosInd>Byp == OFF
indState = OPEN;

<openPosInd> == OFF
<openPosInd>Byp == OFF
<closePosInd> == OFF
<closePosInd>Byp == OFF
indState = TRANSITION;

CASE ELSE
indState = ERROR;

End Case:

3.1.4.3.8. determineOpenSolCmdState()

3.1.4.3.8.1. Descriptive Text

This operation is used to determine the open solenoid's commanded state based on the open power indicator and its associated bypass. The return value will be an enumerated type with possible values of ENERGIZED, DEENERGIZED and ERROR.

3.1.4.3.8.2. Effectivity

Operates on change data.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.4.3.8.3. Sequence

Case:

<openPwr> == OFF
<openPwr>Byp == OFF
openSolCmdState = DEENERGIZED;

<openPwr> == ON
<openPwr>Byp == OFF
openSolCmdState = ENERGIZED;

<openPwr>Byp==On
openSolCmdState = ERROR;

End Case:

3.1.4.3.9. determineCloseSolCmdState()

3.1.4.3.9.1. Descriptive Text

This method will determine the close solenoid's commanded state based on the two close RPCs, the close power, and the associated bypasses. The return value will be an enumerated
type with possible values of ENERGIZED, DEENERGIZED, ERROR or UNSET.

3.1.4.3.9.2. Effectivity

Operates on change data.

3.1.4.3.9.3. Sequence

Case:

<closePwr> == OFF
<closePwr>Byp == OFF
closeSolCmdState = DEENERGIZED;

<closePwr> == ON
<closePwr>Byp == OFF
closeSolCmdState = ENERGIZED;

<closePwr>Byp == ON
closeSolCmdState = ERROR;

End Case:

3.1.4.3.10. getIndState()

3.1.4.3.10.1. Descriptive Text

This method will return the Enumerated Indicator State of the valve

3.1.4.3.10.2. Sequence

```
getIndState()
{
    return IndState;
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.1.4.3.11. getOpenSolCmdState()

3.1.4.3.11.1. Descriptive Text

This method will return the Enumerated Command State of the valve Opening Solenoid.

3.1.4.3.11.2. Sequence

```
getOpenSolCmdState()
{
        return OpenSolCmdState;
}
```

3.1.4.3.12. getCloseSolCmdState()

3.1.4.3.12.1. Descriptive Text

This method will return the Enumerated Command State of the valve Closing Solenoid.

3.1.4.3.12.2. Sequence

```
getCloseSolCmdState()
{
        return CloseSolCmdState;
}
```

3.1.4.3.13. verifyCommandsOff()

3.1.4.3.13.1. Descriptive Text

This function will verify that both the open and close commands are OFF

3.1.4.3.13.2. Sequence

```
verifyCommandsOff()
{
        open command status1 = byteTest4 <Open Command A>
        open command status2 = byteTest4 <Open Command B>
        close command status = byteTest4 <Close Command>
        {
        verify  open command status1 && open command status2  && close command status == OFF

        return TRUE
        }
        ELSE return FALSE
}
```

3.1.4.3.14. setCloseConstraints()

3.1.4.3.14.1. Descriptive Text

This method changes the constraint criteria of the measurements associated with this valve to those expected with the valve in the closed position.

3.1.4.3.14.2. Sequence

```
setCloseConstraints()
        {
                CHANGE <openPwr> CONSTRAINT CRITERIA TO OFF;

              CHANGE <openPosInd>  CONSTRAINT CRITERIA TO OFF;

              CHANGE <closePwr> CONSTRAINT CRITERIA TO ON;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

CHANGE <closePosInd> CONSTRAINT CRITERIA TO ON;
}

3.1.4.3.15. setClosePosConstraints()

3.1.4.3.15.1. Descriptive Text

This method changes the constraint criteria of the measurements associated with this valve to those expected with the valve in the closed position with no power (pneumatics) applied.

3.1.4.3.15.2. Sequence

setClosePosConstraints()
{
CHANGE <openPosInd> CONSTRAINT CRITERIA TO OFF;

CHANGE <closePosInd> CONSTRAINT CRITERIA TO ON;

CHANGE <closePwr> CONSTRAINT CRITERIA TO OFF;

CHANGE <openPwr> CONSTRAINT CRITERIA TO OFF;

}

3.1.4.3.16. setOpenConstraints()

3.1.4.3.16.1. Descriptive Text

This method changes the constraint criteria of the measurements associated with this valve to those expected with the valve in the open position.

3.1.4.3.16.2. Sequence

setOpenConstraints()
{
CHANGE <openPwr> CONSTRAINT CRITERIA TO ON;

CHANGE <openPosInd> CONSTRAINT CRITERIA TO ON;

CHANGE <closePwr> CONSTRAINT CRITERIA TO OFF;

CHANGE <closePosInd> CONSTRAINT CRITERIA TO OFF;

}

3.1.4.3.17. setOpenPosConstraints()

3.1.4.3.17.1. Description

This method changes the constraint criteria of the measurements associated with this valve to those expected with the valve in the closed position with no power (or pneumatics) applied.

3.1.4.3.17.2. Sequence

setOpenPosConstraints()
{
CHANGE <openPosInd> CONSTRAINT CRITERIA TO ON;

CHANGE <closePosInd> CONSTRAINT CRITERIA TO OFF;

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

CHANGE <closePwr> CONSTRAINT CRITERIA TO OFF;

CHANGE <openPwr> CONSTRAINT CRITERIA TO OFF;

}

3.1.4.3.18. activateConstraints()

3.1.4.3.18.1. Descriptive Text

This operation activates constraint monitoring on valve position and power indicators.

3.1.4.3.18.2. Sequence

activateConstraints()
{
Initiate Constraint Monitoring on
<openPwr>
<closePwr>
<openPosInd>
<closePosInd>
}

3.1.4.3.19. inhibitConstraints()

3.1.4.3.19.1. Descriptive Text

This operation de-activates constraint monitoring on valve position and power indicators.

3.1.4.3.19.2. Sequence

inhibitConstraints()
{
Inhibit Constraint Monitoring on
<openPwr>
<closePwr>
<openPosInd>
<closePosInd>
}

3.1.4.3.20. autocycle()

3.1.4.3.20.1. Descriptive Text

3.1.4.3.20.2. Sequence

{

If instance IndState == CLOSED
open();
If valveCycleError is ON, set autoValveCheckoutError = ON
close();
If valveCycleError is ON, set autoValveCheckoutError = ON

If instance IndState == OPEN
close();
If valveCycleError is ON, set autoValveCheckoutError = ON
open();
If valveCycleError is ON, set autoValveCheckoutError = ON

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
        Else
                sendOrbValveMsg(5)

Return autoValveCheckoutError;

}
```

3.1.4.3.21. verifyTminus3BT4()//Phase 3

3.1.4.3.21.1. Description

This method performs a byte test 4 of fill and drain valve commands to verify they are in a proper T-3 hour configuration. For MPS_PV12 instance of this class, open commands should be OFF, close commands should be ON. For MPS_PV10 instance of this class, open commands should be ON, close commands should be OFF.

3.1.4.3.21.2. Sequence

```
begin sequence;

set commanderrorcheck OFF
{
For MPS_PV10 instance;

perform ByteTest4 of <openCMD>
        verify ByteTest4 = ON
        else send message to page A
                "Valve" <openCMD> "is not in T-3 hour configuration."
                and set commanderrorcheck ON
}
{
perform ByteTest4 of <closeCMD>
        verify ByteTest4 = OFF
        else send message to page A
                "Valve" <closeCMD> "is not in T-3 hour configuration."
                and set commanderrorcheck ON
}
For MPS_PV12 instance;

perform ByteTest4 of <openCMD>
        verify ByteTest4 = OFF
        else send message to page A
                "Valve" <openCMD> "is not in T-3 hour configuration."
                and set commanderrorcheck ON
}
{
perform ByteTest4 of <closeCMD>
        verify ByteTest4 = ON
        else send message to page A
                "Valve" <closeCMD> "is not in T-3 hour configuration."
                and set commanderrorcheck ON
}
return commanderrorcheck

end sequence;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.4.3.22. verifyTminus6BT4()//Phase 3

3.1.4.3.22.1. Description

This method performs a byte test 4 of fill and drain valve commands to verify they are in a proper T-3 hour configuration. For both instances of this class, open commands should be ON, close commands should be OFF.

3.1.4.3.22.2. Sequence

begin sequence;

```
set commanderrorcheck OFF
{
perform ByteTest4 of <openCMD>
        verify ByteTest4 = ON
        else send message to page A
                "Valve" <openCMD> "is not in T-3 hour configuration."
                and set commanderrorcheck ON
}
{
perform ByteTest4 of <closeCMD>
        verify ByteTest4 = OFF
        else send message to page A
                "Valve" <closeCMD> "is not in T-3 hour configuration."
                and set commanderrorcheck ON
}
return commanderrorcheck
```

end sequence;

3.1.4.3.23. setCloseInhibitOn()//Phase 3

3.1.4.3.24. setCloseInhibitOff()//Phase 3

3.1.4.3.25. setValveCloseConstraints()//Phase 3

3.1.4.3.26. monitorForInitialCloseTime()//Phase 3

3.1.4.3.27. stopMonitorForInitialCloseTime()//Phase 3

3.1.4.3.28. getInitialCloseTime()//Phase 3

3.1.4.4. Safety Requirements

3.1.5. MPS_Valve_ORB_LH2_PreValve

3.1.5.1. Description

MPS_Valve_ORB_BISTABLE_LH2_Prevalve is a bi-stable valve with 3 open and 3 close commands, 2 open and 1 close position indications, 1 open and 1 close power indications, 2 open and 2 close RPCs. The valve actuator solenoids (1 each open and close) are each redundantly commanded/powered. There are 3 instances of this valve, one for each SSME feedline (MPS_PV4, PV5, PV6). Functionally, the LH2 prevalves serve to provide the capability to isolate the orbiter LH2 feed manifold from a shut down SSME during ascent or in the event of a hydrogen leak during tanking. The valve class inherits from the MPS_Valve_ORB Class. The valve has an association with the MPS_Helium_System class for status of the MPS 750 pneumatic system. For integrated

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

operations, this valve class has an association with the MPS_System for determination of the cryo load stautus via a Cryo Flag pseudo. During test and check-out mode, valve cycles will break-out and remove all commands/power when a failure is detected. When the Cyro Flag is ON, the valve cycle will continue to completion regardless of failures.

3.1.5.1.1. Instance Index

Instance	Display User	Other Users
MPS_PV4	MPS	None
MPS_PV5	MPS	None
MPS_PV6	MPS	None

3.1.5.1.2. Class Diagram

<insert class diagram here>

3.1.5.1.3. Attributes and Aggregations

Attribute	Data Type	Initial Value	Publish / Persistence	Client Access	Description
openPosIndA	DM_ON_OFF			R	
openPosIndB	DM_ON_OFF			R	
closePosInd	DM_ON_OFF			R	
openCmdA	DS_ON_OFF			R,W	
openCmdB	DS_ON_OFF			R,W	
openCmdC	DS_ON_OFF			R,W	
openRPC1	DM_ON_OFF			R	
openRPC2	DM_ON_OFF			R	
openPwr	DM_ON_OFF			R	
closeCmdA	DS_ON_OFF			R,W	
closeCmdB	DS_ON_OFF			R,W	
closeCmdC	DS_ON_OFF			R,W	
closePwr	DM_ON_OFF			R	
closeRPC1	DM_ON_OFF			R	
closeRPC2	DM_ON_OFF			R	

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

closeSolCmdState	MPS_ORB_SOL_STATE		PUB	R	
openSolCmdState	MPS_ORB_SOL_STATE		PUB	R	

3.1.5.1.4. Methods Index

Operation	**Description**
open()	This method performs necessary functions to cycle valve to the open position.
close()	This method performs necessary functions to cycle valve to the closed position.
energizeOpenSolenoid()	This method energizes open solenoid. Method includes a check of MPS pneumatics status, and present valve indicator state to determine consequence of energizing valve. The method then runs the appropriate open() method if the operator chooses to continue via prompt response.
energizeCloseSolenoid()	This method energizes closing solenoid. Method includes a check of MPS pneumatics status, and present valve indicator state to determine consequence of energizing valve. The method then runs the appropriate close() method if the operator chooses to continue via prompt response.
deenergizeOpenSolenoid()	This method de-energizes opening solenoid. Method includes a check of MPS pneumatics status, and present valve indicator state to determine consequence of energizing valve. The method then runs the appropriate close() method if the operator chooses to continue via prompt response.
deenergizeCloseSolenoid()	This method de-energizes closing solenoid. Method includes a check of MPS pneumatics status, and present valve indicator state to determine consequence of energizing valve. The method then runs the appropriate open() method if the operator chooses to continue via prompt response.
openOnCheck()	This method verifies that the open power and RPCs energize within 2 seconds, and prints error messages for any unbypassed indicators that do not.
openOffCheck()	deThis method verifies that the open power and RPCs energize within 2 seconds, and prints error messages for any unbypassed indicators that do not.
closeOnCheck()	This method verifies that the close power and RPCs energize within 2 seconds, and prints error messages for any unbypassed indicators that do not.
closeOffCheck()	This method verifies that the close power and RPCs deenergize within 2 seconds, and prints error messages for any unbypassed indicators that do not.
openPosIndOnCheck()	This method displays error messages for any open position indicators that are off and unbypassed.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

openPosIndOffCheck()	This method displays error messages for any open position indicators that are on and unbypassed.
determineOpSolCmdState()	This method will determine the command state of the opening solenoid valve based on valve power indicators and will return either: ENERGIZED DEENERGIZED ERROR
determineClSolCmdState()	This method will determine the command state of the opening solenoid valve based on valve power indicators and will return either: ENERGIZED DEENERGIZED ERROR
determineIndState()	This method will determine the position state of the valve based on all unbypassed open/close indicators and will return either: OPEN CLOSED TRANSITION ERROR
determineCmdState()//Phase 3	This method will determine the overall command state of the valve by voting on openSolCmdState and closeSolCmdState. The possible overall command states are: CMD_CLOSED CMD_OPEN CMD_SNUBBED UNCOMMANDED ERROR
determineState()//Phase 3	This method will determine the overall state of the valve by voting on the overall command state and the indicator state. The possible states are: OPEN CLOSED OPEN_ERROR CLOSED_ERROR TRANSITION ERROR
getIndState()	This method will return valve indicator Enumerated state.
getOPSolCmdState()	This method will return valve opening solenoid Enumerated command state.
getClSolCmdState()	This method will return valve closing solenoid Enumerated command state.
getCmdState()	This method will return the overall Enumerated valve command state.
getState()	This method will return the overall state of the valve

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

verifyCmdsOff()	This method will verify no command issued by performing a byte Test 4 of all LH2 prevalve commands.
activateConstraints()	This method will activate Constraints for either the close or open positions when in standby mode (uncommanded present state).
inhibitConstraints()	This method will inhibit Constraint notification during valve cycle operations.
autoCycle()	This method will perform valve open/close autocycle operations.
setOpenConstraints()	This method will set the constraint criteria for all indicators associated with this valve to the states expected for the valve in the open position.
setOpenPosConstraints()	This method will set-up constraints for a uncommanded open bistable valve.
setCloseConstraints()	This method will set the constraint criteria of all indicators associated with this valve to the states expected for the valve in the close position.
setClosePosConstraints	This method will set-up constraints for a uncommanded closed bistable valve.
verifyTminus6BT4()//Phase 3 method	This method does a byte test 4 of valve commands to verify T-6 hour configuration.
verifyTminus3BT4()//Phase 3 method	This method does a byte test 4 of valve commands to verify T-3 hour configuration.

3.1.5.1.5. Associations

Association	Type	Class	Description
MPS_SubSystem			CryoFlag status.
MPS_Helium_SubSystem			MPS 750 pneumatic status.

3.1.5.2. State Description

3.1.5.2.1. State Diagram

<insert state diagram here>

3.1.5.2.2. State Table

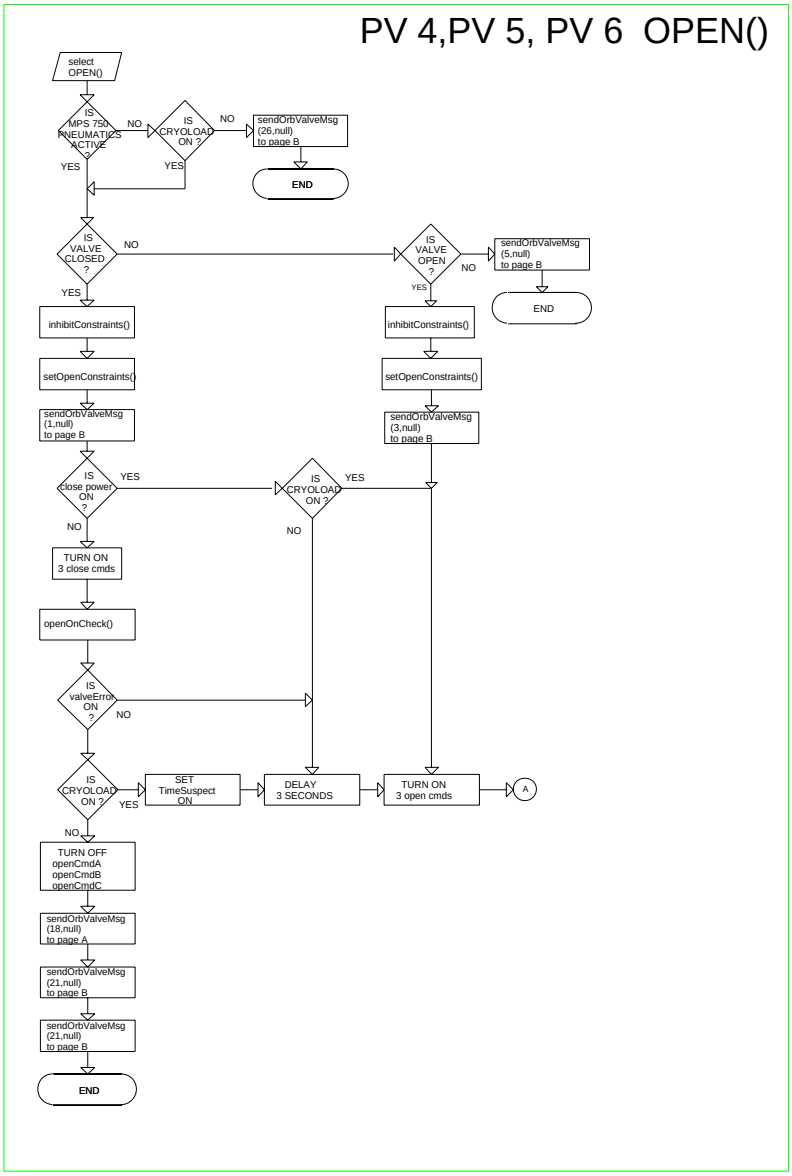
3.1.5.3. Operations

3.1.5.3.1. open()

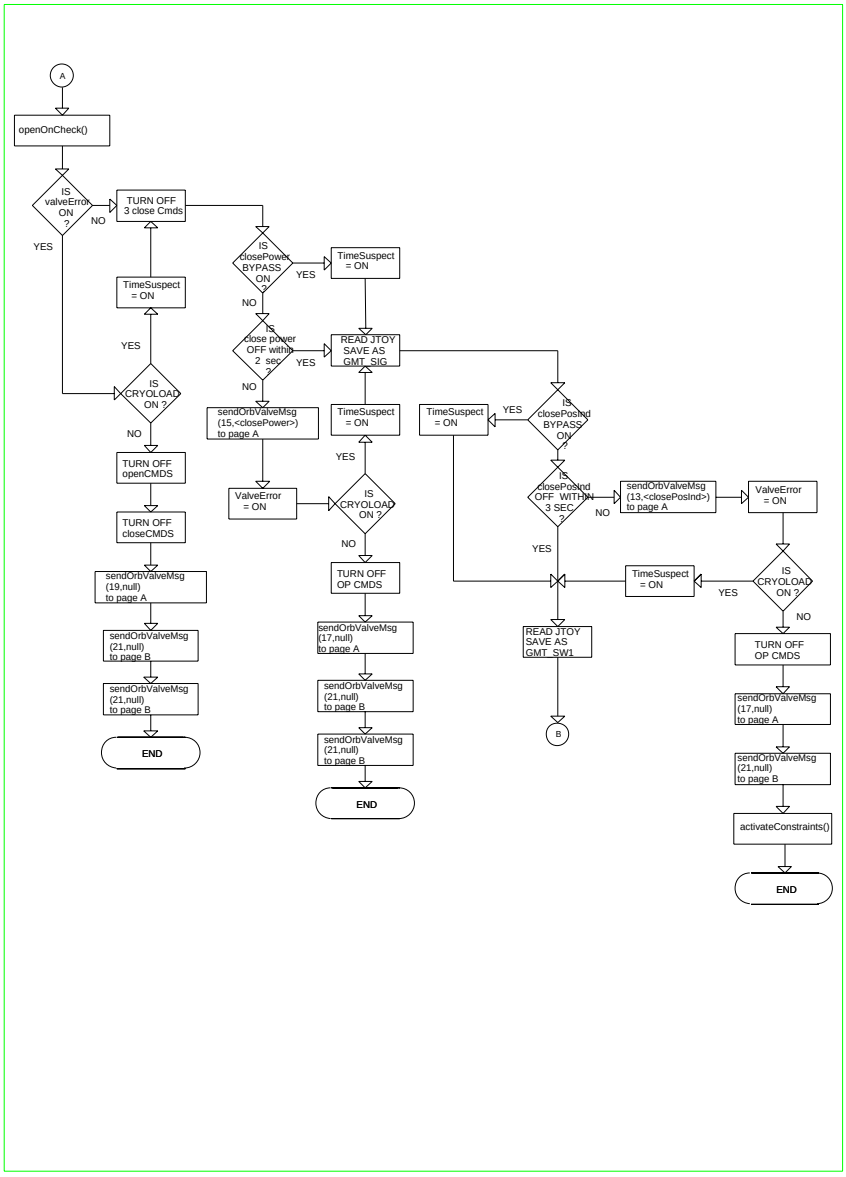
3.1.5.3.1.1. Descriptive Text

The flow charts contained below describe the logic to be used to code the method.

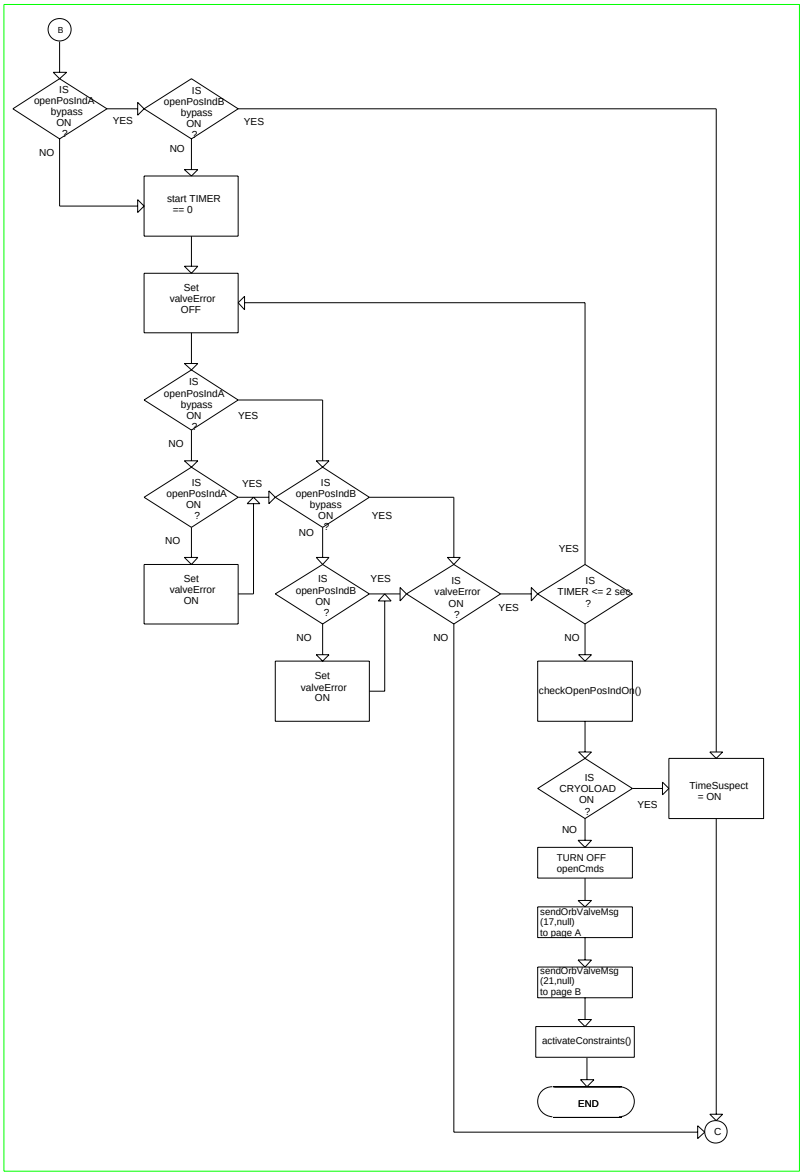
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



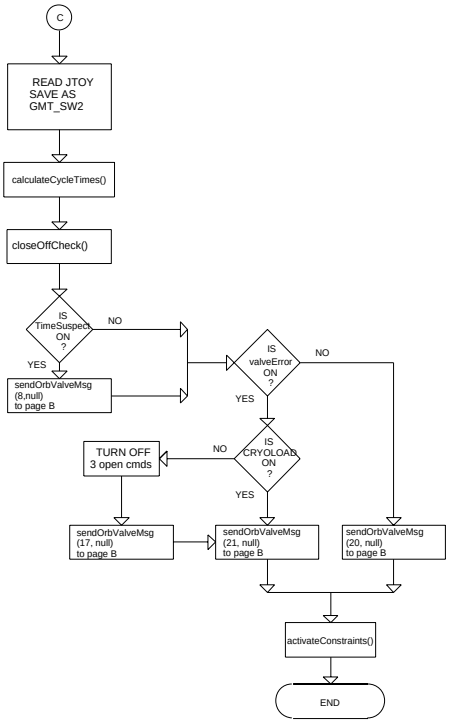
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

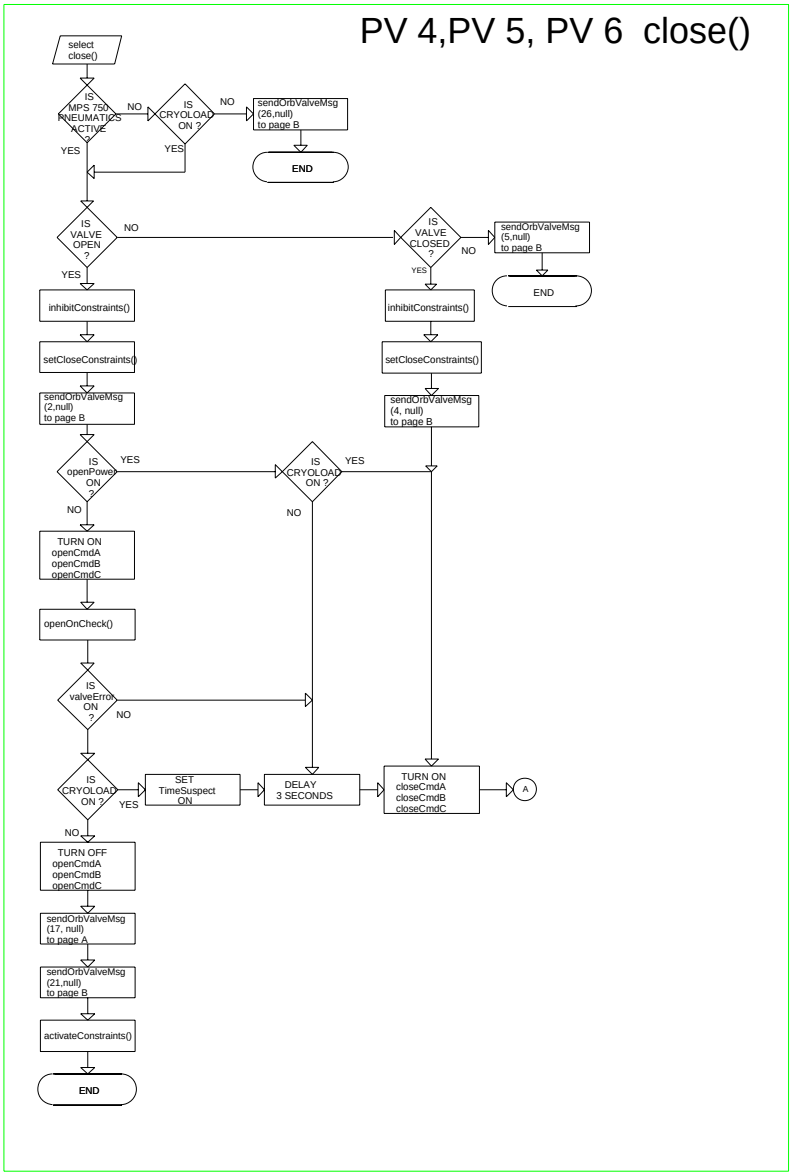


3.1.5.3.2. close()

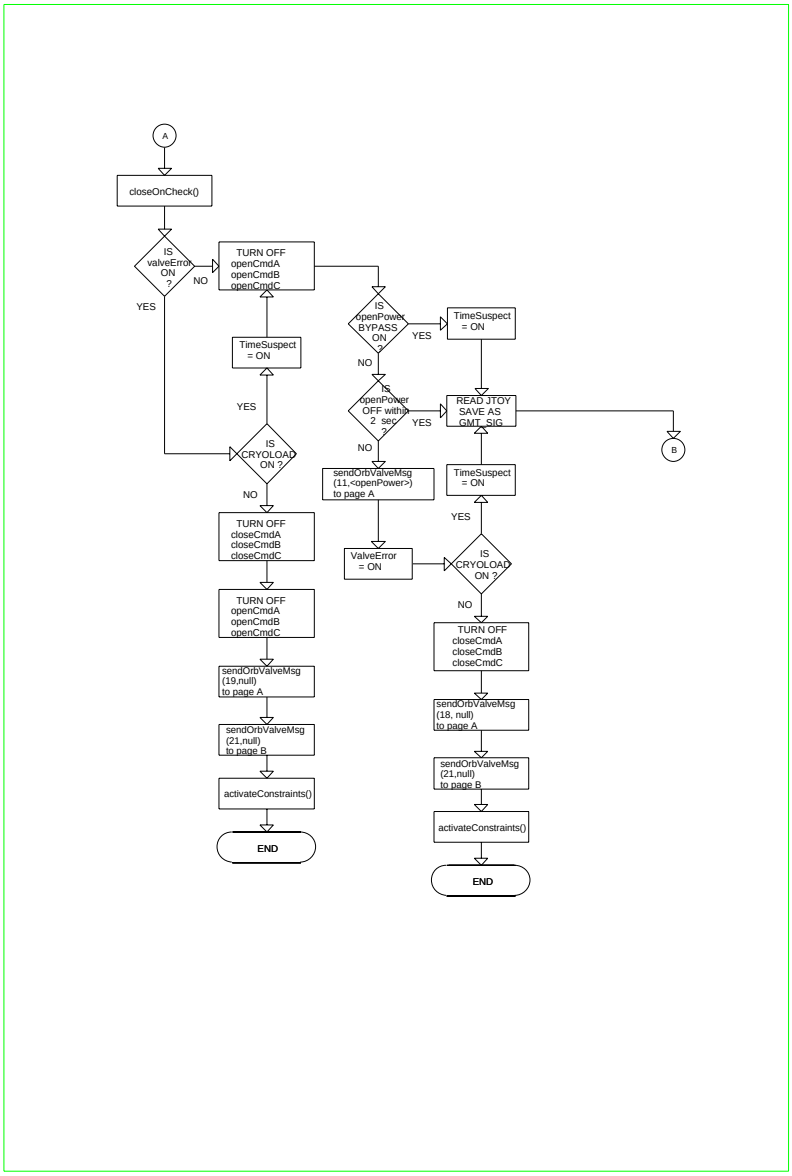
3.1.5.3.2.1. Descriptive Text

The flow charts contained below describe the logic to be used to code the method.

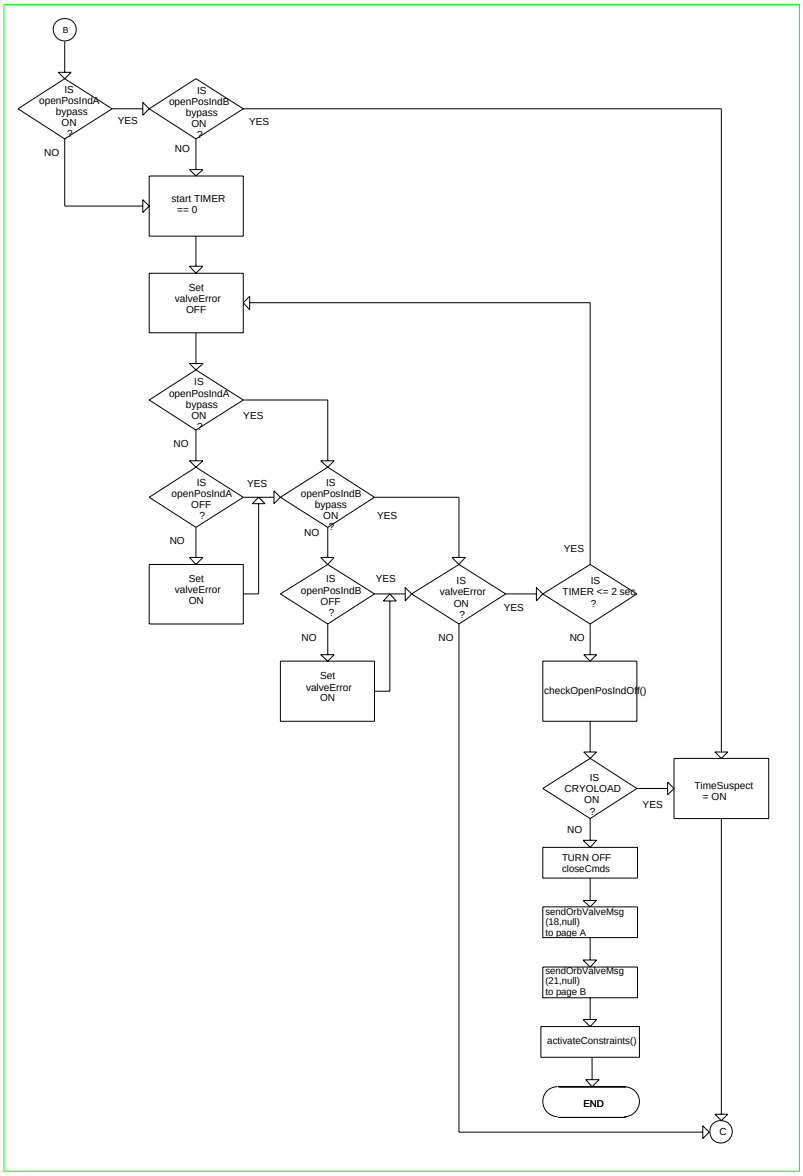
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work



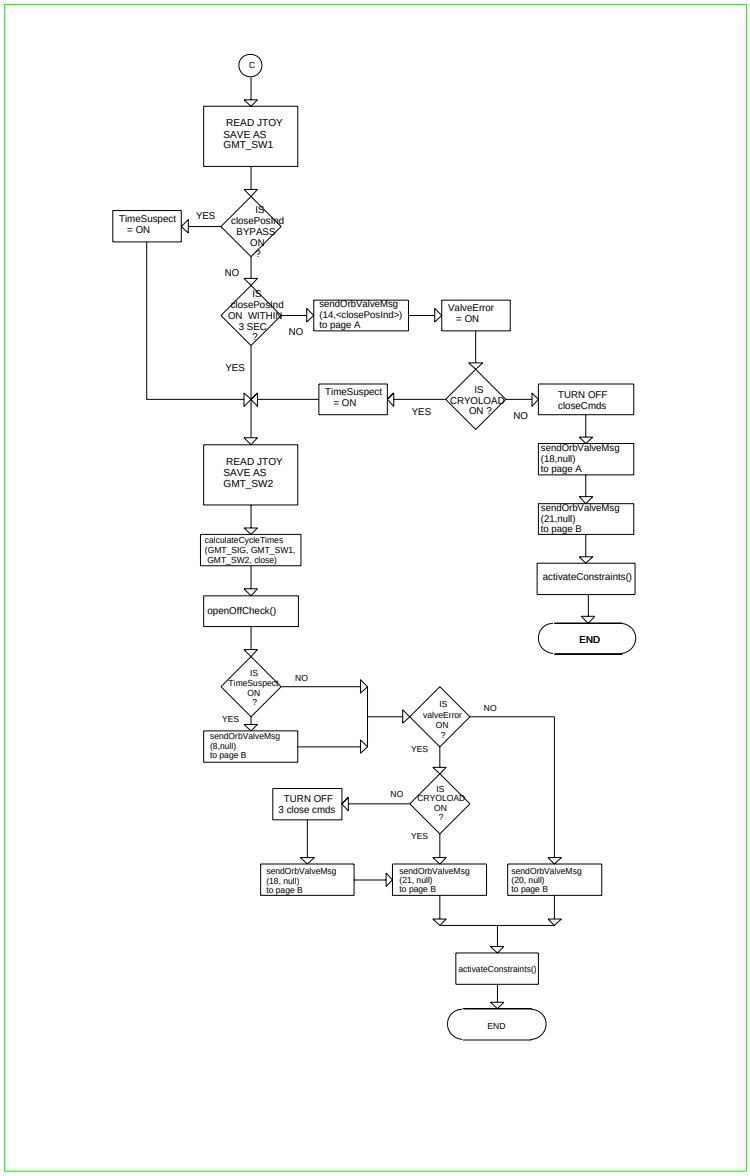
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

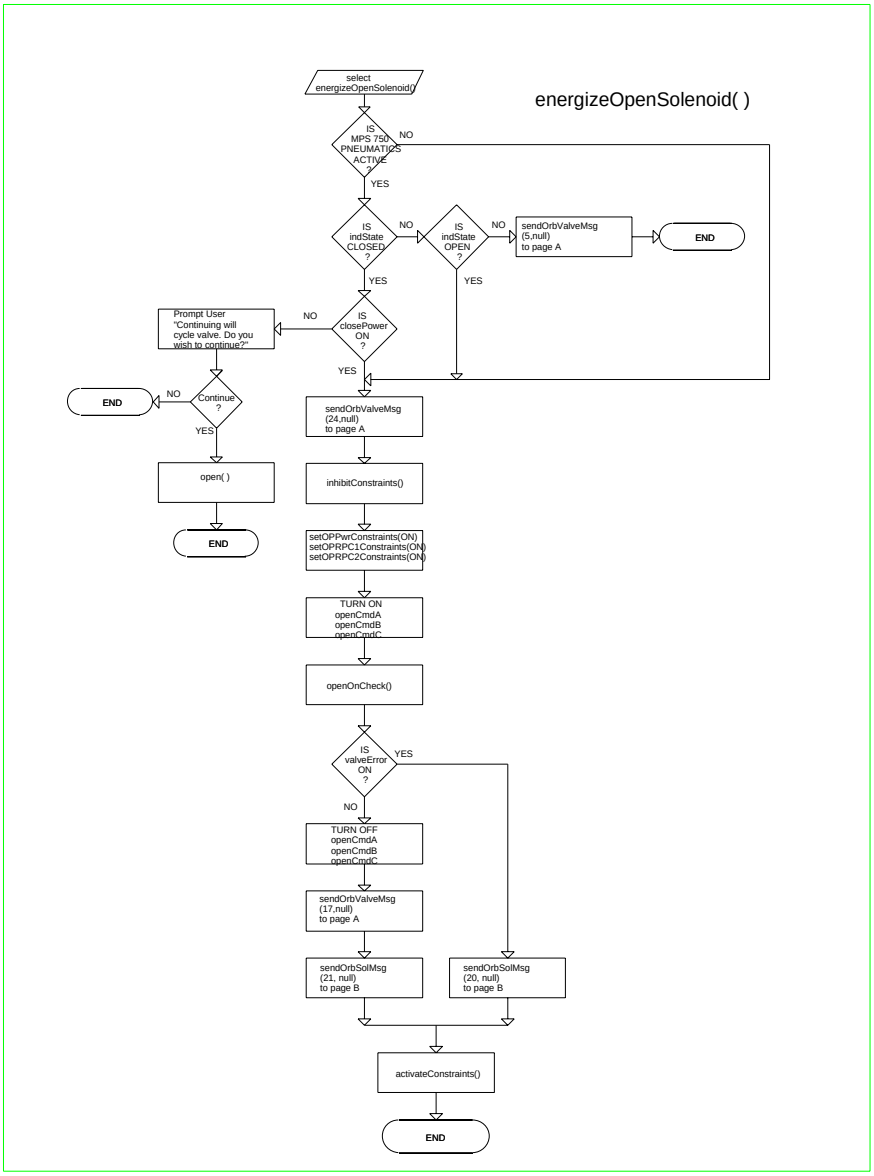


3.1.5.3.3. energizeOpenSolenoid()

3.1.5.3.3.1. Descriptive Text

The flow charts contained below describe the logic to be used to code the method.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

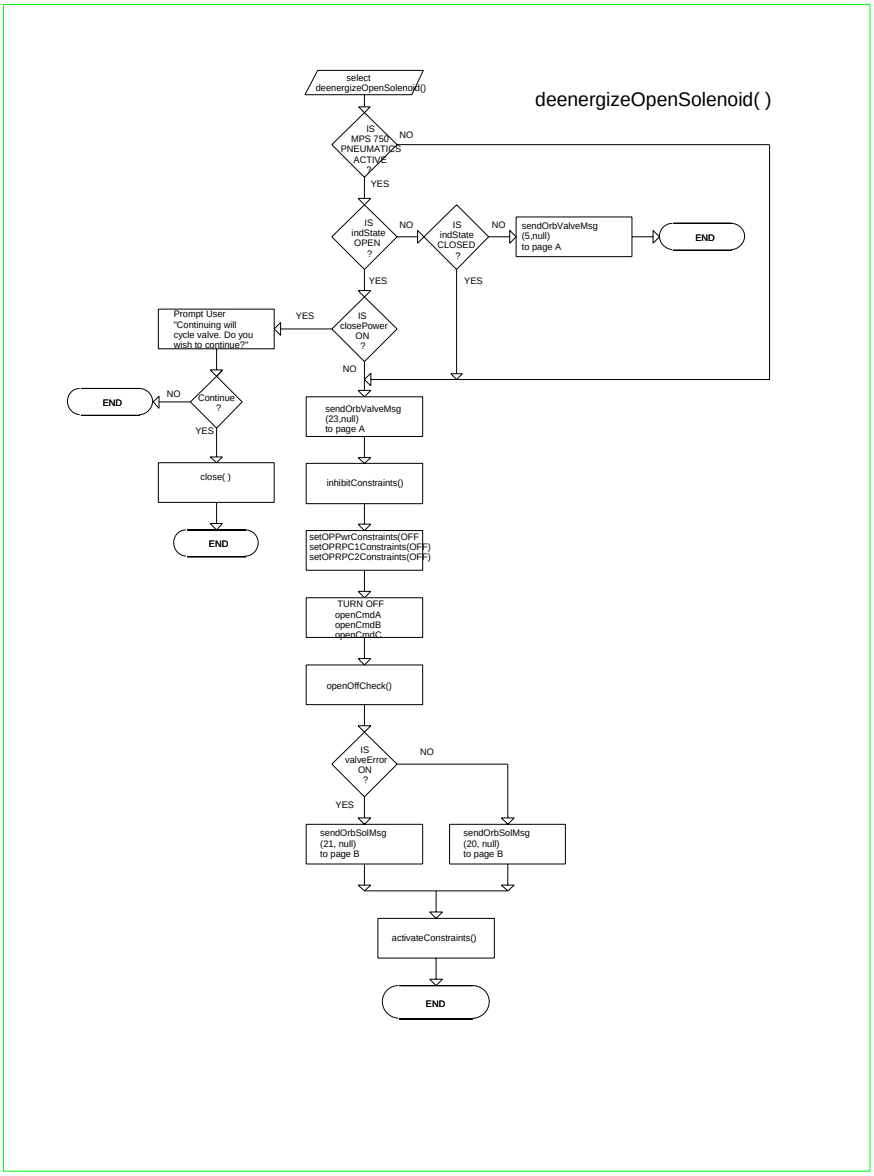


3.1.5.3.4. deenergizeOpenSolenoid()

3.1.5.3.4.1. Descriptive Text

The flow charts contained below describe the logic to be used to code the method.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

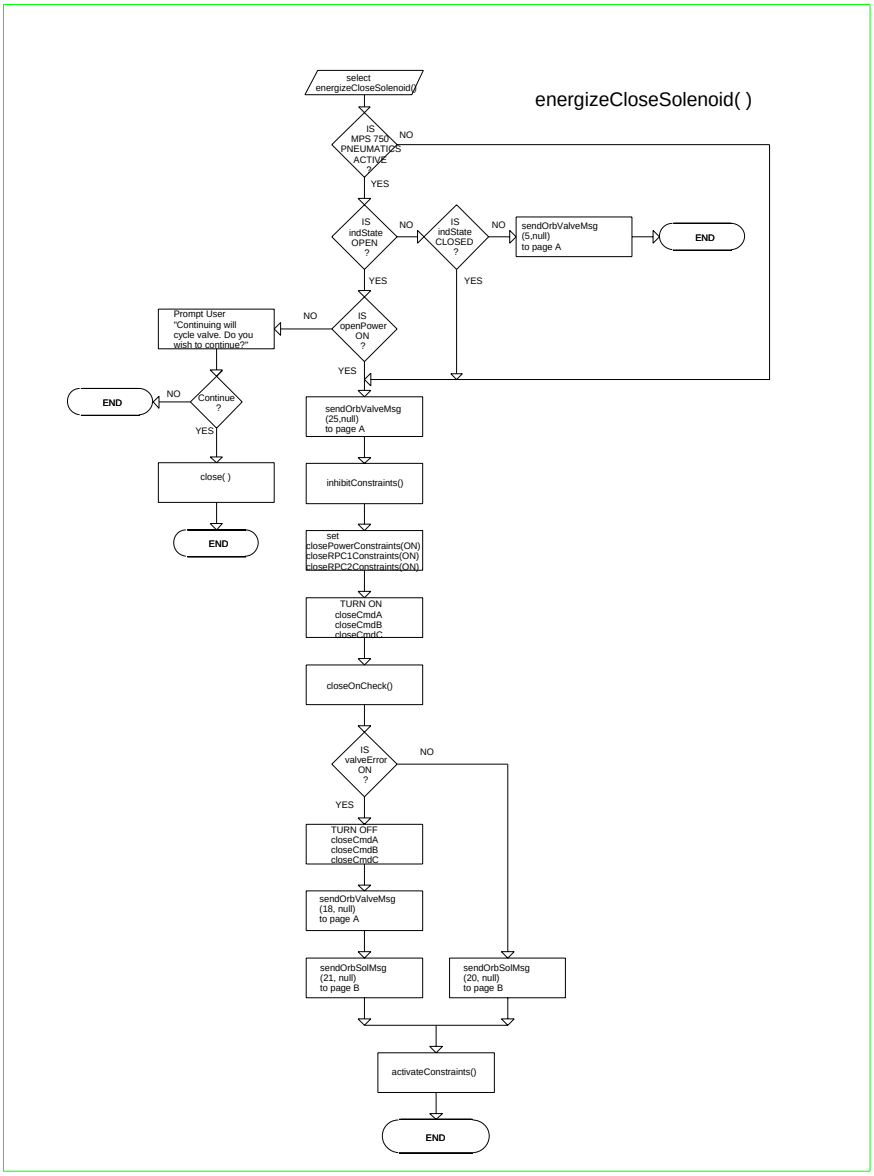


3.1.5.3.5. energizeCloseSolenoid()

3.1.5.3.5.1. Descriptive Text

The flow charts contained below describe the logic to be used to code the method.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

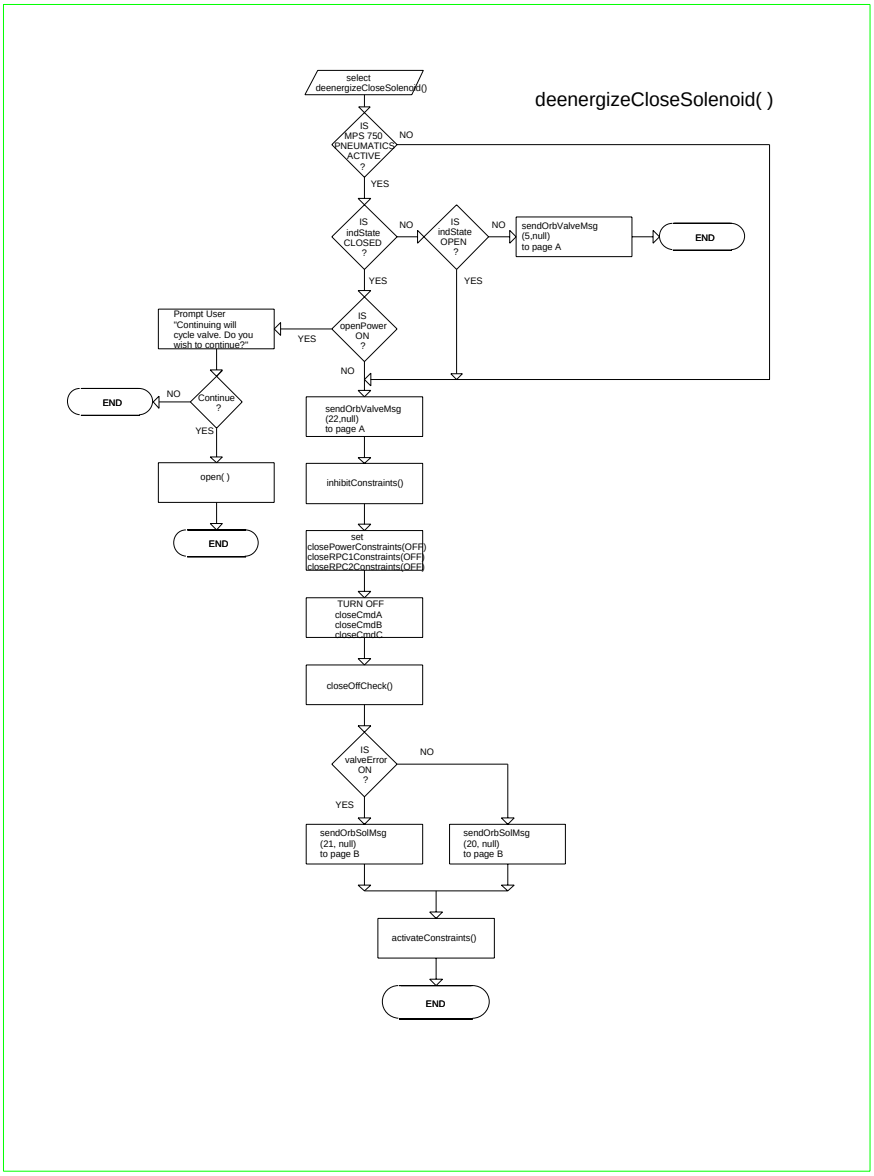


3.1.5.3.6. deenergizeCloseSolenoid()

3.1.5.3.6.1. Descriptive Text

This method perfroms the steps necessary to de-energize the LH2 prevalve closing solenoids. This includes checking for failures of RPC/power indicators and maintenance of constraint criteria (Constraints).

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

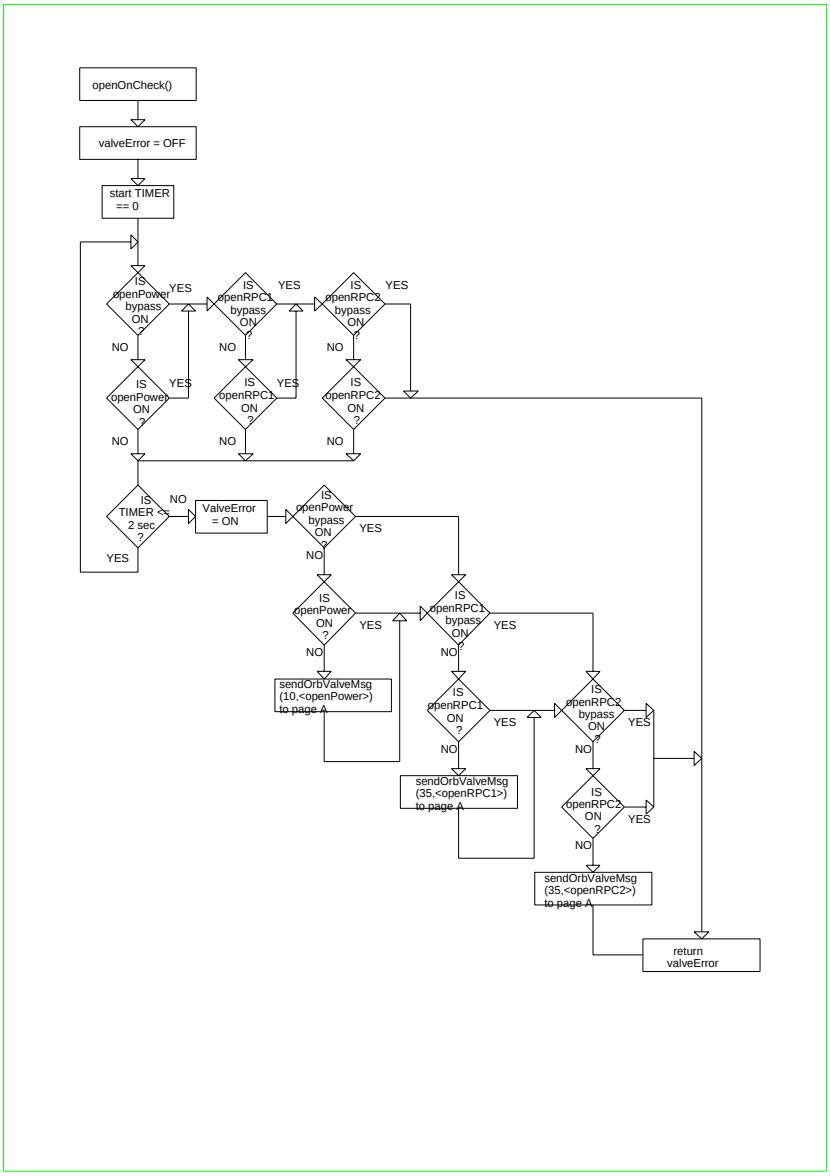


3.1.5.3.7. openOnCheck()

3.1.5.3.7.1. Descriptive Text

This method is used by the valve cycle method to determine whether the open power and RPCs energize within 2 seconds. The method prints error messages for any open power or RPC that does not energize in the alloted time. The method also returns valveError = ON if errors occur and OFF for a nominal cycle.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

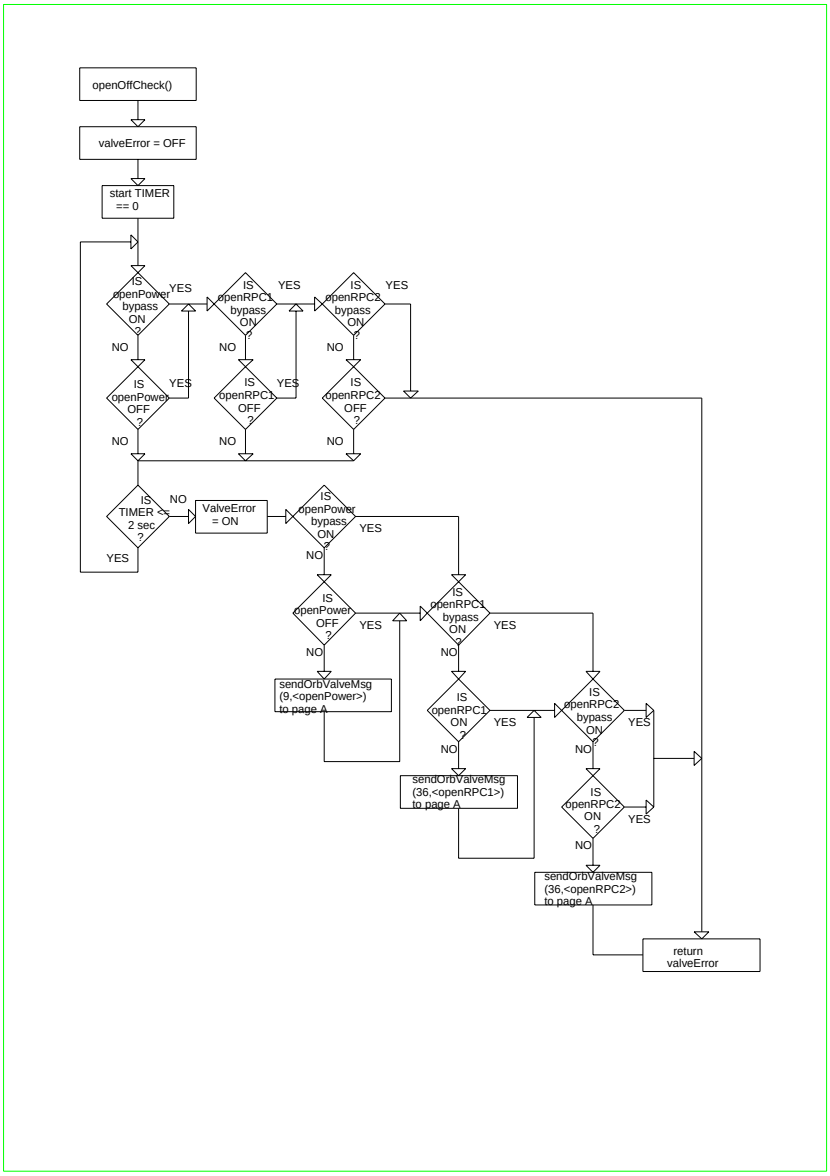


3.1.5.3.8. openOffCheck()

3.1.5.3.8.1. Descriptive Text

This method is used by the valve cycle method to determine whether the open power and RPCs deenergize within 2 seconds. The method prints error messages for any open power or RPC that does not deenergize in the alloted time. The method also returns valveError = ON if errors occur and OFF for a nominal cycle.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

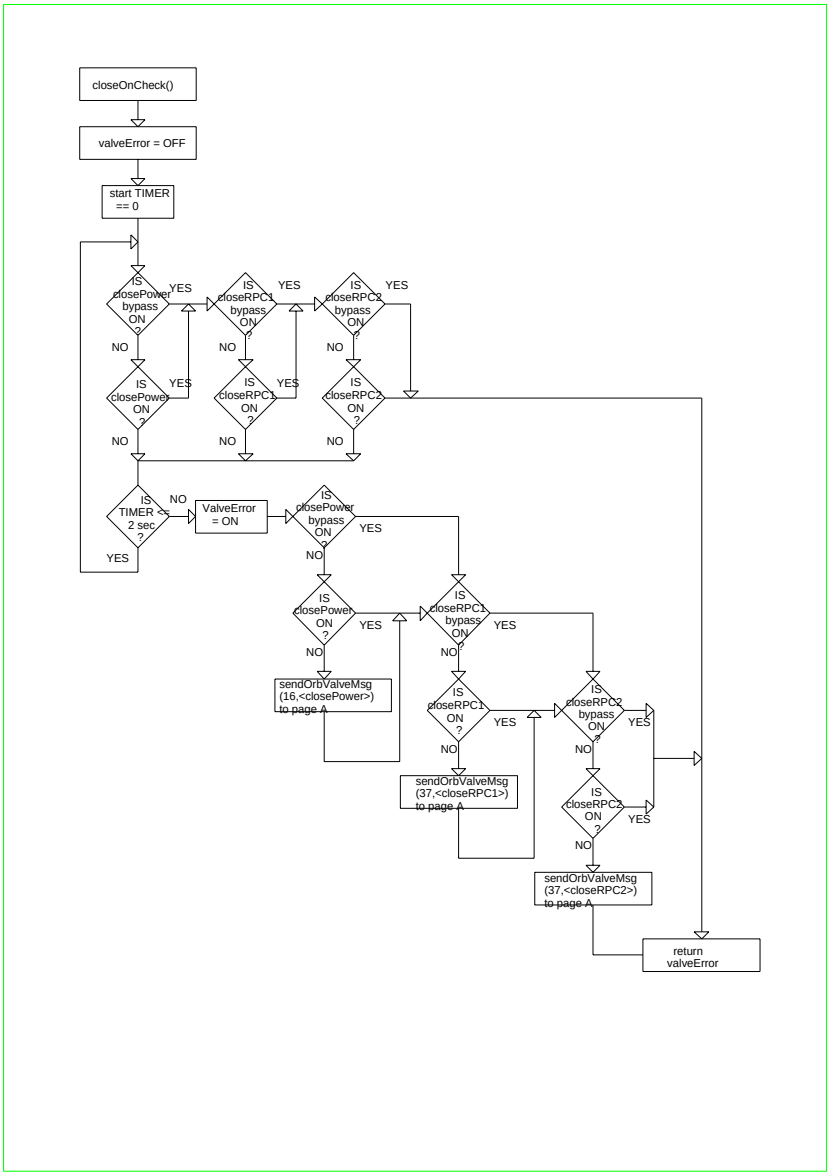


3.1.5.3.9. closeOnCheck()

3.1.5.3.9.1. Descriptive Text

This method is used by the valve cycle method to determine whether the close power and RPCs energize within 2 seconds. The method prints error messages for any close power or RPC that does not energize in the alloted time. The method also returns valveError = ON if errors occur and OFF for a nominal cycle.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

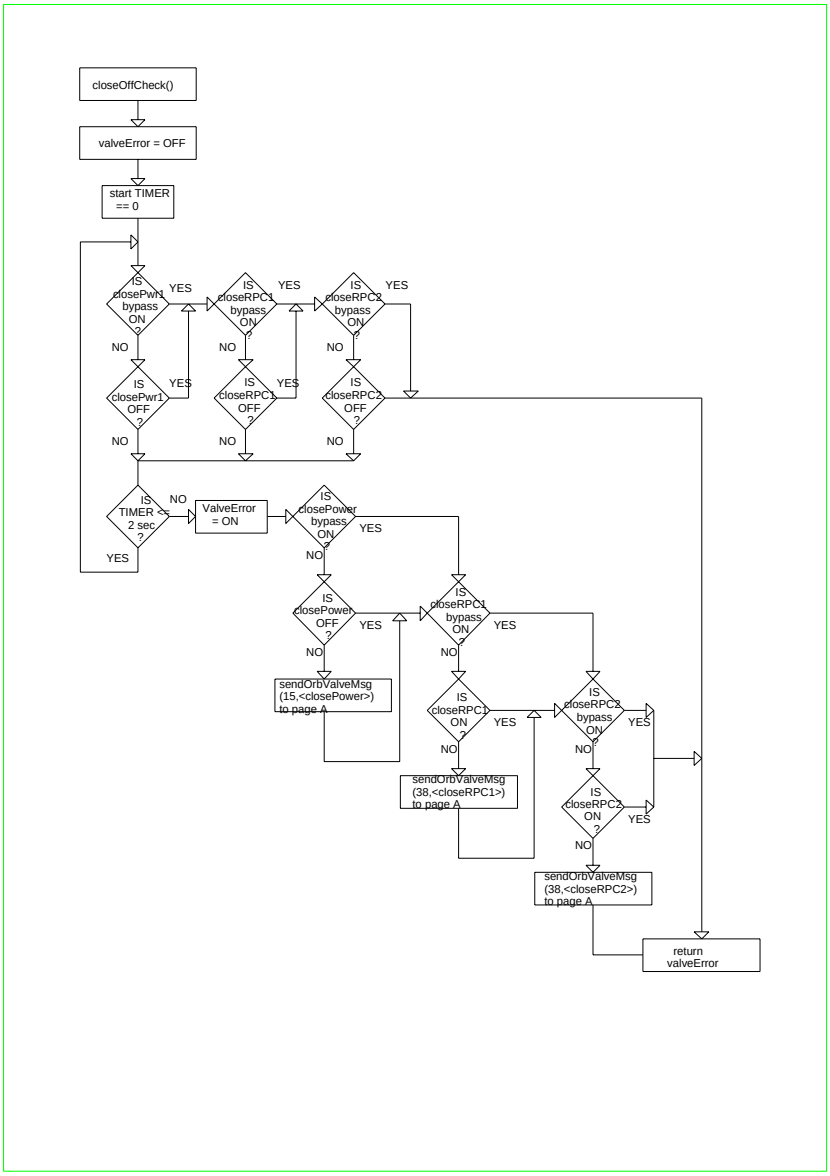


3.1.5.3.10. closeOffCheck()

3.1.5.3.10.1. Descriptive Text

This method is used by the valve cycle method to determine whether the close power and RPCs deenergize within 2 seconds. The method prints error messages for any close power or RPC that does not deenergize in the alloted time. The method also returns valveError = ON if errors occur and OFF for a nominal cycle.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

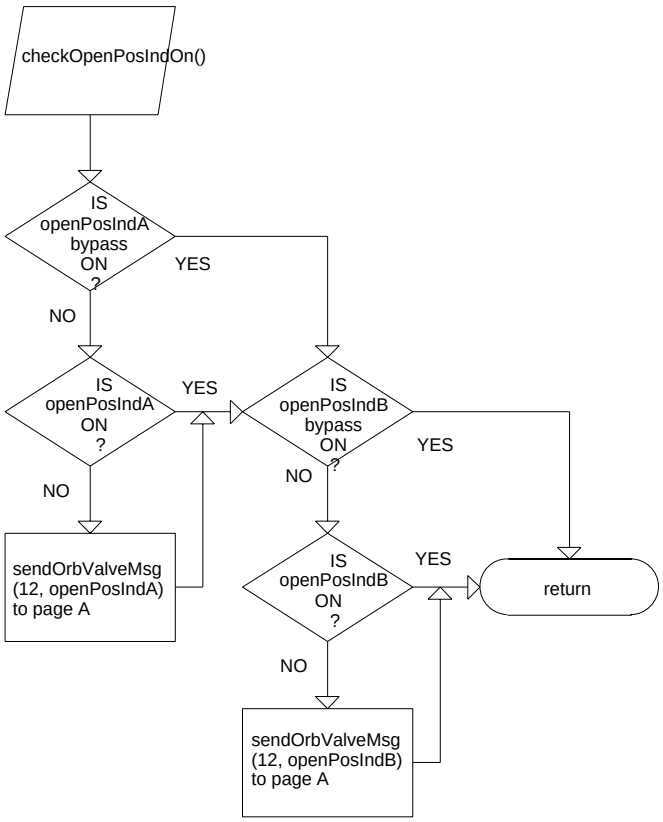


3.1.5.3.11. checkOpenPosIndOn()

3.1.5.3.11.1. Descriptive Text

This method prints error messages for open position indicators are not either ON or bypassed.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

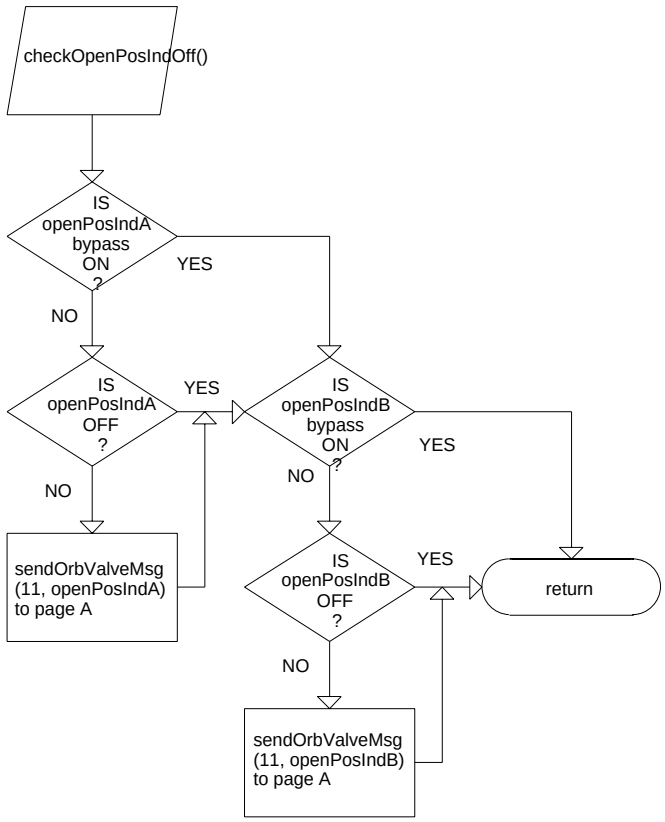


3.1.5.3.12. checkOpenPosIndOff()

3.1.5.3.12.1. Descriptive Text

This method prints error messages for open position indicators are not either OFF or bypassed.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



3.1.5.3.13. determineIndState()

3.1.5.3.13.1. Descriptive Text

This method will determine the valve's position based on its position indicators and associated indicator bypasses. The return value will be an enumerated type with possible values of OPEN, CLOSED, TRANSITION, and ERROR.

3.1.5.3.13.2. Effectivity

Operates on change data.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.5.3.13.3. Sequence

INDICATOR STATE						
open pos ind A	**open pos A byp**	**open pos ind B**	**open pos B byp**	**close pos ind**	**close pos byp**	**Indicator State**
OFF	OFF	OFF	OFF	----	ON	CLOSED
OFF	OFF	OFF	OFF	ON	OFF	CLOSED
OFF	OFF	----	ON	----	ON	CLOSED
OFF	OFF	----	ON	ON	OFF	CLOSED
----	ON	OFF	OFF	ON	OFF	CLOSED
----	ON	----	ON	ON	OFF	CLOSED
----	ON	OFF	OFF	----	ON	CLOSED
----	ON	----	ON	OFF	OFF	OPEN
----	ON	ON	OFF	OFF	OFF	OPEN
----	ON	ON	OFF	----	ON	OPEN
ON	OFF	----	ON	OFF	OFF	OPEN
ON	OFF	----	ON	----	ON	OPEN
ON	OFF	ON	OFF	OFF	OFF	OPEN
ON	OFF	ON	OFF	----	ON	OPEN
----	ON	OFF	OFF	OFF	OFF	TRANSITION
OFF	OFF	OFF	OFF	OFF	OFF	TRANSITION
OFF	OFF	----	ON	OFF	OFF	TRANSITION
OFF	OFF	ON	OFF	OFF	OFF	ERROR
OFF	OFF	ON	OFF	----	ON	ERROR
OFF	OFF	ON	OFF	ON	OFF	ERROR
----	ON	----	ON	----	ON	ERROR
----	ON	ON	OFF	ON	OFF	ERROR
ON	OFF	OFF	OFF	OFF	OFF	ERROR
ON	OFF	OFF	OFF	ON	OFF	ERROR
ON	OFF	----	ON	ON	OFF	ERROR
ON	OFF	ON	OFF	ON	OFF	ERROR
ON	OFF	OFF	OFF	----	ON	ERROR

3.1.5.3.14. determineOpenSolCmdState()

3.1.5.3.14.1. Descriptive Text

This method will determine the open solenoid's commanded state based on the two open RPCs, the open power, and the associated bypasses. The return value will be an enumerated
type with possible values of ENERGIZED, DEENERGIZED and ERROR.

3.1.5.3.14.2. Effectivity

Operates on change data.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.1.5.3.14.3. Sequence

Open Solenoid Command state

open Power Ind	open power bypass	open RPC_A	open RPC_A Bypass	open RPC_B	open RPC_B Bypass	OP CMD State
OFF	OFF	OFF	OFF	OFF	OFF	deenergized
OFF	OFF	OFF	OFF	----	ON	deenergized
OFF	OFF	----	ON	OFF	OFF	deenergized
OFF	OFF	----	ON	----	ON	deenergized
----	ON	OFF	OFF	OFF	OFF	deenergized
----	ON	OFF	OFF	----	ON	deenergized
----	ON	----	ON	OFF	OFF	deenergized
----	ON	----	ON	ON	OFF	energized
----	ON	ON	OFF	----	ON	energized
----	ON	ON	OFF	ON	OFF	energized
ON	OFF	----	ON	----	ON	energized
ON	OFF	----	ON	ON	OFF	energized
ON	OFF	ON	OFF	----	ON	energized
ON	OFF	ON	OFF	ON	OFF	energized
OFF	OFF	OFF	OFF	ON	OFF	error
OFF	OFF	----	ON	ON	OFF	error
OFF	OFF	ON	OFF	OFF	OFF	error
OFF	OFF	ON	OFF	----	ON	error
OFF	OFF	ON	OFF	ON	OFF	error
----	ON	OFF	OFF	ON	OFF	error
----	ON	----	ON	----	ON	error
----	ON	ON	OFF	OFF	OFF	error
ON	OFF	OFF	OFF	OFF	OFF	error
ON	OFF	OFF	OFF	----	ON	error
ON	OFF	OFF	OFF	ON	OFF	error
ON	OFF	----	ON	OFF	OFF	error
ON	OFF	ON	OFF	OFF	OFF	error

3.1.5.3.15. determineCloseSolCmdState()

3.1.5.3.15.1. Descriptive Text

This method will determine the close solenoid's commanded state based on the two close RPCs, the close power, and the associated bypasses. The return value will be an enumerated
type with possible values of ENERGIZED, DEENERGIZED and ERROR.

3.1.5.3.15.2. Effectivity

Operates on change data.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.1.5.3.15.3. Sequence

Close Solenoid Command State

close power	close power bypass	close RPC_A	close RPC_A Bypass	close RPC_B	close RPC_B Bypass	CL CMD State
OFF	OFF	OFF	OFF	OFF	OFF	deenergized
OFF	OFF	OFF	OFF	----	ON	deenergized
OFF	OFF	----	ON	OFF	OFF	deenergized
OFF	OFF	----	ON	----	ON	deenergized
----	ON	OFF	OFF	OFF	OFF	deenergized
----	ON	OFF	OFF	----	ON	deenergized
----	ON	----	ON	OFF	OFF	deenergized
----	ON	----	ON	ON	OFF	energized
----	ON	ON	OFF	----	ON	energized
----	ON	ON	OFF	ON	OFF	energized
ON	OFF	----	ON	----	ON	energized
ON	OFF	----	ON	ON	OFF	energized
ON	OFF	ON	OFF	----	ON	energized
ON	OFF	ON	OFF	ON	OFF	energized
OFF	OFF	OFF	OFF	ON	OFF	error
OFF	OFF	----	ON	ON	OFF	error
OFF	OFF	ON	OFF	OFF	OFF	error
OFF	OFF	ON	OFF	----	ON	error
OFF	OFF	ON	OFF	ON	OFF	error
----	ON	OFF	OFF	ON	OFF	error
----	ON	----	ON	----	ON	error
----	ON	ON	OFF	OFF	OFF	error
ON	OFF	OFF	OFF	OFF	OFF	error
ON	OFF	OFF	OFF	----	ON	error
ON	OFF	OFF	OFF	ON	OFF	error
ON	OFF	----	ON	OFF	OFF	error
ON	OFF	ON	OFF	OFF	OFF	error

3.1.5.3.16. determineCmdState()//Phase 3

3.1.5.3.16.1. Descriptive Text

This method will determine the valve's overall commanded state based on closeSolCmdState and openSolCmdState. The return value will be an enumerated type with possible values of CMD_CLOSED, CMD_OPEN, UNCOMMANDED, CMD_SNUBBED, and ERROR.

3.1.5.3.16.2. Effectivity

Operates on change data.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.5.3.16.3. Sequence

Combined Command State			
OP_CMDS \ CL_CMDS	ENERGIZED	DEENERGIZED	ERROR
DEENERGIZED	CMD_CLOSED	UNCOMMANDED	ERROR
ENERGIZED	CMD_SNUBBED	CMD_OPEN	ERROR
ERROR	ERROR	ERROR	ERROR

3.1.5.3.17. determineState()//Phase 3

3.1.5.3.17.1. Descriptive Text

Determine state performs a logical comparison of the valve indicator state and command state to determine an overall state of the valve.

3.1.5.3.17.2. Sequence

State (overall)					
pos \ cmd	cmd closed	cmd open	cmd snubbed	uncommanded	error
closed	closed	close error	closed_error	closed_error	closed_error
open	open error	open	open_error	open_error	open_error
transition	transition	transition	error	error	error
error	error	error	error	error	error

3.1.5.3.18. getIndState()

3.1.5.3.18.1. Descriptive Text

This method will return the Enumerated Indicator State of the valve

3.1.5.3.18.2. Sequence

```
getIndState()
{
        return indState;
}
```

3.1.5.3.19. getOpenSolCmdState()

3.1.5.3.19.1. Descriptive Text

This method will return the Enumerated Command State of the valve.

3.1.5.3.19.2. Sequence

```
getOpenSolCmdState()
{
        return openSolCmdState;
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.5.3.20. getCloseSolCmdState()

3.1.5.3.20.1. Descriptive Text

This method will return the Enumerated Command State of the valve.

3.1.5.3.20.2. Sequence

```
getCloseSolCmdState()
{
        return closeSolCmdState;
}
```

3.1.5.3.21. getCmdState()

3.1.5.3.21.1. Descriptive Text

This method will return the Enumerated Command State of the valve.

3.1.5.3.21.2. Sequence

```
getCmdState()
{
        return cmdState;
}
```

3.1.5.3.22. getState()

3.1.5.3.22.1. Descriptive Text

This method will return the overall Enumerated State of the valve.

3.1.5.3.22.2. Sequence

```
getState()
{
        return state;
}
```

3.1.5.3.23. autocycle()

3.1.5.3.23.1. Descriptive Text

3.1.5.3.23.2. Sequence

```
{

        If IndState == CLOSED
                open();
                If valveCycleError is ON, set autoValveCheckoutError = ON
                close();
                If valveCycleError is ON, set autoValveCheckoutError = ON


        If IndState == OPEN
                close();
                If valveCycleError is ON, set autoValveCheckoutError = ON
                open();
                If valveCycleError is ON, set autoValveCheckoutError = ON

        Else
                sendOrbValveMsg(5, autocycle)
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

Return autoValveCheckoutError;

}

3.1.5.3.24. activateConstraints()

3.1.5.3.24.1. Descriptive Text

This method will activate constraint monitoring of all unbypassed measurement of this valve.

3.1.5.3.24.2. Sequence

```
activateConstraints()
        {
        if(<openPosIndA>.Byp == OFF)
                {
                activate constraint monitoring for <openPosIndA>
                }
        if(<openPosIndB>.Byp == OFF)
                {
                activate constraint monitoring for <openPosIndB>
                }
        if(<openPwr>.Byp == OFF)
                {
                activate constraint monitoring for <openPwr>
                }
        if(<openRPC1>.Byp == OFF)
                {
                activate constraint monitoring for <openRPC1>
                }
        if(<openRPC2>.Byp == OFF)
                {
                activate constraint monitoring for <openRPC2>
                }
        if(<closePwr>.Byp == OFF)
                {
                activate constraint monitoring for <closePwr>
                }
        if(<closePosInd>.Byp == OFF)
                {
                activate constraint monitoring for <closePosInd>
                }
        if(<closeRPC1>.Byp == OFF)
                {
                activate constraint monitoring for <closeRPC1>
                }
        if(<closeRPC2>.Byp == OFF)
                {
                activate constraint monitoring for <closeRPC2>
                }
```

3.1.5.3.25. inhibitConstraints()

3.1.5.3.25.1. Descriptive Text

This method will inhibit constraint monitoring of all measurements related to this valve.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.5.3.25.2. Sequence

inhibitConstraints()

{

inhibit constraint monitoring of the following:

<openPosIndA>

<openPosIndB>

<openRPC1>

<openRPC2>

<openPwr>

<closePosInd>

<closePwr>

<closeRPC1>

<closeRPC2>

}

3.1.5.3.26. setOpenConstraints()

3.1.5.3.26.1. Descriptive Text

This method changes the constraint criteria of the measurements associated with this valve to those expected with the valve in the open position.

3.1.5.3.26.2. Sequence

void setOpenConstraints()

{

CHANGE <openPwr> CONSTRAINT CRITERIA TO ON;

CHANGE <openPosIndA> CONSTRAINT CRITERIA TO ON;

CHANGE <openPosIndB> CONSTRAINT CRITERIA TO ON;

CHANGE <openRPC1> CONSTRAINT CRITERIA TO ON;

CHANGE <openRPC2> CONSTRAINT CRITERIA TO ON;

CHANGE <closePwr> CONSTRAINT CRITERIA TO OFF;

CHANGE <closePosInd> CONSTRAINT CRITERIA TO OFF;

CHANGE <closeRPC1> CONSTRAINT CRITERIA TO OFF;

CHANGE <closeRPC2> CONSTRAINT CRITERIA TO OFF;

}

3.1.5.3.27. setOpenPosConstraints()

3.1.5.3.27.1. Descriptive Text

This method changes the constraint criteria of the measurements associated with this valve to those expected with the valve in the open position with no power (pneumatics) applied.

3.1.5.3.27.2. Sequence

setOpenPosConstraints()

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
{
        CHANGE <openPosIndA>  CONSTRAINT CRITERIA TO ON;

        CHANGE <openPosIndB> CONSTRAINT CRITERIA TO ON;

          CHANGE <closePosInd> CONSTRAINT CRITERIA TO OFF;

          CHANGE <openPwr> CONSTRAINT CRITERIA to OFF;

          CHANGE <closePwr> CONSTRAINT CRITERIA to OFF;

          CHANGE <openRPC1> CONSTRAINT CRITERIA TO OFF;

        CHANGE <openRPC2>  CONSTRAINT CRITERIA TO OFF;

        CHANGE <closeRPC1> CONSTRAINT CRITERIA TO OFF;

          CHANGE <closeRPC2> CONSTRAINT CRITERIA TO OFF;


}
```

3.1.5.3.28. setClosedConstraints()

3.1.5.3.28.1. Descriptive Text

This method changes the constraint criteria of the measurements associated with this valve to those expected with the valve in the closed position.

3.1.5.3.28.2. Sequence

```
void setCloseConstraints()
{
          CHANGE <openPwr> CONSTRAINT CRITERIA TO OFF;

        CHANGE <openPosIndA>  CONSTRAINT CRITERIA TO OFF;

        CHANGE <openPosIndB> CONSTRAINT CRITERIA TO OFF;

          CHANGE <openRPC1> CONSTRAINT CRITERIA TO OFF;

        CHANGE <openRPC2>  CONSTRAINT CRITERIA TO OFF;

        CHANGE <closePwr> CONSTRAINT CRITERIA TO ON;

          CHANGE <closePosInd> CONSTRAINT CRITERIA TO ON;

        CHANGE <closeRPC1> CONSTRAINT CRITERIA TO ON;

          CHANGE <closeRPC2> CONSTRAINT CRITERIA TO ON;
}
```

3.1.5.3.29. setClosePosConstraints()

3.1.5.3.29.1. Descriptive Text

This method changes the constraint criteria of the measurements associated with this valve to those expected with the valve in the closed position with no power (pneumatics) applied.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.5.3.29.2. Sequence

```
setClosePosConstraints()
        {
                CHANGE <openPosIndA>  CONSTRAINT CRITERIA TO OFF;

                CHANGE <openPosIndB> CONSTRAINT CRITERIA TO OFF;

                  CHANGE <closePosInd> CONSTRAINT CRITERIA TO ON;

                  CHANGE <openPwr> CONSTRAINT CRITERIA to OFF;

                  CHANGE <closePwr> CONSTRAINT CRITERIA to OFF;

                  CHANGE <openRPC1> CONSTRAINT CRITERIA TO OFF;

                CHANGE <openRPC2>  CONSTRAINT CRITERIA TO OFF;

                CHANGE <closeRPC1> CONSTRAINT CRITERIA TO OFF;

                  CHANGE <closeRPC2> CONSTRAINT CRITERIA TO OFF;

        }
```

3.1.5.3.30. verifyCommandsOff()

3.1.5.3.30.1. Descriptive Text

This function will verify that both the open and close commands are OFF

3.1.5.3.30.2. Sequence

```
verifyCommandsOff()
{
        open command status1 = byteTest4 <Open CommandA>
                verify  open command status1 == OFF
                else send message ",valveID, open command;<openCMDA>; " is ON, should be OFF" to Page A

        open command status2 = byteTest4 <Open CommandB>
                verify  open command status2 == OFF
                else send message ",valveID, open command;<openCMDB>; " is ON, should be OFF" to Page A

        open command status3 = byteTest4 <Open CommandC>
                verify  open command status3 == OFF
                else send message ",valveID, open command;<openCMDC>; " is ON, should be OFF" to Page A


        close command status1 = byteTest4 <Close CommandA>
                verify close command status1 == OFF
                else print message ",valveID, close command;<closeCMDA>; " is ON, should be OFF" to Page A

        close command status2 = byteTest4 <Close CommandB>
                verify close command status2 == OFF
                else print message ",valveID, close command;<closeCMDB>; " is ON, should be OFF" to Page A

        close command status3 = byteTest4 <Close CommandC>
                verify close command status3 == OFF
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
                else print message ",valveID, close command;<closeCMDC>; " is ON, should be OFF" to Page A

        {
                verify  open command status1, 2, & 3 == OFF
                &
                verify close command status1, 2, & 3 == OFF

                return TRUE

                ELSE return FALSE
        }
}
```

3.1.5.3.31. verifyTminus3BT4()//Phase 3

3.1.5.3.31.1. Description

This method performs a byte test 4 of prevalve commands to verify they are in a proper T-3 hour configuration. LH2 prevalves are closed (with recirc pumps active).

3.1.5.3.31.2. Sequence

begin sequence;

```
set commanderrorcheck OFF
{
perform ByteTest4 of <openCMDA>
        verify ByteTest4 = OFF
        else send message to page A
                "LH2 Prevalve" <openCMDA> "is not in T-3 hour configuration."
                and set commanderrorcheck ON

perform ByteTest4 of <openCMDB>
        verify ByteTest4 = OFF
        else send message to page A
                "LH2 Prevalve" <openCMDB> "is not in T-3 hour configuration."
                and set commanderrorcheck ON

perform ByteTest4 of <openCMDC>
        verify ByteTest4 = OFF
        else send message to page A
                "LH2 Prevalve" <openCMDC> "is not in T-3 hour configuration."
                and set commanderrorcheck ON
}
{
perform ByteTest4 of <closeCMDA>
        verify ByteTest4 = ON
        else send message to page A
                "LH2 Prevalve" <closeCMDA> "is not in T-3 hour configuration."
                and set commanderrorcheck ON

perform ByteTest4 of <closeCMDB>
        verify ByteTest4 = ON
        else send message to page A
                "LH2 Prevalve" <closeCMDB> "is not in T-3 hour configuration."
                and set commanderrorcheck ON
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
perform ByteTest4 of <closeCMDC>
        verify ByteTest4 = ON
        else send message to page A
                "LH2 Prevalve" <closeCMDC> "is not in T-3 hour configuration."
                and set commanderrorcheck ON

}
return commanderrorcheck

end sequence;
```

3.1.5.3.32. verifyTminus6BT4()//Phase 3

3.1.5.3.32.1. Description

This method performs a byte test 4 of fill and drain valve commands to verify they are in a proper T-3 hour configuration. For both instances of this class, open commands should be ON, close commands should be OFF.

3.1.5.3.32.2. Sequence

```
begin sequence;

set commanderrorcheck OFF
{
perform ByteTest4 of <openCMDA>
        verify ByteTest4 = ON
        else send message to page A
                "LH2 Prevalve" <openCMDA> "is not in T-3 hour configuration."
                and set commanderrorcheck ON

perform ByteTest4 of <openCMDB>
        verify ByteTest4 = ON
        else send message to page A
                "LH2 Prevalve" <openCMDB> "is not in T-3 hour configuration."
                and set commanderrorcheck ON

perform ByteTest4 of <openCMDC>
        verify ByteTest4 = ON
        else send message to page A
                "LH2 Prevalve" <openCMDC> "is not in T-3 hour configuration."
                and set commanderrorcheck ON
}
{
perform ByteTest4 of <closeCMDA>
        verify ByteTest4 = OFF
        else send message to page A
                "LH2 Prevalve" <closeCMDA> "is not in T-3 hour configuration."
                and set commanderrorcheck ON

perform ByteTest4 of <closeCMDB>
        verify ByteTest4 = OFF
        else send message to page A
                "LH2 Prevalve" <closeCMDB> "is not in T-3 hour configuration."
                and set commanderrorcheck ON
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
perform ByteTest4 of <closeCMDC>
        verify ByteTest4 = OFF
        else send message to page A
                "LH2 Prevalve" <closeCMDC> "is not in T-3 hour configuration."
                and set commanderrorcheck ON

}
return commanderrorcheck

end sequence;
```

3.1.5.4. User Interfaces

3.1.5.5. Safety Requirements

3.1.6. MPS_Valve_ORB_4inchdisc

3.1.6.1. Description

MPS_Valve_ORB_4inchdisc is a bi-stable valve with 1 open and 1 close commands, 1 open and 1 close position indications, and 1 open and 1 close power indications. Functionally, the valve provides a flow path between the Orbiter and ET during load. Additionally, the valve provides 4 inch topping line/ET isolation capability during pad abort operations and during leak isolation procedures. Due to the integral design of the valve requiring a mated ORB/ET disconnect, the valve requires the ORB/ET to be in the mated configuration for cycling (open/close), but the actuator may be cycled in the OPF (with no resultant change in valve position). The Valve class inherits from the MPS_Valve_ORB Class. The valve has an association with the MPS_Helium_System class for status of the MPS 750 pneumatic system. For integrated operations, this valve class has an association with the MPS_System for determination of the cryo load stautus via a Cryo Flag pseudo.

3.1.6.1.1. Instance Index

Instance	Display User	Other Users
MPS_PD3	MPS	

3.1.6.1.2. Class Diagram

<insert class diagram here>

3.1.6.1.3. Attributes and Aggregations

Attribute	Data Type	Initial Value	Publish / Persistence	Client Access	Description
openPosInd	DM_ON_OFF			R	
closePosInd	DM_ON_OFF			R	
openCmd	DS_ON_OFF			R,W	
closeCmd	DS_ON_OFF			R,W	
openPwr	DM_ON_OFF			R	

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

closePwr	DM_ON_OFF			R	
openSolCmdState	MPS_ORB_SOL_STATE		PUB	R	
closeSolCmdState	MPS_ORB_SOL_STATE		PUB	R	

3.1.6.1.4. Methods Index

Operation	**Description**
open()	This method performs necessary functions to cycle the mated configuration valve to the open position (method is only available if at VAB/PAD).
extendActuator()	This method performs necessary functions to cycle valve actuator to the extended position (method is only available in OPF).
close()	This method performs necessary functions to cycle the mated configuration valve to the open position (method is only available if at VAB/PAD).
retractActuator()	This method performs necessary functions to cycle valve actuator to the retracted position (method is only available in OPF).
energizeOpenSolenoid()	This method energizes open solenoid. Method includes a check of MPS pneumatics status, and present valve indicator state to determine consequence of energizing valve. The method then runs the appropriate open() method if the operator chooses to continue via prompt response.
deenergizeOpenSolenoid()	This method de-energizes opening solenoid. Method includes a check of MPS pneumatics status, and present valve indicator state to determine consequence of energizing valve. The method then runs the appropriate close() method if the operator chooses to continue via prompt response.
energizeCloseSolenoid()	This method energizes closing solenoid. Method includes a check of MPS pneumatics status, and present valve indicator state to determine consequence of energizing valve. The method then runs the appropriate close() method if the operator chooses to continue via prompt response..
deenergizeCloseSolenoid()	This method de-energizes closing solenoid. Method includes a check of MPS pneumatics status, and present valve indicator state to determine consequence of energizing valve. The method then runs the appropriate open() method if the operator chooses to continue via prompt response.
determineState()//Phase 3	This method determines over-all valve state based on position and command states and will return either: OPEN, CLOSED, OPEN_ERROR, CLOSE_ERROR, and TRANSITION..
DetermineCmdState()//Phase 3	This method determines over-all valve command state based on open and close solenoid command states and will return either:

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

	COMMANDED_OPEN, COMMANDED_CLOSED, UNCOMMANDED, COMMANDED_SNUBBED, and ERROR.
determineIndState()	This method will determine the position state of the valve based on all-of-all unbypassed open/close indicators and will return either: OPEN CLOSED TRANSITION ERROR
determineOpSolCmdState()	This method will determine the command state of the opening solenoid valve based on valve power indicators and will return either: ENERGIZED DEENERGIZED ERROR
determineClSolCmdState()	This method will determine the command state of the opening solenoid valve based on valve power indicators and will return either: ENERGIZED DEENERGIZED ERROR
getIndState()	This method will return valve enumerated indicator state.
getOpSolCmdState()	This method will return valve opening solenoid enumerated command state.
getClSolCmdState()	This method will return valve closing solenoid enumerated command state.
verifyCmdsOff()	This method will verify no commands are issued.
setCloseConstraints()	This method will set-up Constraints for a closed valve.
setClosePosConstraints()	This method will set-up Constraints for a closed bistable valve.
setOpenConstraints()	This method will set-up Constraints for an open valve.
activateConstraints()	This method will activate Constraints for either the close or open positions when in standby mode (uncommanded present state).
inhibitConstraints()	This method will inhibit Constraint notification during valve cycle operations.
emergencyClose()//Phase 3	This method will perform emergency closure of PD3 during cryo operations. PAD method required to support load..
verifyTminus3BT4//Phase 3	This method does a byte test 4 of valve commands to verify T-3 hour configuration. PAD method required to support load.
verifyTminus6BT4//Phase 3	This method does a byte test 4 of valve commands to verify T-6 hour configuration. PAD method required to support load.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.6.1.5. Associations

Association	Type	Class	Description
MPS_Hydrogen_SubSystem			LH2CryoFlag status.
MPS_Helium_SubSystem			MPS 750 pneumatic status.
Orbiter Location			

3.1.6.2. State Description

3.1.6.2.1. State Diagram

<insert state diagram here>

3.1.6.2.2. State Table

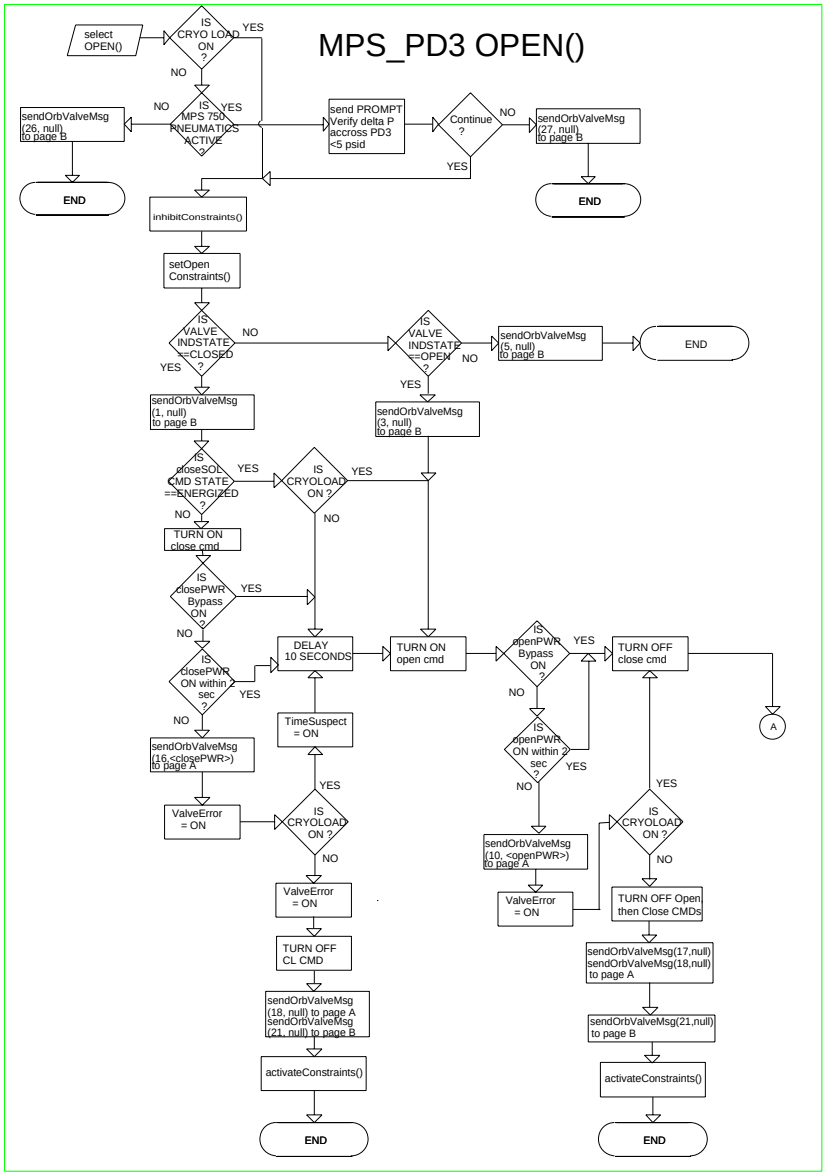
3.1.6.3. Operations

3.1.6.3.1. open()

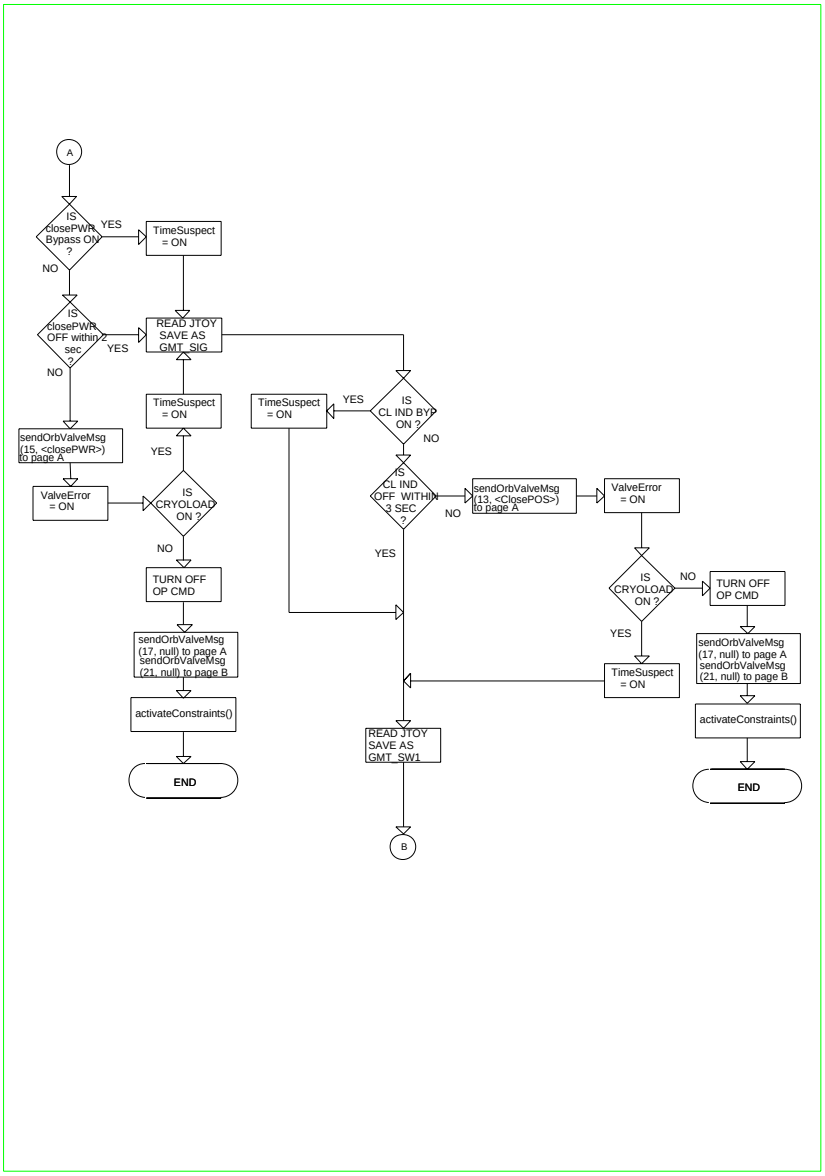
3.1.6.3.1.1. Descriptive Text

This method performs cycling of the bistable LH2 4" LH2 disconnect valve to the open position (requires being mated to the ET). The flow chart provides the necessary logic for valve cycles, including checks for snubbing pressure, proper response of power indications, and proper response of position indications. The operator is to be prompted to verify less than 5 psi differential pressure accrossed PD3 prior to valve cycling open. The valve cycles in a fail safe mode and 'wimps-out' (all command removed) at the first indication of hardware failure, unless the orbiter is in cryogenic load. If in cryo load, the valve cycles in a 'damn-the-torpedos' mode and completes all steps necessary to achieve the valve cycle, regardless of failures which are detected during the cycle. Valve switch to switch timing is also calculated. ValveError and TimeSuspect are initialized to OFF and Valve cycle times are set to 0 prior to starting valve cycle methods. If ValveError gets set ON during method run, cycle time FD data health should be set to invalid to aid in identifying suspect cycle data in trend analysis.

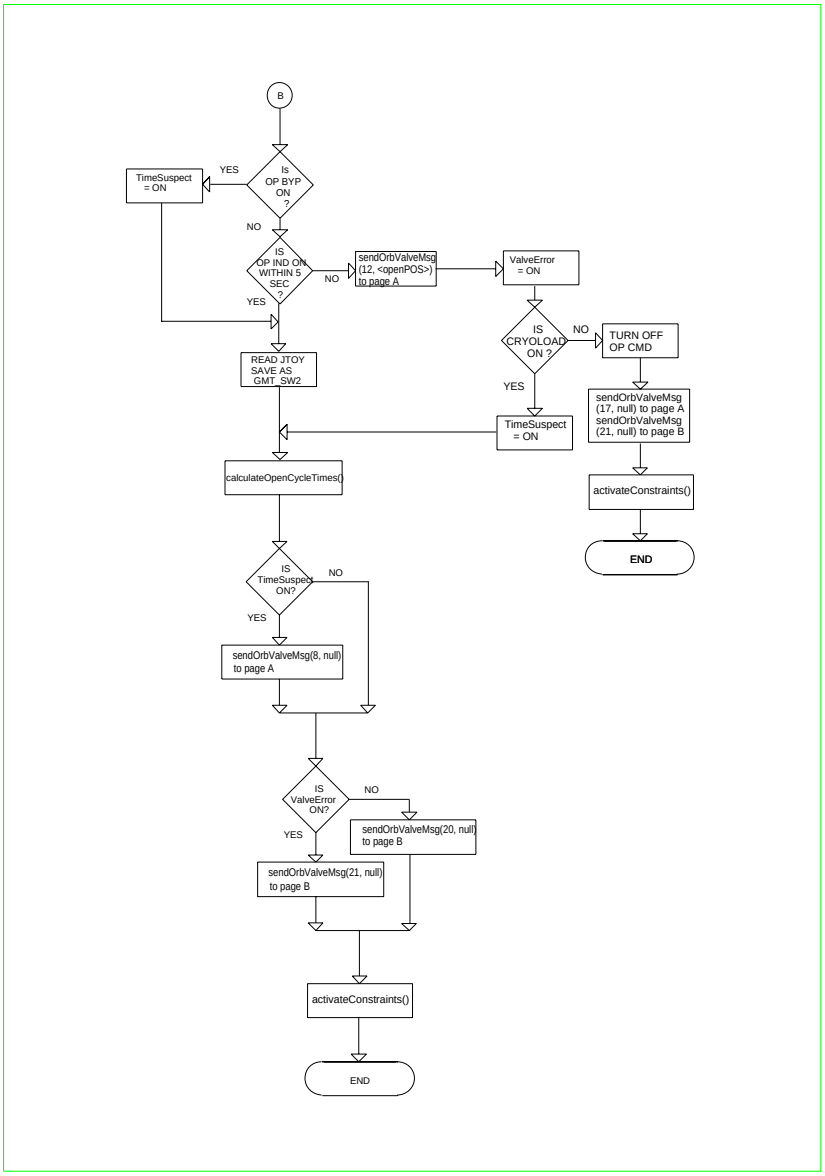
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

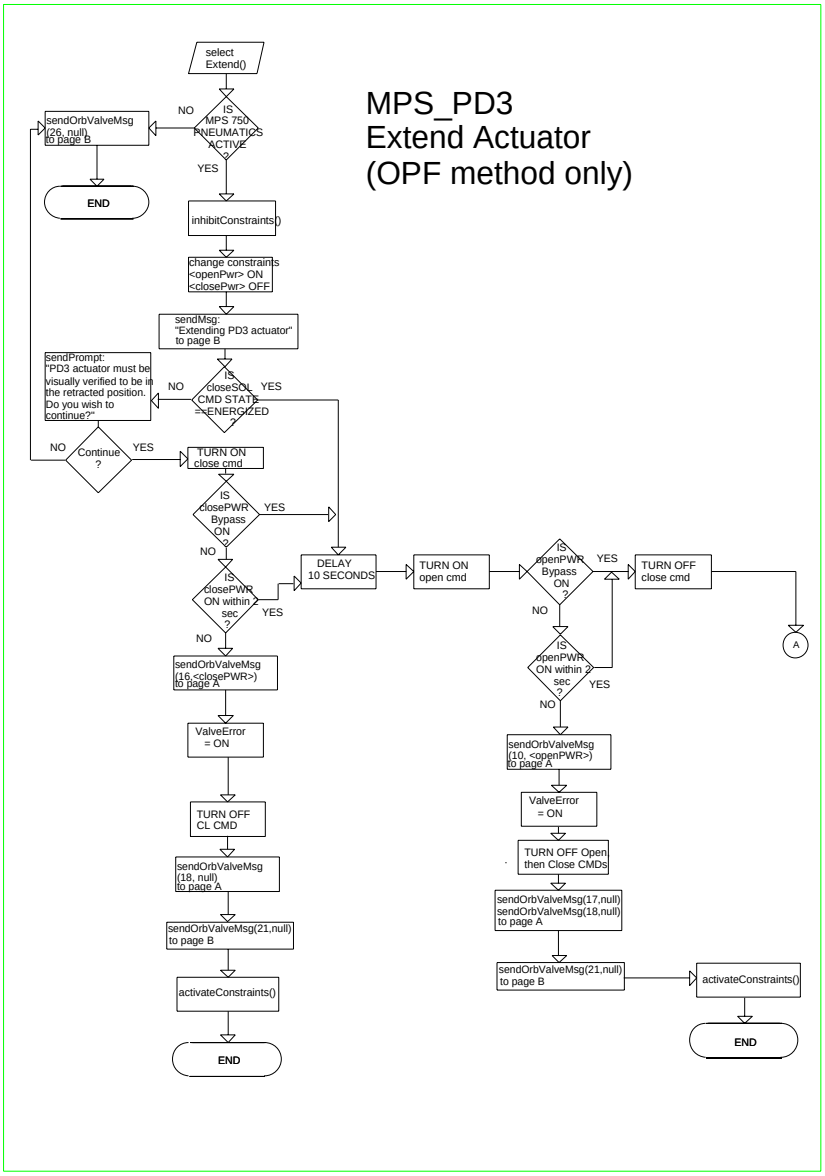


3.1.6.3.2. extendActuator()

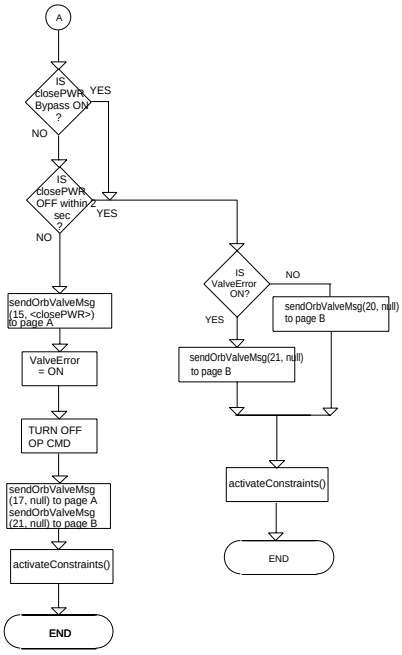
3.1.6.3.2.1. Descriptive Text

This method performs cycling of the bistable LH2 4" LH2 disconnect valve actuator to the extended position (in the OPF). The flow chart provides the necessary logic for valve cycles, including checks for snubbing pressure, and proper response of power indications (since the disconnect cannot be cycled in the OPF, position indications do not change). The valve cycles in a fail safe mode and 'wimps-out' (all command removed) at the first indication of hardware failure. ValveError and is initialized to OFF prior to starting valve cycle methods.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

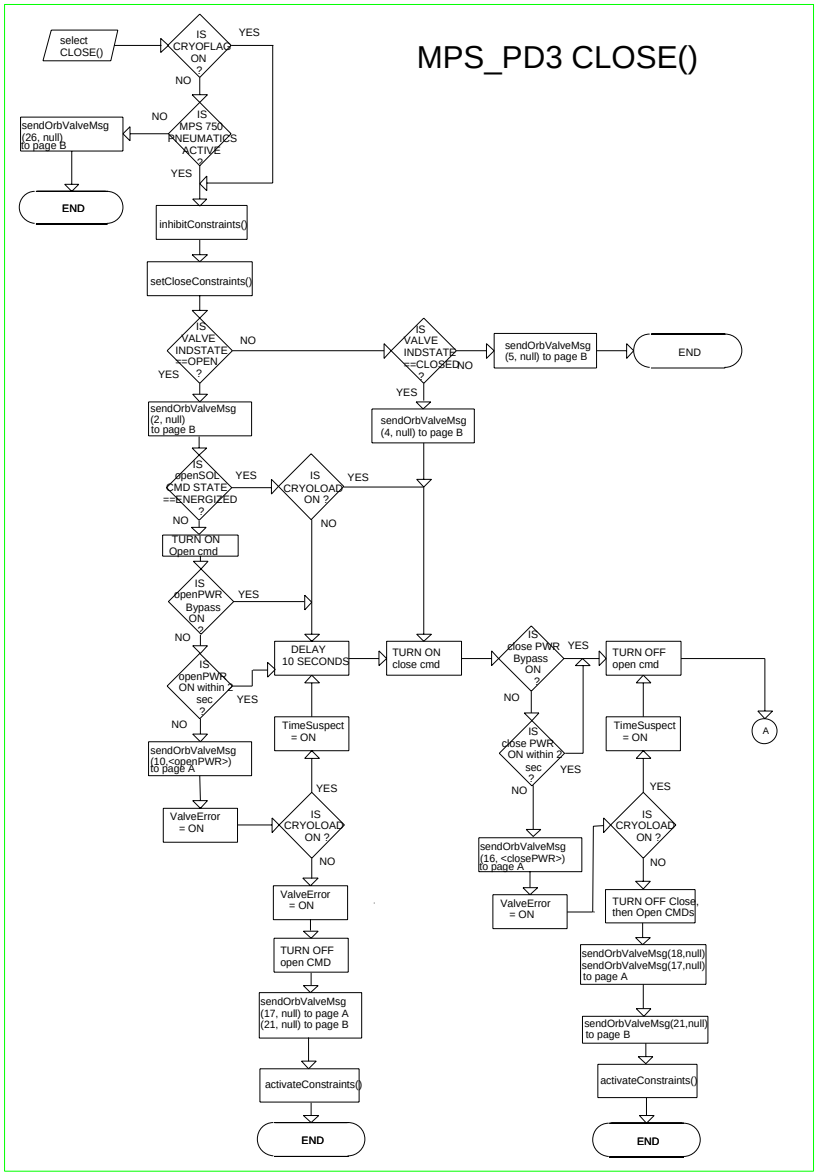


3.1.6.3.3. close()

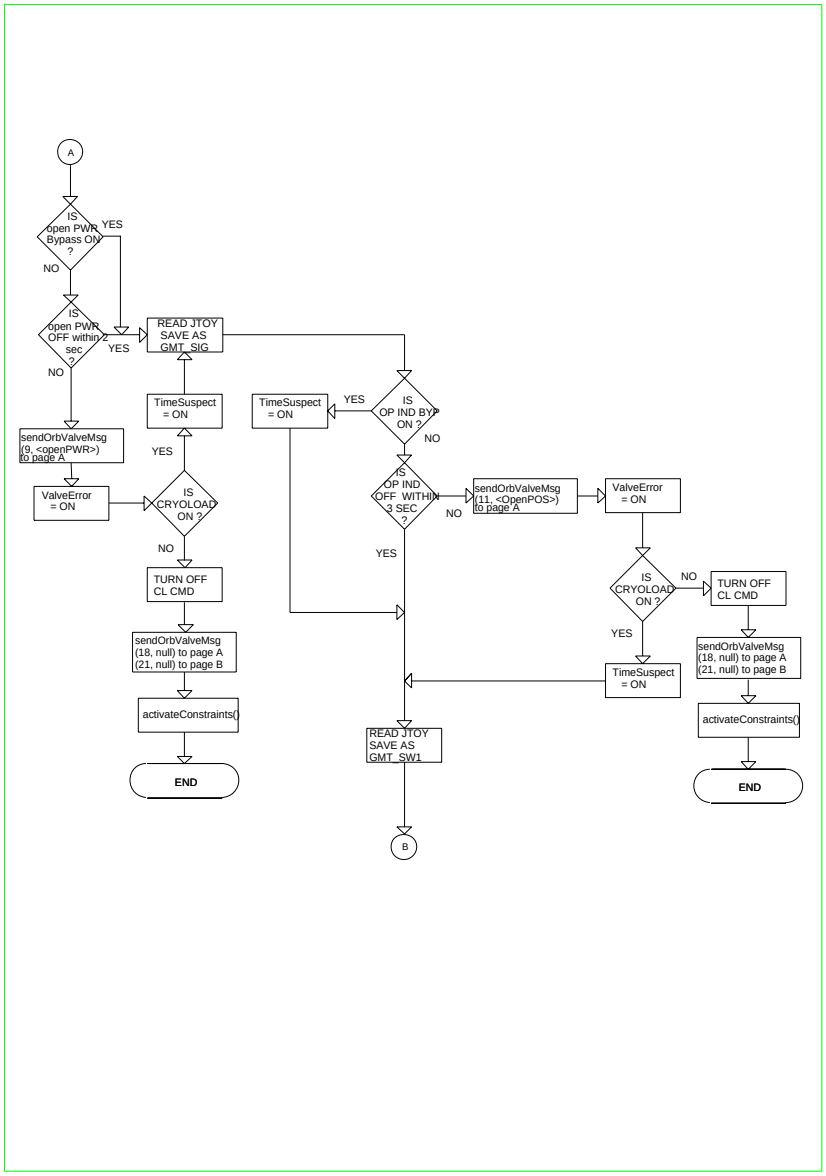
3.1.6.3.3.1. Descriptive Text

This method performs cycling of the bistable LH2 4" LH2 disconnect valve to the closed position (if mated to the ET) or cycles the actuator to the retracted position (in the OPF). The flow chart provides the necessary logic for valve cycles, including checks for snubbing pressure, proper response of power indications, and proper response of position indications. The valve cycles in a fail safe mode and 'wimps-out' (all command removed) at the first indication of hardware failure, unless the orbiter is in cryogenic load. If in cryo load, the valve cycles in a 'damn-the-torpedoes' mode and completes all steps necessary to achieve the valve cycle, regardless of failures which are detected during the cycle. Valve signal to switch and switch to switch timing is also calculated. ValveError and TimeSuspect are initialized to OFF and Valve cycle times are set to 0 prior to starting valve cycle methods. If ValveError gets set ON during method run, cycle time FD data health should be set to invalid to aid in identifying suspect cycle data in trend analysis.

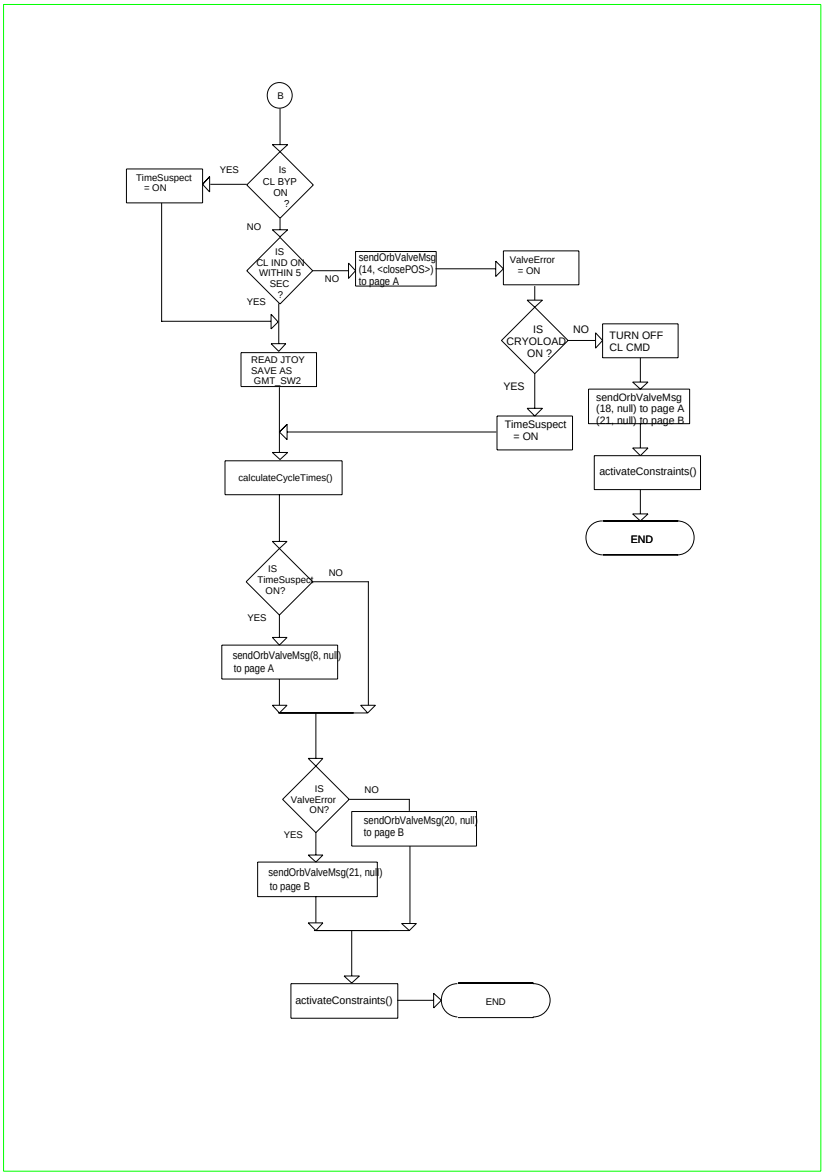
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

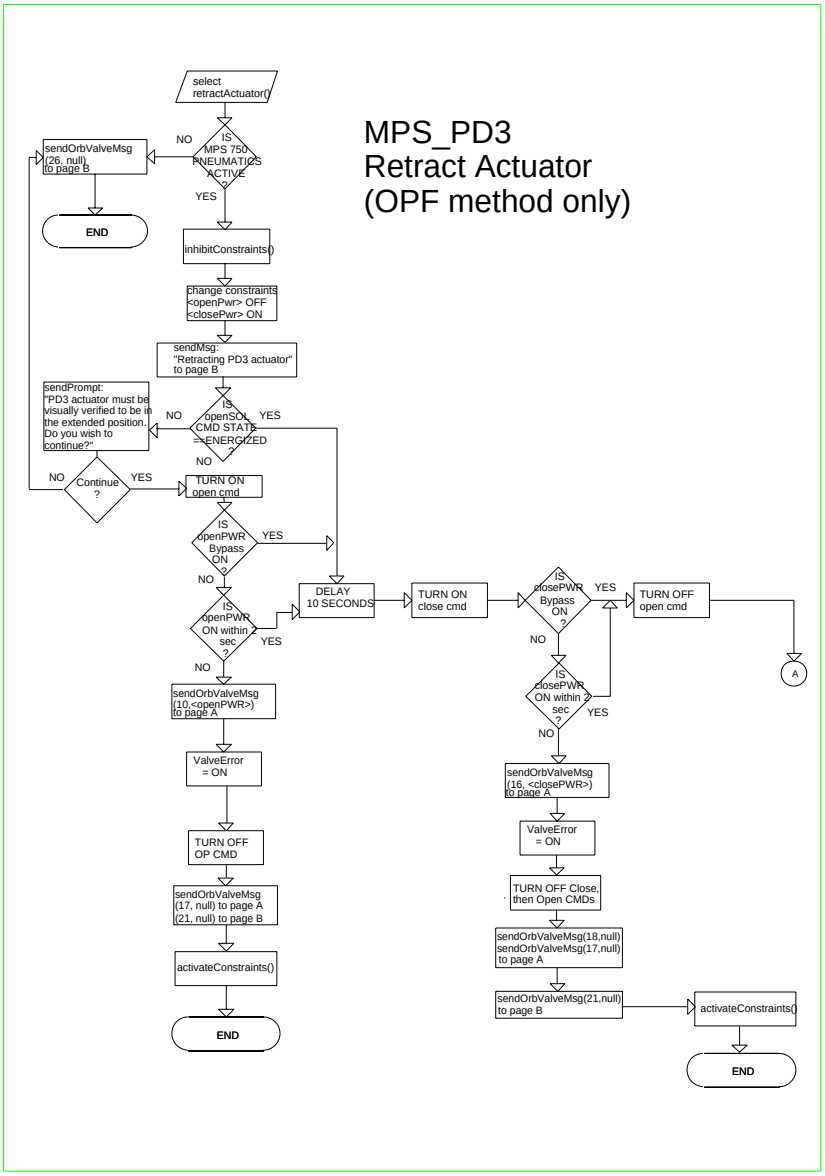


3.1.6.3.4. retractActuator()

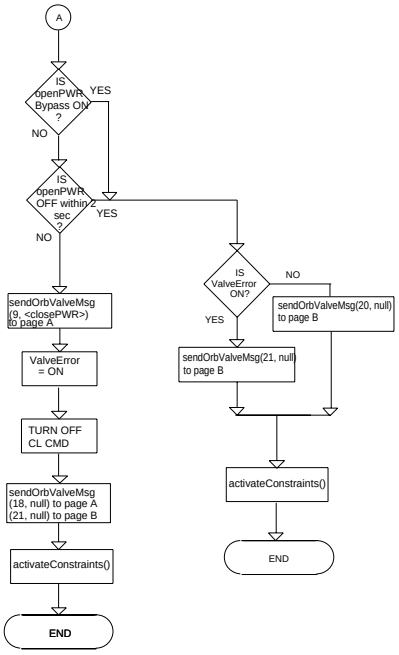
3.1.6.3.4.1. Description

This method performs cycling of the bistable LH2 4" LH2 disconnect valve actuator to the retracted position (in the OPF). The flow chart provides the necessary logic for valve cycles, including checks for snubbing pressure, and proper response of power indications (since the disconnect cannot be cycled in the OPF, position indications do not change). The valve cycles in a fail safe mode and 'wimps-out' (all command removed) at the first indication of hardware failure. ValveError and is initialized to OFF prior to starting valve cycle methods.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

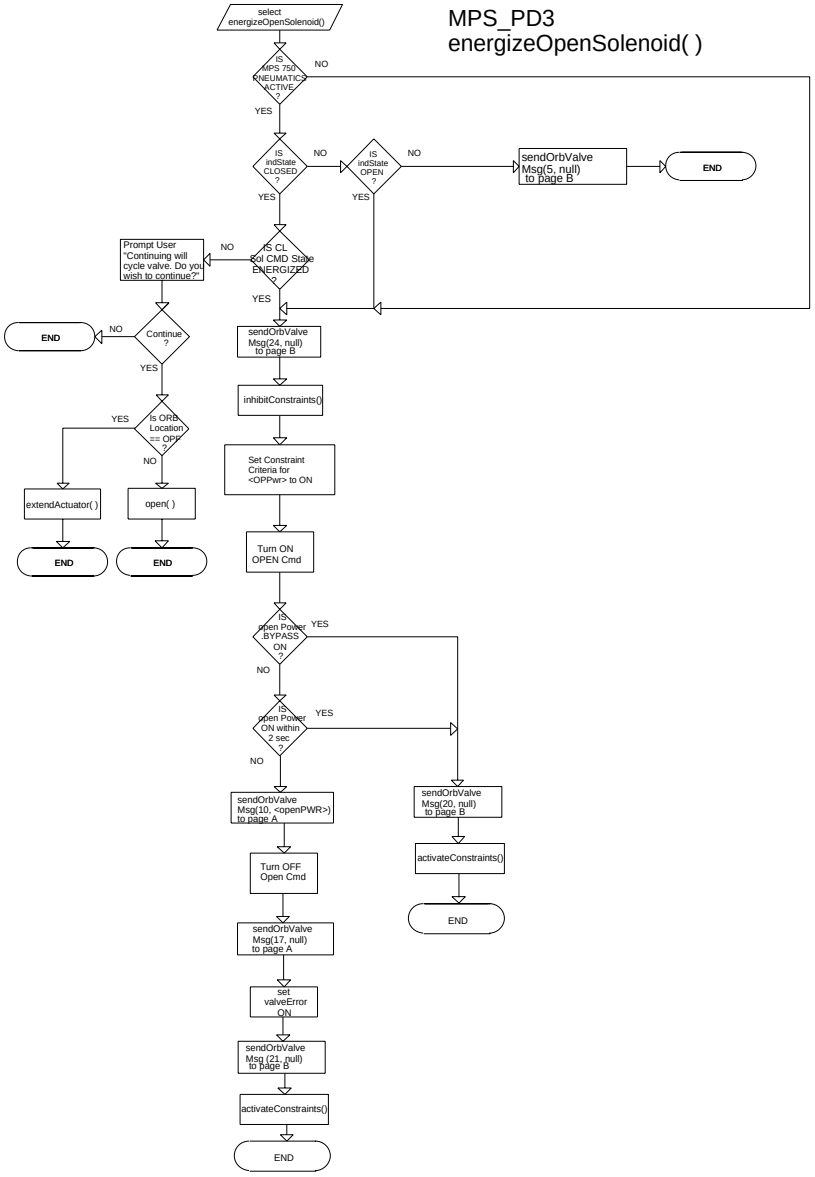


3.1.6.3.5. energizeOpenSolenoid()

3.1.6.3.5.1. Descriptive Text

This method performs the necessary checks to energize the 4" LH2 disconnect valve opening solenoid. The sequence as identified on the following flow chart checks for valve present position, status of MPS 750 pneumatics, and result of energizing the opening solenoid if pneumatics is active. If a valve cycle will occur, the operator is prompted to provide input to continue (Yes/No option), and if a cycle is intended, the close() method is initiated. Constraints are maintained during this operation, with failure messages being generated on valve power indications.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

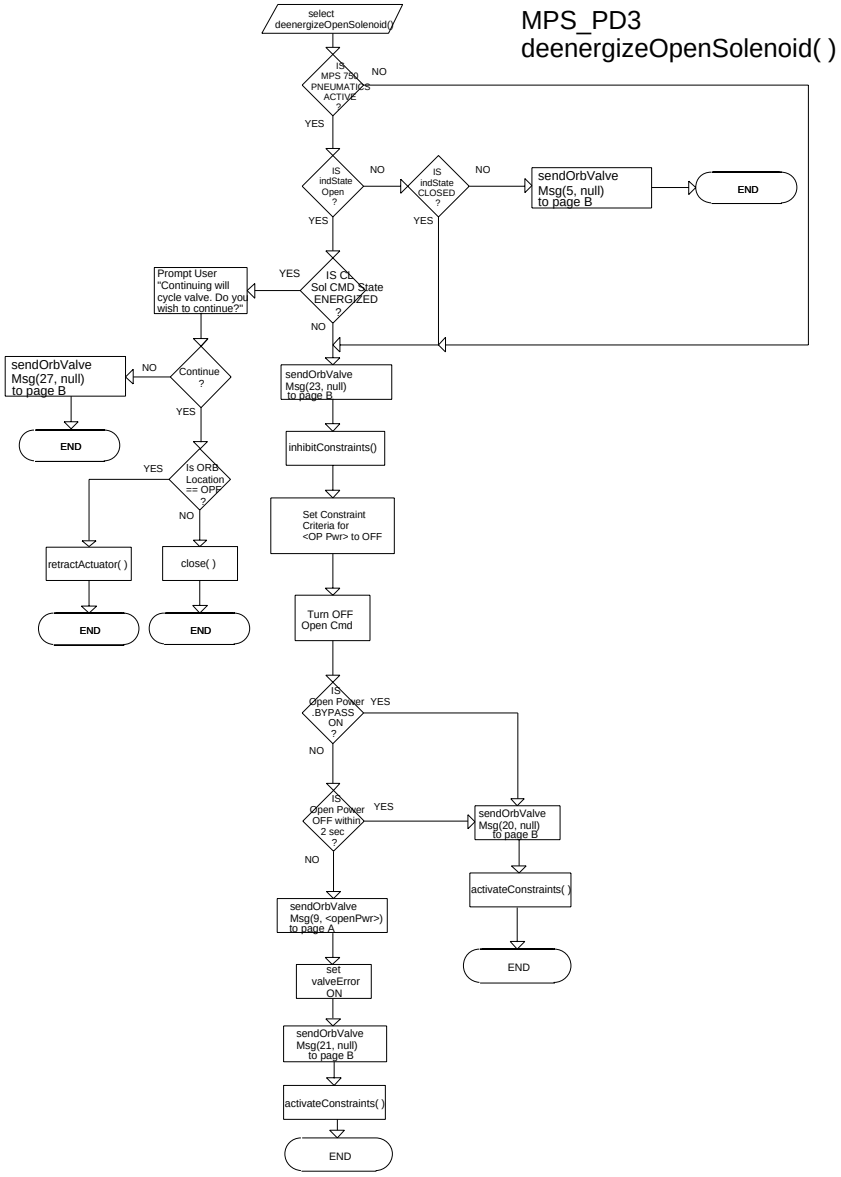


3.1.6.3.6. deenergizeOpenSolenoid()

3.1.6.3.6.1. Descriptive Text

This method performs the necessary checks to deenergize 4" LH2 disconnect valve opening solenoid. The sequence as identified on the following flow chart checks for valve present position, status of MPS 750 pneumatics, and result of deenergizing the opening solenoid if pneumatics is active. If a valve cycle will occur, the operator is prompted to provide input to continue (Yes/No option), and if a cycle is intended, the close() method is initiated. Constraints are maintained during this operation, with failure messages being generated on valve power indications.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

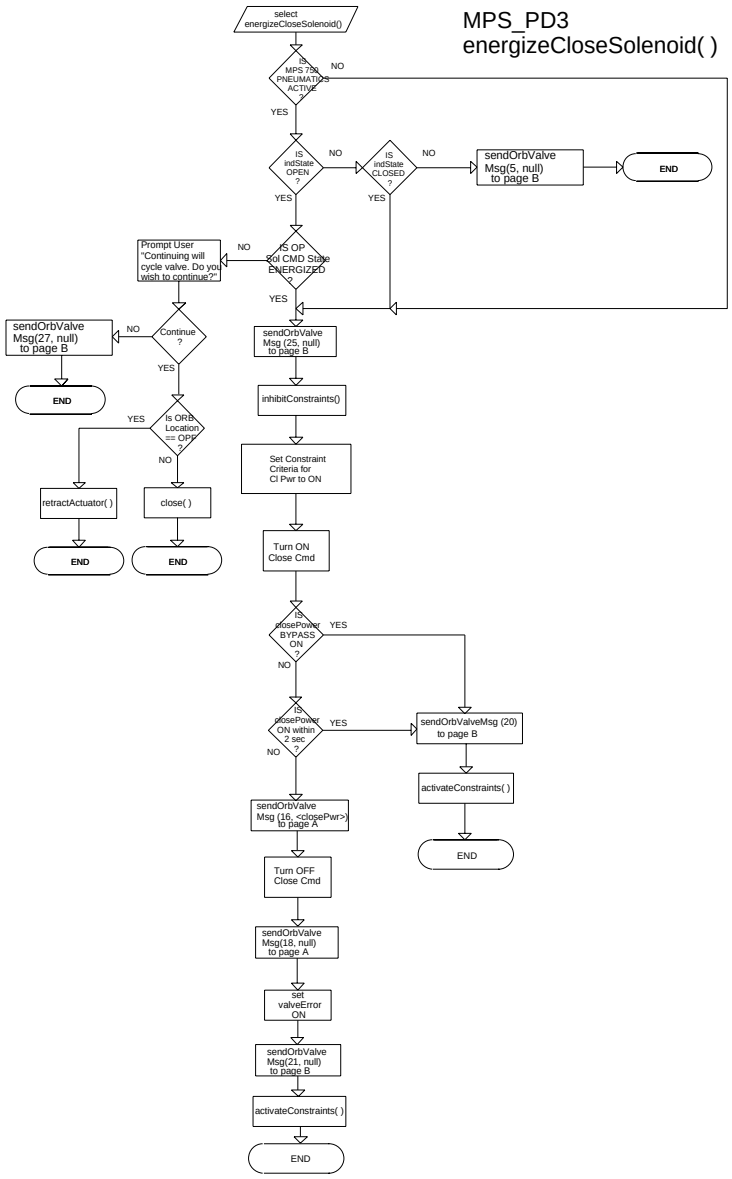


3.1.6.3.7. energizeCloseSolenoid()

3.1.6.3.7.1. Descriptive Text

This method performs the necessary checks to energize the 4" LH2 disconnect valve closing solenoid. The sequence as identified on the following flow chart checks for valve present position, status of MPS 750 pneumatics, and result of energizing the closing solenoid if pneumatics is active. If a valve cycle will occur, the operator is prompted to provide input to continue (Yes/No option), and if a cycle is intended, the close() method is initiated. Constraints are maintained during this operation, with failure messages being generated on valve power indications.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

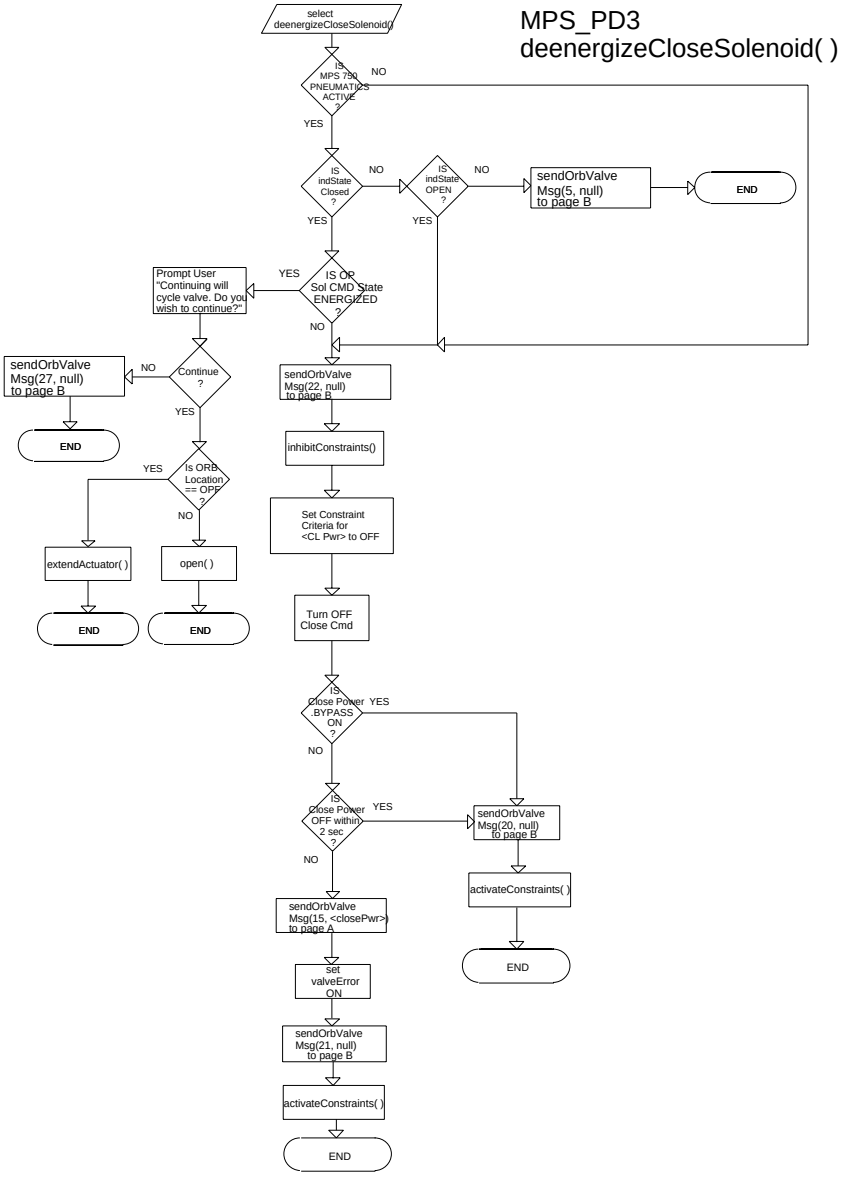


3.1.6.3.8. deenergizeCloseSolenoid()

3.1.6.3.8.1. Descriptive Text

This method performs the necessary checks to deenergize the 4" LH2 disconnect valve closing solenoid. The sequence as identified on the following flow chart checks for valve present position, status of MPS 750 pneumatics, and result of deenergizing the closing solenoid if pneumatics is active. If a valve cycle will occur, the operator is prompted to provide input to continue (Yes/No option), and if a cycle is intended, the open() method is initiated. Constraints are maintained during this operation, with failure messages being generated on valve power indications.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



3.1.6.3.9. determineIndState()

3.1.6.3.9.1. Descriptive Text

This method will determine the valve's position based on its position indicators and associated indicator bypasses. The return value will be an enumerated type with possible values of OPEN, CLOSED, TRANSITION, and ERROR.

3.1.6.3.9.2. Effectivity

3.1.6.3.9.3. Sequence

Case:

<openPosInd> == OFF
<openPosInd>Byp == OFF

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
<closePosInd> == ON
<closePosInd>Byp == OFF
                    indState = CLOSED;

<openPosInd> == OFF
<openPosInd>Byp == OFF
<closePosInd>Byp == ON
                    indState = CLOSED;

<openPosInd>Byp == ON
<closePosInd> == ON
<closePosInd>Byp == OFF
                    indState = CLOSED;

<openPosInd> == ON
<openPosInd>Byp == OFF
<closePosInd> == OFF
<closePosInd>Byp == OFF
                    indState = OPEN;

<openPosInd> == ON
<openPosInd>Byp == OFF
<closePosInd>Byp == ON
                    indState = OPEN;

<openPosInd>Byp == ON
<closePosInd> == OFF
<closePosInd>Byp == OFF
                    indState = OPEN;



<openPosInd> == OFF
<openPosInd>Byp == OFF
<closePosInd> == OFF
<closePosInd>Byp == OFF
                    indState = TRANSITION;


CASE ELSE
                     indState = ERROR;
```

End Case:

3.1.6.3.10. determineOpenSolCmdState()

3.1.6.3.10.1. Descriptive Text

This operation is used to determine the open solenoid's commanded state based on the open power indicator and its associated bypass. The return value will be an enumerated type with possible values of ENERGIZED, DEENERGIZED and ERROR.

3.1.6.3.10.2. Effectivity

Operates on change data.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.6.3.10.3. Sequence

Case:

<openPwr> == OFF
<openPwr>Byp == OFF
openSolCmdState = DEENERGIZED;

<openPwr> == ON
<openPwr>Byp == OFF
openSolCmdState = ENERGIZED;

<openPwr>Byp==On
openSolCmdState = ERROR;

End Case:

3.1.6.3.11. determineCloseSolCmdState()

3.1.6.3.11.1. Descriptive Text

This method will determine the close solenoid's commanded state based on the two close RPCs, the close power, and the associated bypasses. The return value will be an enumerated
type with possible values of ENERGIZED, DEENERGIZED and ERROR.

3.1.6.3.11.2. Effectivity

Operates on change data.

3.1.6.3.11.3. Sequence

Case:

<closePwr> == OFF
<closePwr>Byp == OFF
closeSolCmdState = DEENERGIZED;

<closePwr> == ON
<closePwr>Byp == OFF
closeSolCmdState = ENERGIZED;

<closePwr>Byp == ON
closeSolCmdState = ERROR;

End Case:

3.1.6.3.12. determineCmdState()//Phase 3 method

3.1.6.3.13. determineState()//Phase 3 method

3.1.6.3.14. getIndState()

3.1.6.3.14.1. Descriptive Text

This method will return the Enumerated Indicator State of the valve

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.6.3.14.2. Sequence

```
getIndState()
{
        return IndState;
}
```

3.1.6.3.15. getOpenSolCmdState()

3.1.6.3.15.1. Descriptive Text

This method will return the Enumerated Command State of the valve Opening Solenoid.

3.1.6.3.15.2. Sequence

```
getOpenSolCmdState()
{
        return OpenSolCmdState;
}
```

3.1.6.3.16. getCloseSolCmdState()

3.1.6.3.16.1. Descriptive Text

This method will return the Enumerated Command State of the valve Closing Solenoid.

3.1.6.3.16.2. Sequence

```
getCloseSolCmdState()
{
        return CloseSolCmdState;
}
```

3.1.6.3.17. verifyCommandsOff()

3.1.6.3.17.1. Descriptive Text

This function will verify that both the open and close commands are OFF.

3.1.6.3.17.2. Sequence

```
verifyCommandsOff()
        {
        open command status = byteTest4 <Open Command>
        verify  open command status == OFF
        ELSE send message to MPS Message display: "Open command" <OP CMD>;"is ON, should be OFF."
        }
        {
        close command status = byteTest4 <Close Command>
        verify  close command status == OFF
        ELSE send message to MPS Message display: "Close command" <CL CMD>;"is ON, should be OFF."
        }
        {
        verify  open command status && close command status == OFF
        return TRUE
        ELSE return FALSE
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.1.6.3.18. setCloseConstraints()

3.1.6.3.18.1. Descriptive Text

This method changes the constraint criteria of the measurements associated with this valve to those expected with the valve in the closed position.

3.1.6.3.18.2. Sequence

setCloseConstraints()

{

CHANGE <openPwr> CONSTRAINT CRITERIA TO OFF;

CHANGE <openPosInd> CONSTRAINT CRITERIA TO OFF;

CHANGE <closePwr> CONSTRAINT CRITERIA TO ON;

CHANGE <closePosInd> CONSTRAINT CRITERIA TO ON;

}

3.1.6.3.19. setCloseIndConstraints()

3.1.6.3.19.1. Descriptive Text

3.1.6.3.19.2. Sequence

setCloseIndConstraints()

{

CHANGE <openPwr> CONSTRAINT CRITERIA TO OFF;

CHANGE <openPosInd> CONSTRAINT CRITERIA TO OFF;

CHANGE <closePwr> CONSTRAINT CRITERIA TO OFF;

CHANGE <closePosInd> CONSTRAINT CRITERIA TO ON;

}

3.1.6.3.20. setOpenConstraints()

3.1.6.3.20.1. Descriptive Text

This method changes the constraint criteria of the measurements associated with this valve to those expected with the valve in the open position.

3.1.6.3.20.2. Sequence

setOpenConstraints()

{

CHANGE <openPwr> CONSTRAINT CRITERIA TO ON;

CHANGE <openPosInd> CONSTRAINT CRITERIA TO ON;

CHANGE <closePwr> CONSTRAINT CRITERIA TO OFF;

CHANGE <closePosInd> CONSTRAINT CRITERIA TO OFF;

}

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.6.3.21. activateConstraints()

3.1.6.3.21.1. Descriptive Text

This operation activates constraint monitoring on valve position and power indicators.

3.1.6.3.21.2. Sequence

```
activateConstraints()
{
        If (<openPwr>.Byp == OFF)
        Activate Constraint Monitoring on <openPwr>
        }
        {
        If (<openPosInd>.Byp == OFF)
        Activate Constraint Monitoring on <openPosInd>
        }
        {
        If (<closePosInd>.Byp == OFF)
        Activate Constraint Monitoring on <closePosInd>
        {
        If (<closePwr>.Byp == OFF)
        Activate Constraint Monitoring on <closePwr>

}
```

3.1.6.3.22. inhibitConstraints()

3.1.6.3.22.1. Descriptive Text

This operation de-activates constraint monitoring on valve position and power indicators.

3.1.6.3.22.2. Sequence

```
inhibitConstraints()
{
        Inhibit Constraint Monitoring on
                <openPwr>
                <closePwr>
                <openPosInd>
                <closePosInd>
}
```

3.1.6.3.23. verifyTminus3BT4()//

3.1.6.3.23.1. Description

This method performs a byte test 4 of fill and drain valve commands to verify they are in a proper T-3 hour configuration. For both instances of this class, open commands should be ON, close commands should be OFF.

3.1.6.3.23.2. Sequence

```
begin sequence;

set commanderrorcheck OFF
{
perform ByteTest4 of <openCMD>
        verify ByteTest4 = ON
        else send message to page A
                valveID <openCMD> "is not in T-3 hour configuration."
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
                and set commanderrorcheck ON
}
{
perform ByteTest4 of <closeCMD>
        verify ByteTest4 = OFF
        else send message to page A
                valveID <closeCMD> "is not in T-3 hour configuration."
                and set commanderrorcheck ON
}
return commanderrorcheck

end sequence;
```

3.1.6.3.24. verifyTminus6BT4()//

3.1.6.3.24.1. Description

This method performs a byte test 4 of fill and drain valve commands to verify they are in a proper T-6 hour configuration. For both instances of this class, open commands should be OFF, close commands should be ON.

3.1.6.3.24.2. Sequence

```
begin sequence;

set commanderrorcheck OFF
{
perform ByteTest4 of <openCMD>
        verify ByteTest4 = ON
        else send message to page A
                "Valve" <openCMD> "is not in T-6 hour configuration."
                and set commanderrorcheck ON
}
{
perform ByteTest4 of <closeCMD>
        verify ByteTest4 = OFF
        else send message to page A
                "Valve" <closeCMD> "is not in T-6 hour configuration."
                and set commanderrorcheck ON
}
return commanderrorcheck

end sequence;
```

3.1.6.4. Safety Requirements

3.1.7. MPS_Valve_ORB_LO2_Prevalve

3.1.7.1. Description

MPS_Valve_ORB_BISTABLE_LO2_Prevalve is a bi-stable valve with 4 open and 4 close commands, 1 open and 1 close position indications, 2 open and 2 close power indications, 4 open and 4 close RPCs. The valve actuator solenoids (2 open in series and 2 close in parallel) are each redundantly commanded/powered. There are 3 instances of this valve, one for each SSME feedline (MPS_PV1, PV2, PV3). Functionally, the valves are used to isolate the SSME from the LO2 feed system during leak isolation and SSME shutdown (inclding abort) operations.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

The valve class inherits from the MPS_Valve_ORB Class. The valve has an association with the MPS_Helium_SubSystem Class for status of the MPS 750 pneumatic system. For integrated operations, this valve class has an association with the MPS_SubSystem Class for determination of the cryo load stautus via a Cryo Flag pseudo. During test and check-out mode, valve cycles will break-out and remove all commands/power when a failure is detected. When the Cyro Flag is ON, the valve cycle sequence will continue to completion regardless of failures.

3.1.7.1.1. Instance Index

Instance	Other Users
MPS_PV1	
MPS_PV2	
MPS_PV3	

3.1.7.1.2. Class Diagram

<insert class diagram here>

3.1.7.1.3. Attributes and Aggregations

Attribute	Data Type	Initial Value	Publish / Persistence	Client Access	Description
openPosInd	DM_ON_OFF				
openCmdA	DS_ON_OFF				
openCmdB	DS_ON_OFF				
openCmdC	DS_ON_OFF				
openCmdD	DS_ON_OFF				
openRpc1	DM_ON_OFF				
openRpc2	DM_ON_OFF				
openPwr1	DM_ON_OFF				
openRpc3	DM_ON_OFF				
openRpc4	DM_ON_OFF				
openPwr2	DM_ON_OFF				
closePosInd	DM_ON_OFF				
closeCmdA	DS_ON_OFF				
closeCmdB	DS_ON_OFF				

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

closeCmdC	DS_ON_OFF				
closeCmdD	DS_ON_OFF				
closeRpc1	DM_ON_OFF				
closeRpc2	DM_ON_OFF				
closeRpc3	DM_ON_OFF				
closeRpc4	DM_ON_OFF				
closePwr1	DM_ON_OFF				
closePwr2	DM_ON_OFF				
mainstageCmdA	DS_ON_OFF				
mainstageCmdB	DS_ON_OFF				
mainstageCmdC	DS_ON_OFF				
IndState	MPS_VALVE_IND_STATE ENUM		PUB	R,W	
overallState	MPS_ORB_VALVE_STATE ENUM		PUB	R,W	
openSol1CmdState	MPS_ORB_SOL_STATE ENUM		PUB	R,W	
openSol2CmdState	MPS_ORB_SOL_STATE ENUM		PUB	R,W	
closeSol1CmdState	MPS_ORB_SOL_STATE ENUM		PUB	R,W	
closeSol2CmdState	MPS_ORB_SOL_STATE ENUM		PUB	R,W	

3.1.7.1.4. Methods Index

Operation	**Description**
open()	This method performs necessary functions to cycle valve to the open position.
close()	This method performs necessary functions to cycle valve to the closed position.
energizeOpenSolenoid1()	This method energizes open solenoid 1. Method includes a check of

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

	MPS pneumatics status, and cautions operator of snubbing requirements if MPS pneumatics is active.
energizeOpenSolenoid2()	This method energizes open solenoid 2. Method includes a check of MPS pneumatics status, and cautions operator of snubbing requirements if MPS pneumatics is active.
energizeCloseSolenoid1()	This method energizes closing solenoid 1. Method includes a check of MPS pneumatics status, and cautions operator of snubbing requirements if MPS pneumatics is active.
energizeCloseSolenoid2()	This method energizes closing solenoid 2. Method includes a check of MPS pneumatics status, and cautions operator of snubbing requirements if MPS pneumatics is active.
deenergizeOpenSolenoid1()	This method de-energizes opening solenoid 1.
deenergizeOpenSolenoid2()	This method de-energizes opening solenoid 2.
deenergizeCloseSolenoid1()	This method de-energizes closing solenoid 1.
deenergizeCloseSolenoid2()	This method de-energizes closing solenoid 2.
openON_check()	This is a private method used by the other LO2 prevalve valve cycle methods. Its purpose is to verify that the appropriate open powers and RPCs energize within 2 seconds of command.
openOff_check()	This is a private method used by the other LO2 prevalve valve cycle methods. Its purpose is to verify that the appropriate open powers and RPCs deenergize within 2 seconds of command.
closeON_check()	This is a private method used by the other LO2 prevalve valve cycle methods. Its purpose is to verify that the appropriate close powers and RPCs energize within 2 seconds of command.
closeOff_check()	This is a private method used by the other LO2 prevalve valve cycle methods. Its purpose is to verify that the appropriate close powers and RPCs deenergize within 2 seconds of command.
determineIndState()	This method will determine the position state of the valve based on open/close indicators and will return either: OPEN CLOSED TRANSITION ERROR
determineOpSol1CmdState()	This method will determine the command state of the opening solenoid valve based on valve RPC and power indicators and will return either: ENERGIZED DEENERGIZED ERROR
determineOpSol2CmdState()	This method will determine the command state of the opening solenoid

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

	valve based on valve RPC and power indicators and will return either: ENERGIZED DEENERGIZED ERROR
determineClSol1CmdState()	This method will determine the command state of the opening solenoid valve based on valve RPC and power indicators and will return either: ENERGIZED DEENERGIZED ERROR
determineClSol2CmdState()	This method will determine the command state of the opening solenoid valve based on valve RPC and power indicators and will return either: ENERGIZED DEENERGIZED ERROR
determineCombinedCmdState()	This method will determine the over-all command state for the prevalve and return either: CMD_CLOSED CMD_OPEN CMD_SNUBBED UNCOMMANDED ERROR
getCombinedCmdState()	This method will return valve Enumerated combined command state.
determineOverallState()	This method will determine the over-all combined state for the prevalve and return either: CLOSED OPEN TRANSITION CLOSED_ERROR OPEN_ERROR SNUB_CLOSED SNUB_OPEN ERROR
getOverallState()	This method will return valve Enumerated combined overall state.
getIndState()	This method will return valve indicator Enumerated state.
getOPSol1CmdState()	This method will return valve opening solenoid 1 Enumerated command state.
getOPSol2CmdState()	This method will return valve opening solenoid 2 Enumerated command state.
getClSol1CmdState()	This method will return valve closing solenoid 1 Enumerated command state.
getClSol2CmdState()	This method will return valve closing solenoid 2 Enumerated command state.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

verifyCmdsOff()	This method will verify no commands issued.
setOpenConstraints()	This method establishes Constraint configuration for an open prevalve.
setOpenPosConstraints()	This method establishes Constraint configuration for an uncommanded bistable open prevalve.
setCloseConstraints()	This method establishes Constraint configuration for a closed prevalve.
setClosePosConstraints()	This method establishes Constraint configuration for an uncommanded bistable closed prevalve.
activateConstraints()	This method will activate Constraints for either the close or open positions when in standby mode (uncommanded present state).
verifyTminus3BT4()//	This method verifies valve commands are in a T-3 HR configuration. This is a VAB/PAD method.
verifyTminus6BT4()//	This method verifies valve commands are in a T-6 HR configuration. This is a VAB/PAD method.
inhibitConstraints()	This method will inhibit Constraint notification during valve cycle operations.
autoCycle()	This method will perform valve open/close autocycle operations.

3.1.7.1.5. Associations

Association	Type	Class	Description
MPS_SubSystem			CryoFlag status.
MPS_Helium_SubSystem			MPS 750 pneumatic status.

3.1.7.2. State Description

3.1.7.2.1. State Diagram

<insert state diagram here>

3.1.7.2.2. State Table

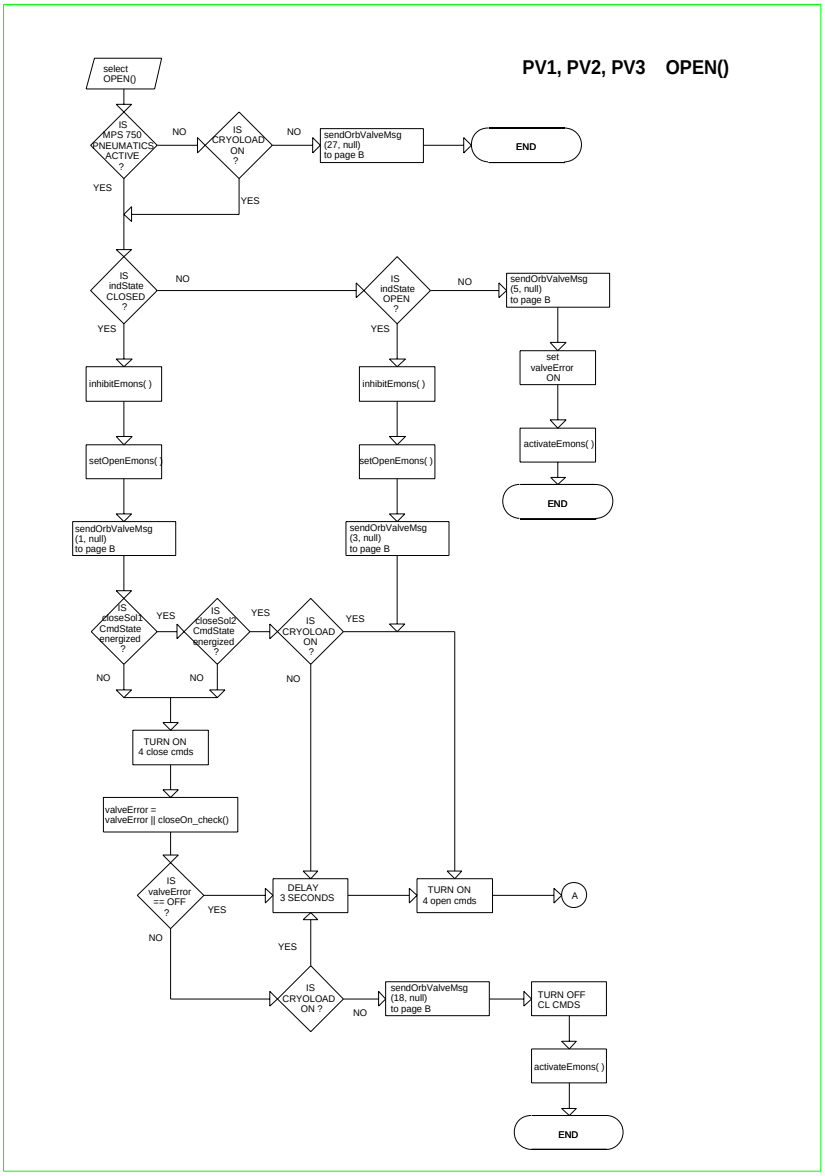
3.1.7.3. Operations

3.1.7.3.1. open()

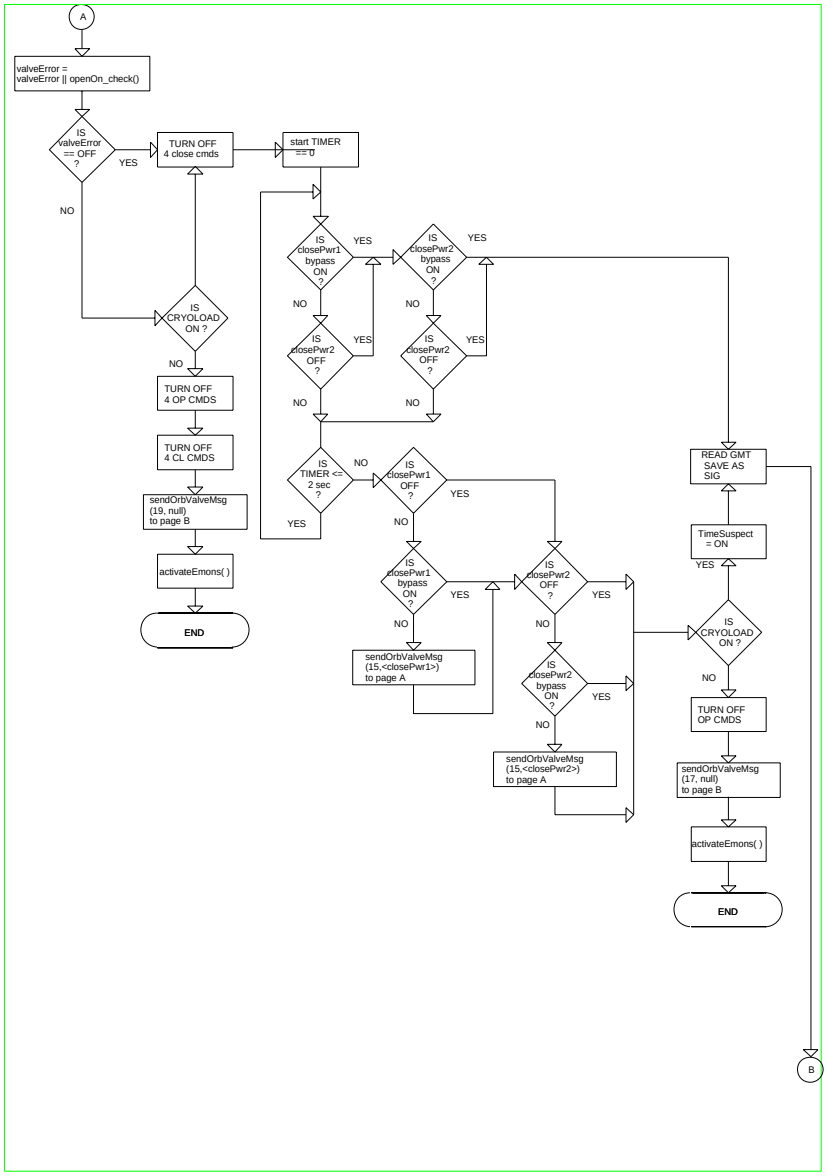
3.1.7.3.1.1. Descriptive Text

The flow charts contained below describe the logic to be used to code the method. The implementation of the details is left up to the coders and is acceptable as long as the overall logic is maintained. It is expected that cycle timing will be accomplished using the most accurate source. All messages will include a time stamp, and the appropriate FD description.

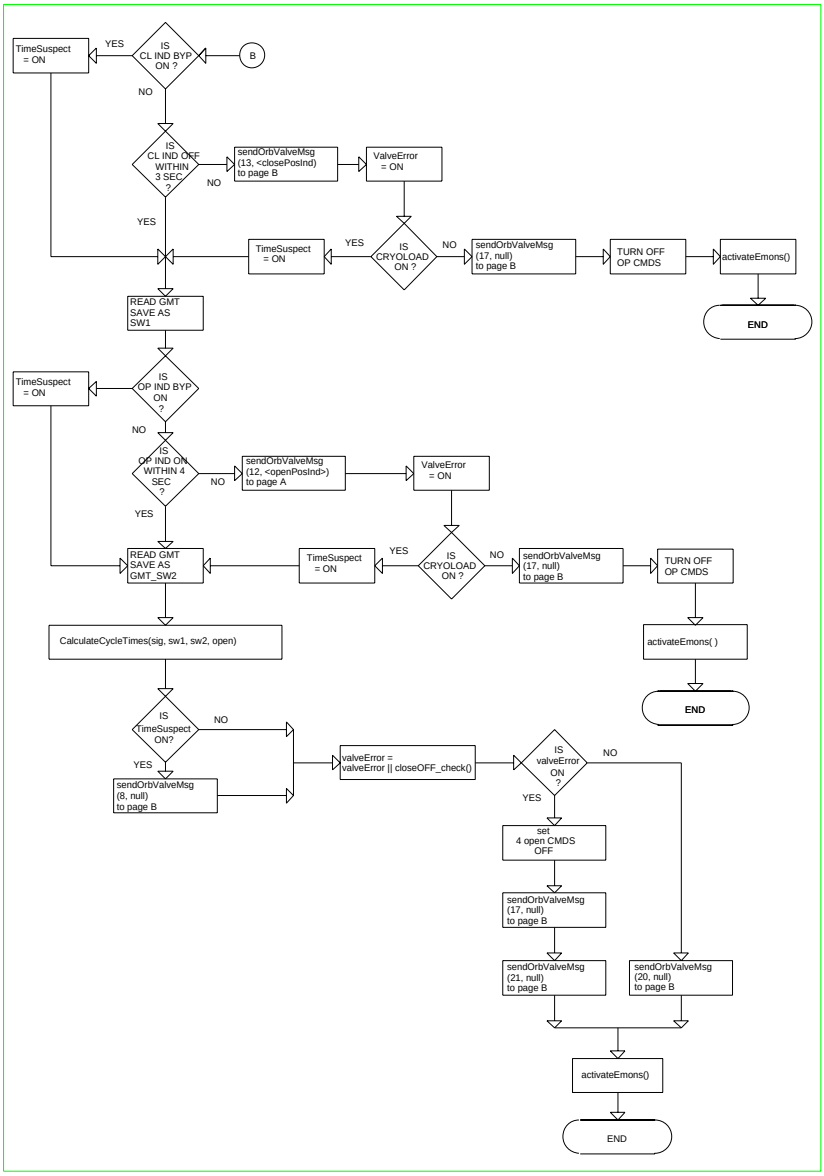
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

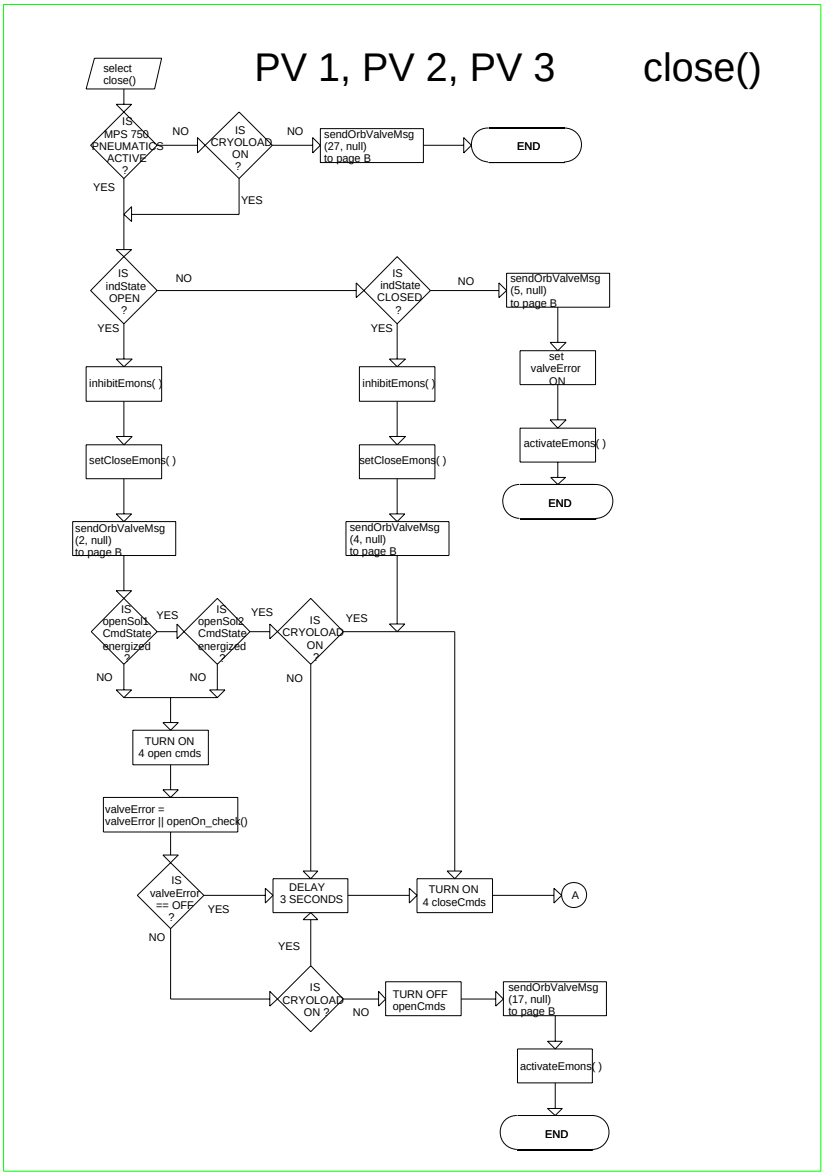


3.1.7.3.2. close()

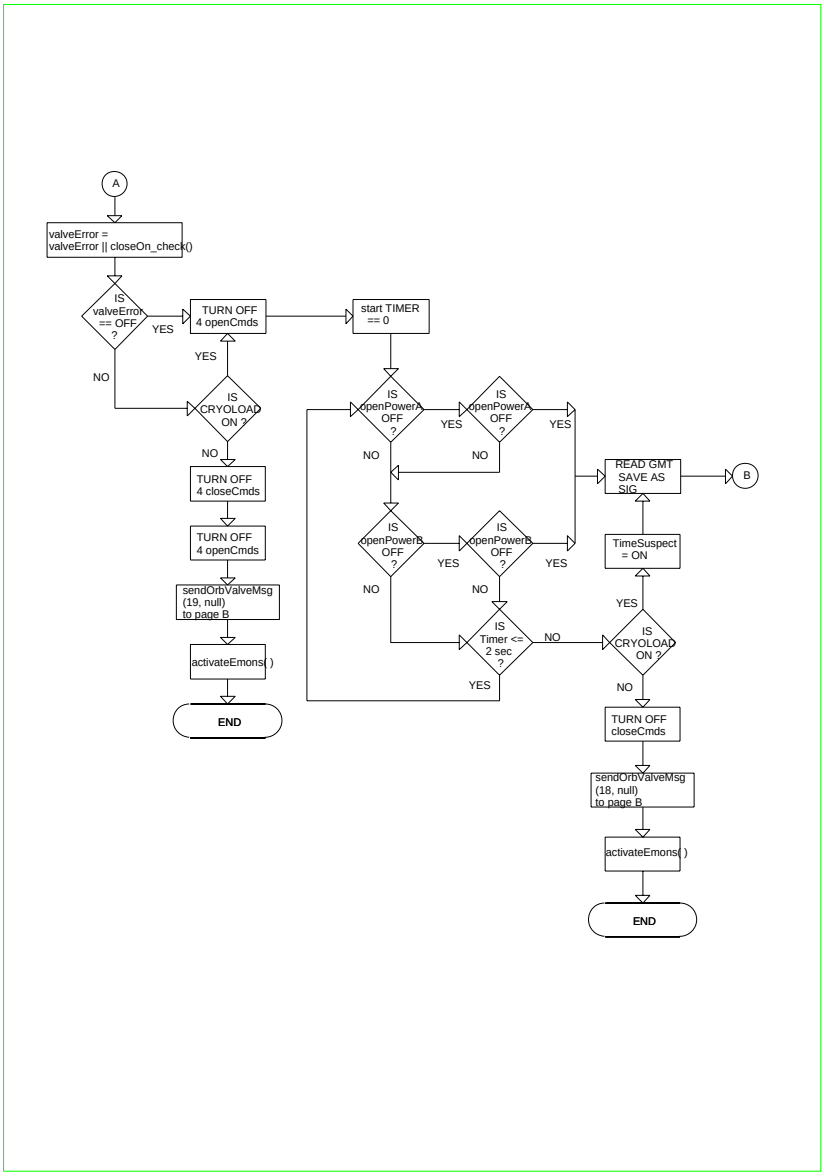
3.1.7.3.2.1. Descriptive Text

The flow charts contained below describe the logic to be used to code the method. The implementation of the details is left up to the coders and is acceptable as long as the overall logic is maintained. It is expected that cycle timing will be accomplished using the most accurate source. All messages will include a time stamp, and the appropriate FD description.

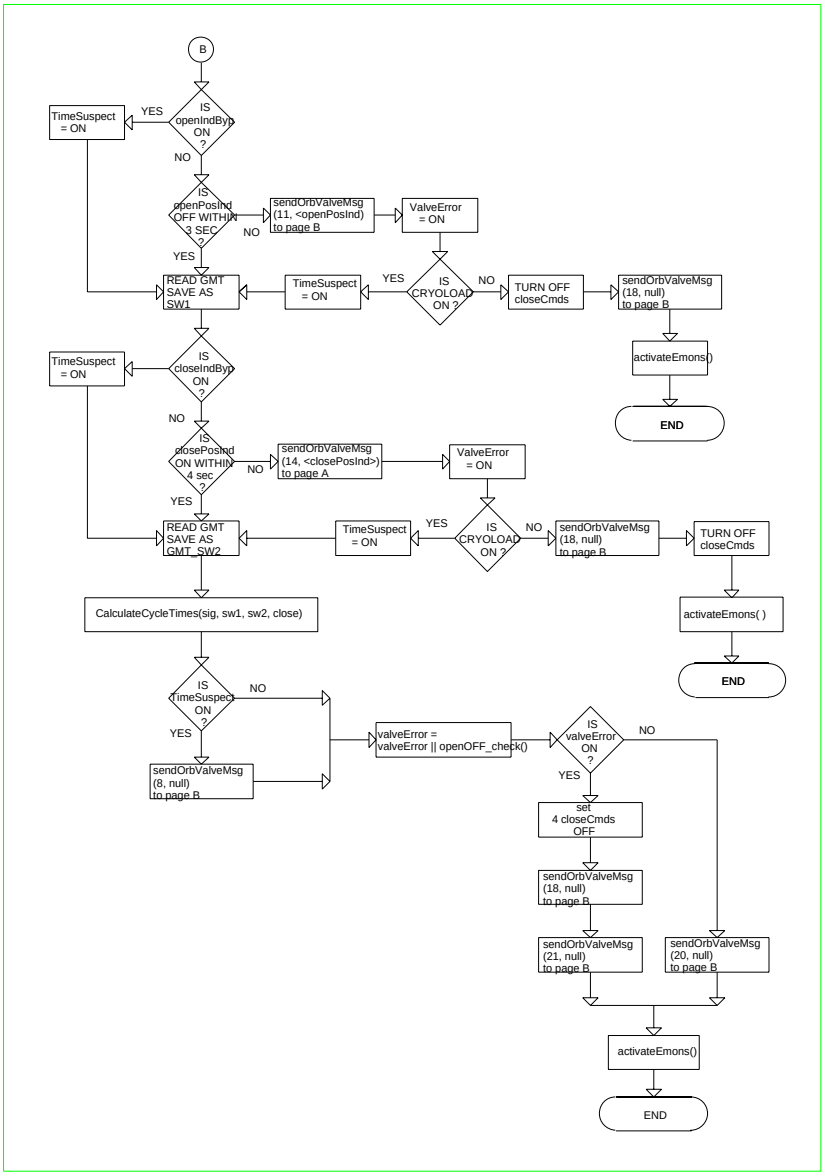
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



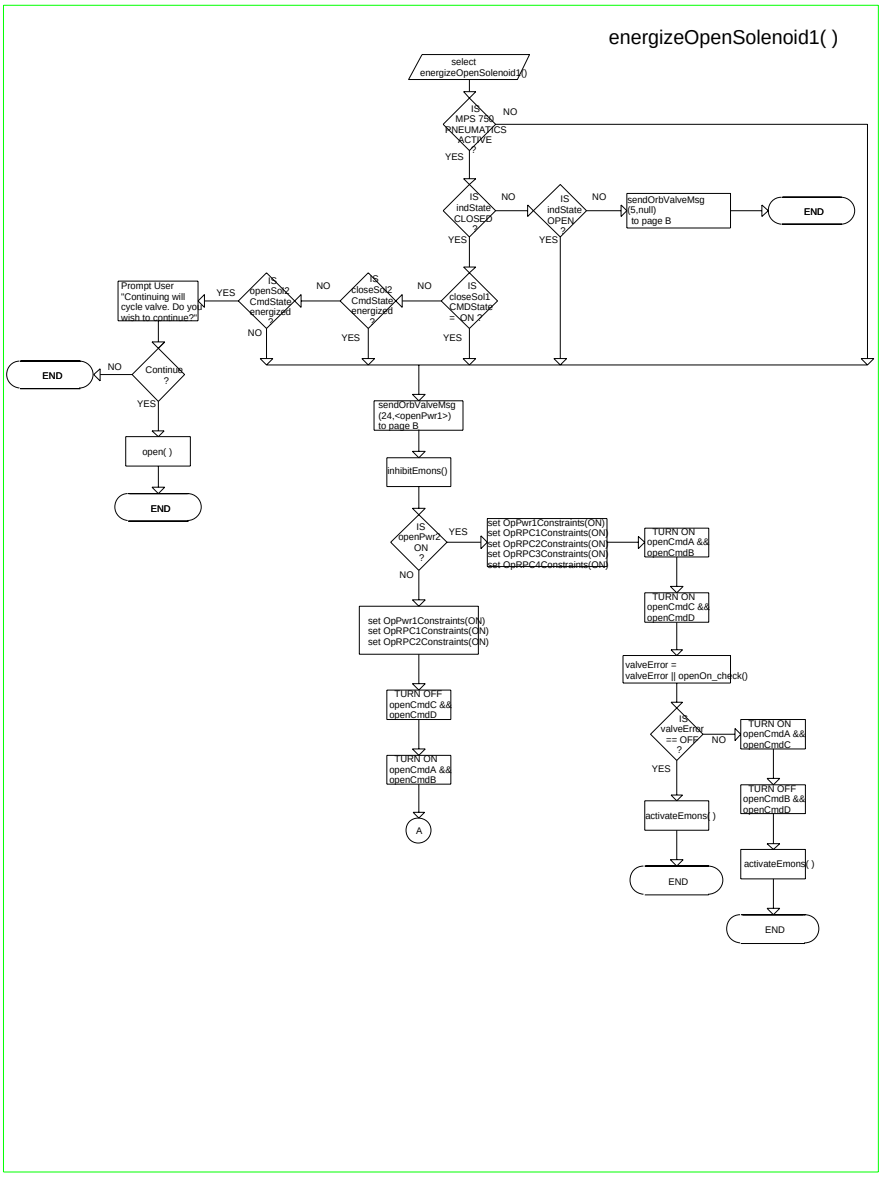
3.1.7.3.3. energizeOpenSolenoid1()

3.1.7.3.3.1. Descriptive Text

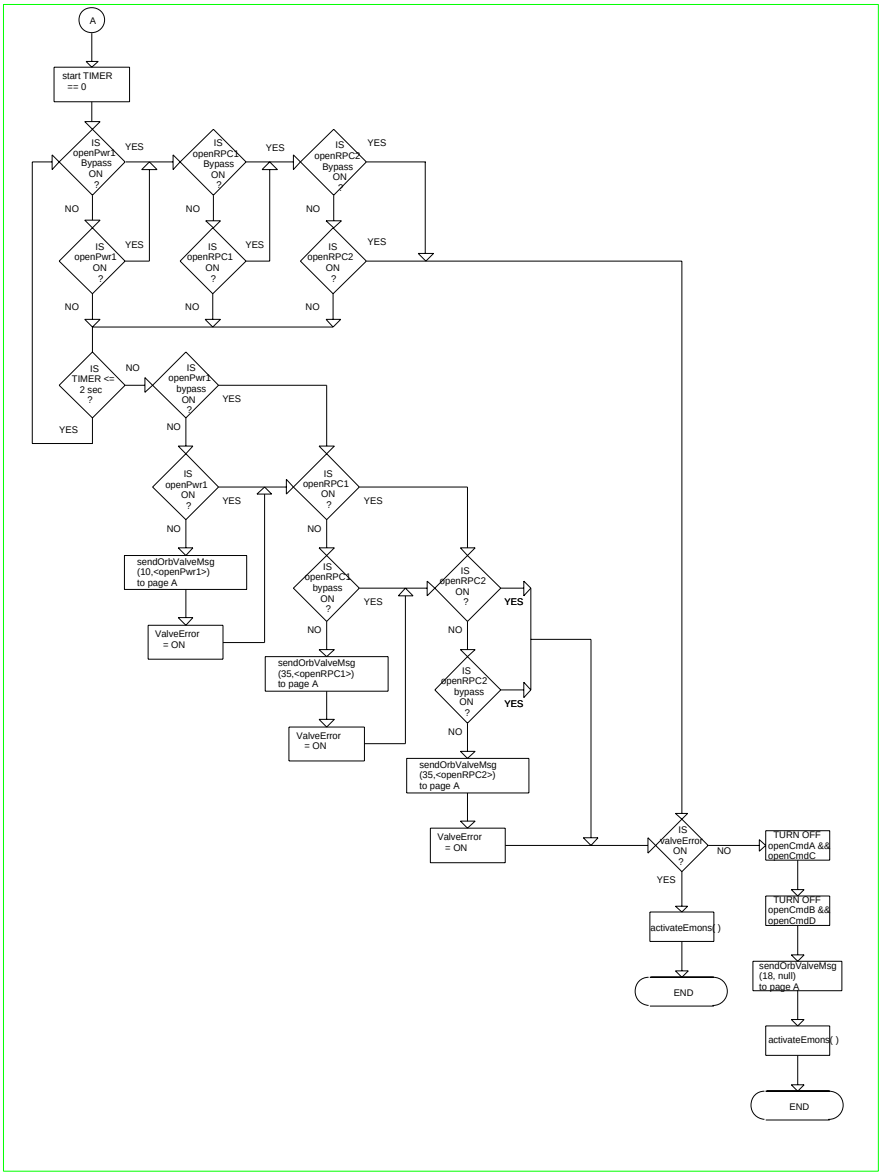
The flow charts contained below describe the logic to be used to code the method. The implementation of the details is left up to the coders and is acceptable as long as the overall logic is maintained. It is expected that cycle timing will be accomplished using the most accurate source. All messages will include a time stamp, and the appropriate FD description.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.7.3.3.2. Sequence



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



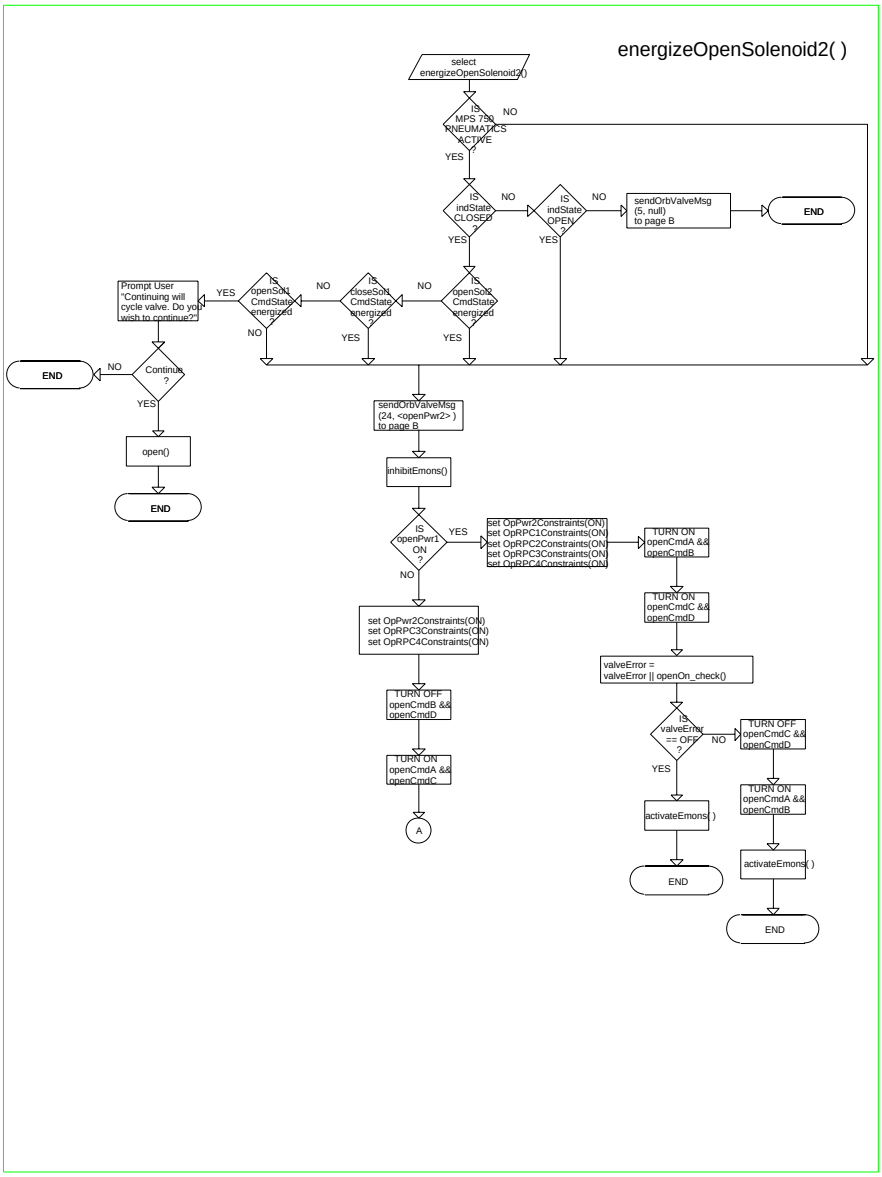
3.1.7.3.4. energizeOpenSolenoid2()

3.1.7.3.4.1. Descriptive Text

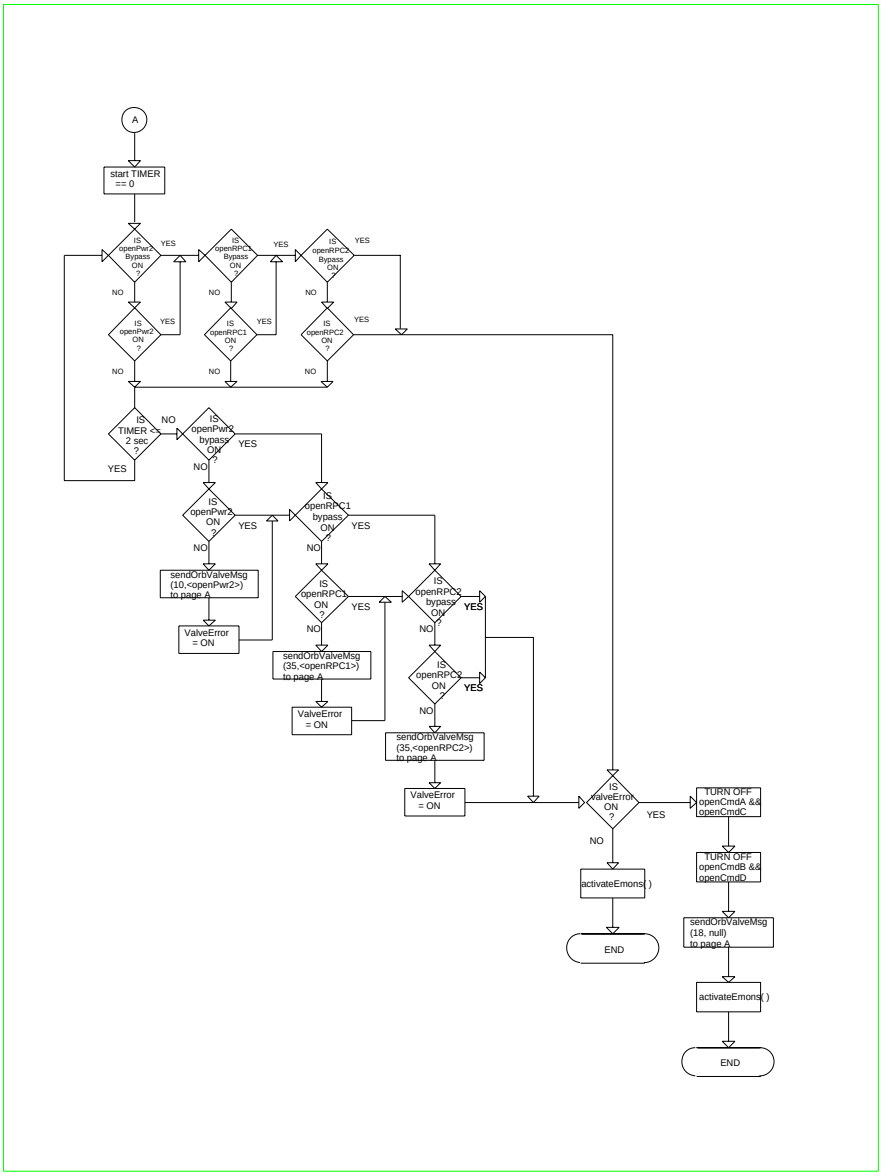
The flow charts contained below describe the logic to be used to code the method. The implementation of the details is left up to the coders and is acceptable as long as the overall logic is maintained. It is expected that cycle timing will be accomplished using the most accurate source. All messages will include a time stamp, and the appropriate FD description.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.7.3.4.2. Sequence



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



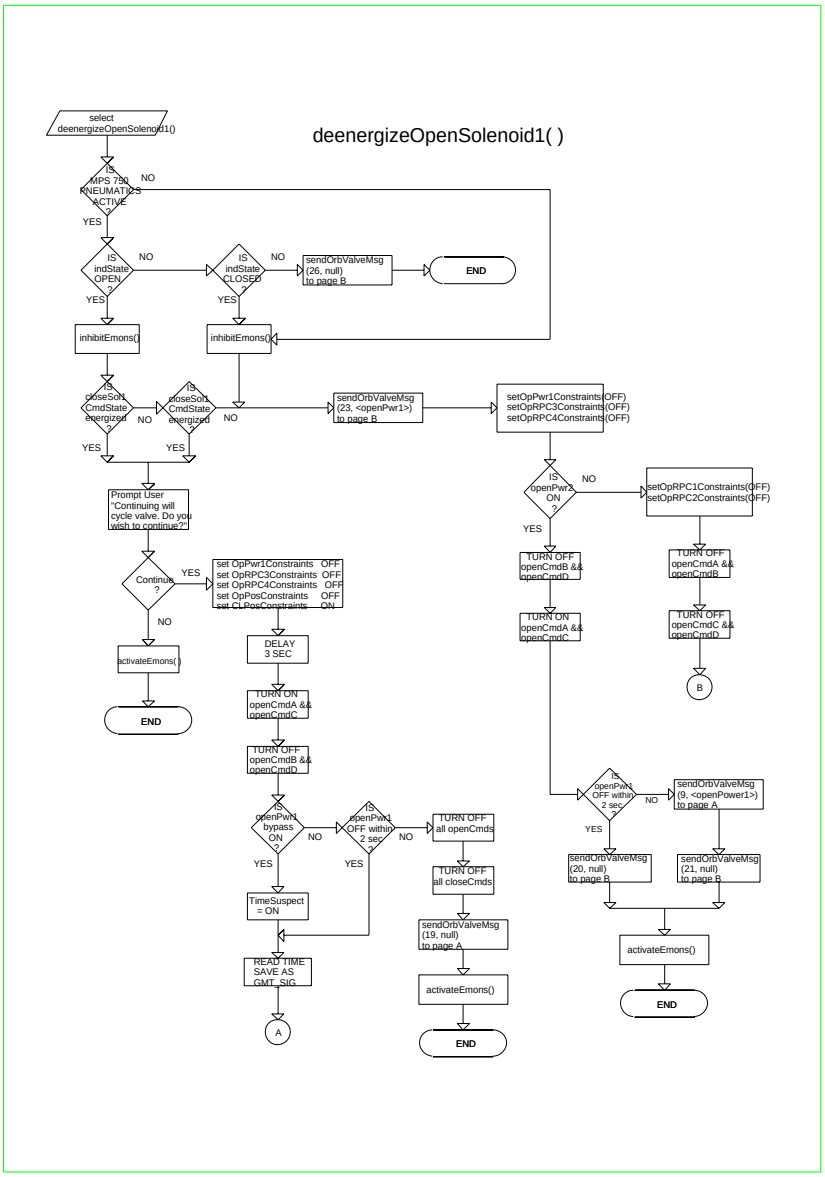
3.1.7.3.5. deenergizeOpenSolenoid1()

3.1.7.3.5.1. Descriptive Text

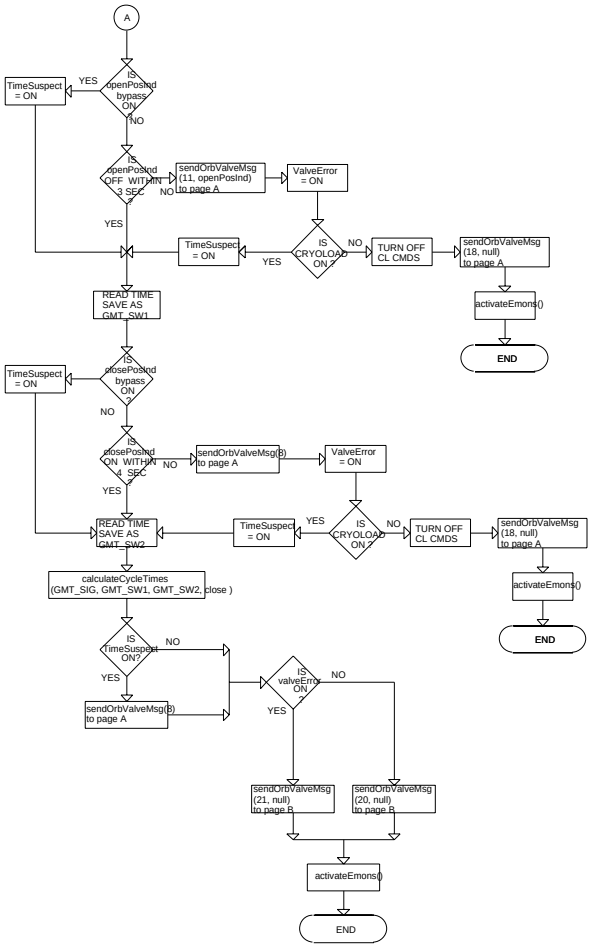
The flow charts contained below describe the logic to be used to code the method. The implementation of the details is left up to the coders and is acceptable as long as the overall logic is maintained. It is expected that cycle timing will be accomplished using the most accurate source. All messages will include a time stamp, and the appropriate FD description.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

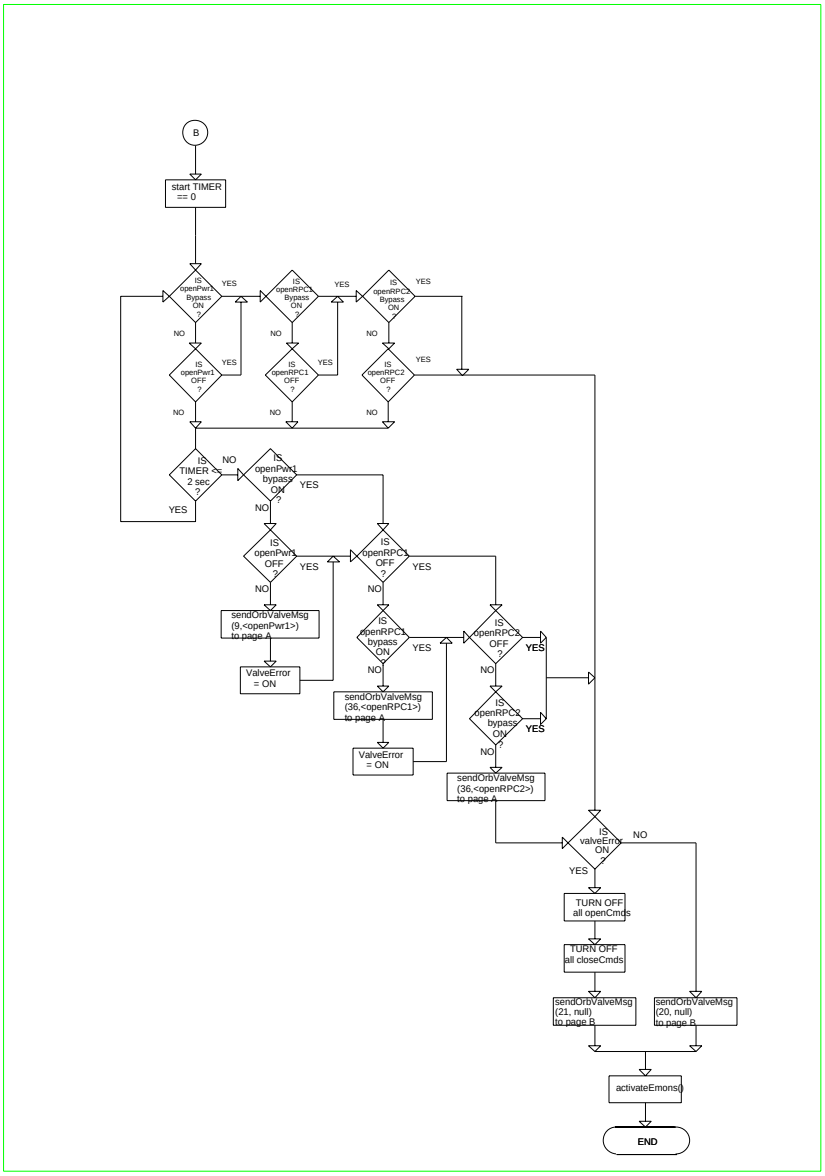
3.1.7.3.5.2. Sequence



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

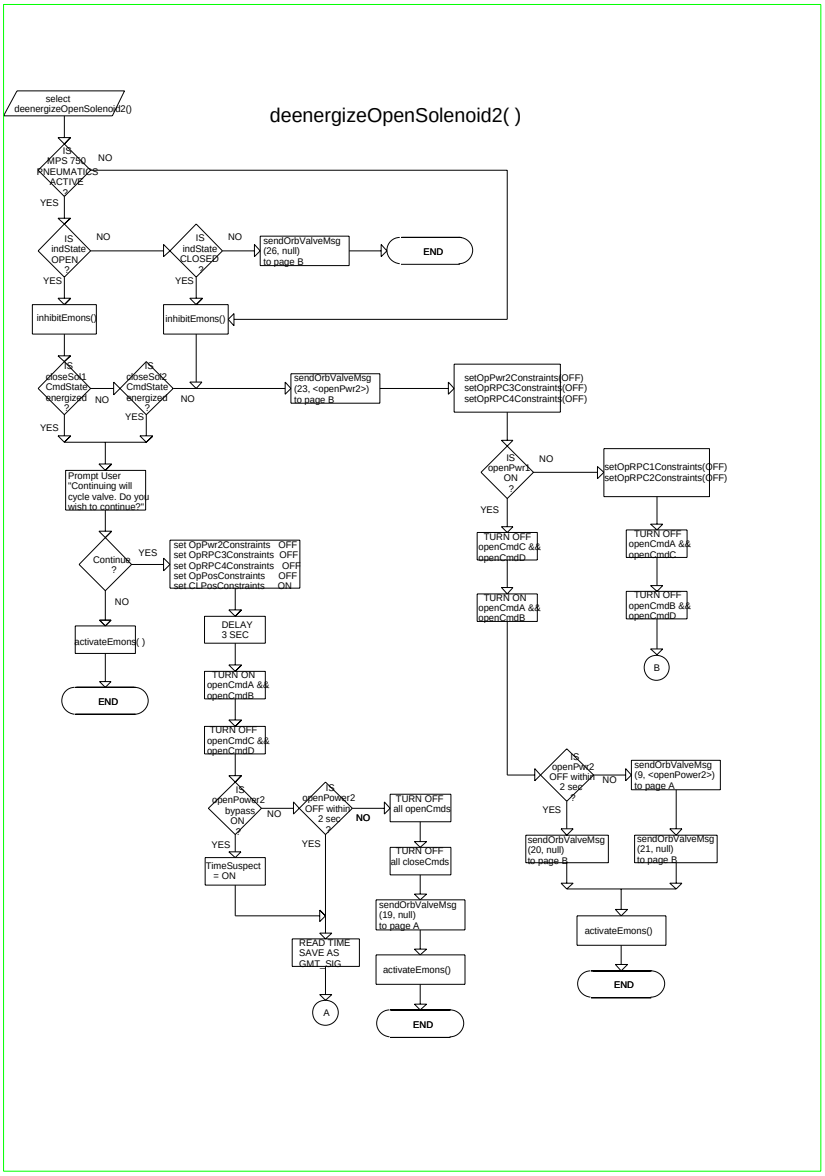


3.1.7.3.6. deenergizeOpenSolenoid2()

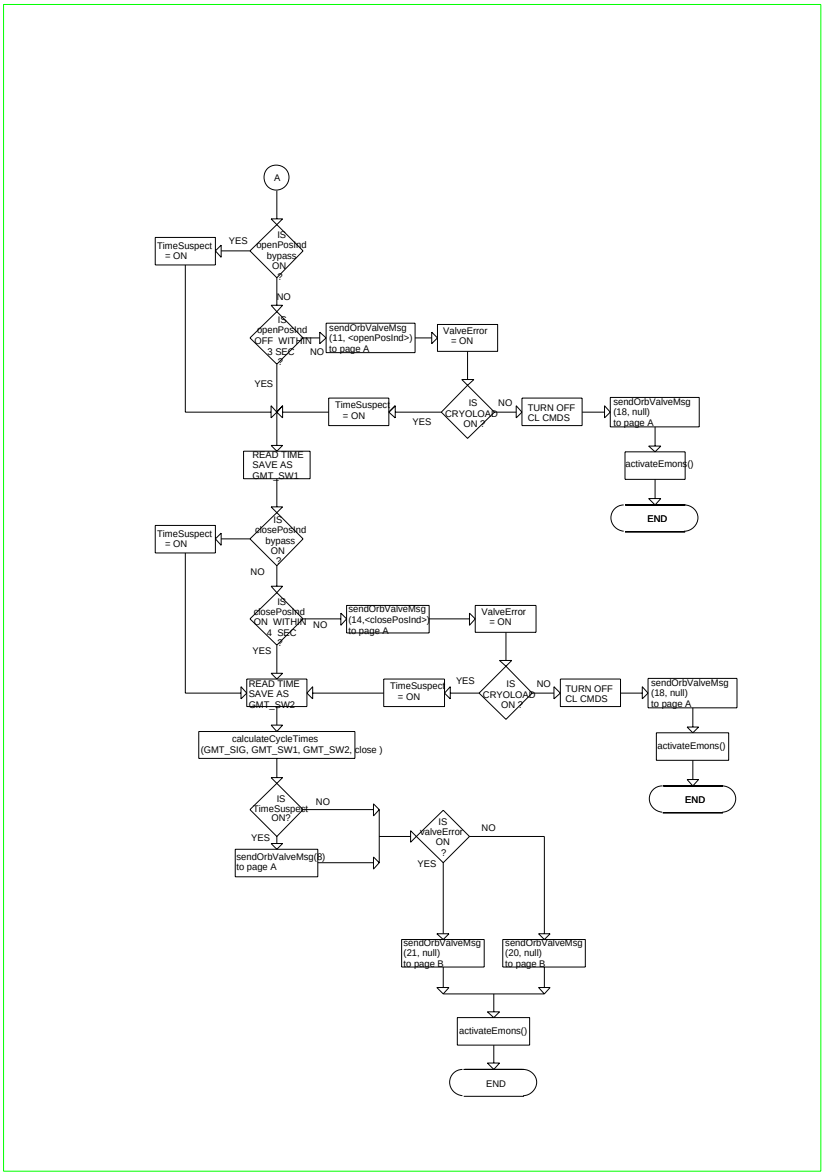
3.1.7.3.6.1. Descriptive Text

The flow charts contained below describe the logic to be used to code the method. The implementation of the details is left up to the coders and is acceptable as long as the overall logic is maintained. It is expected that cycle timing will be accomplished using the most accurate source. All messages will include a time stamp, and the appropriate FD description.

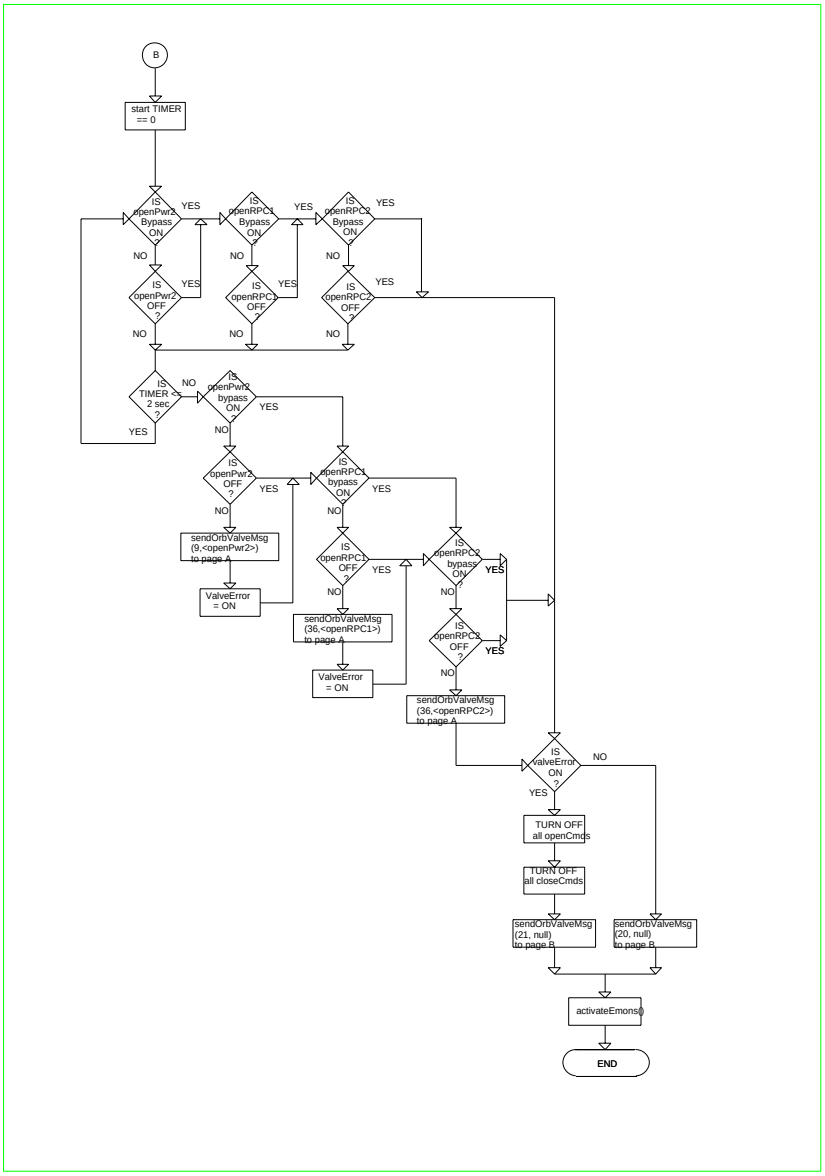
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

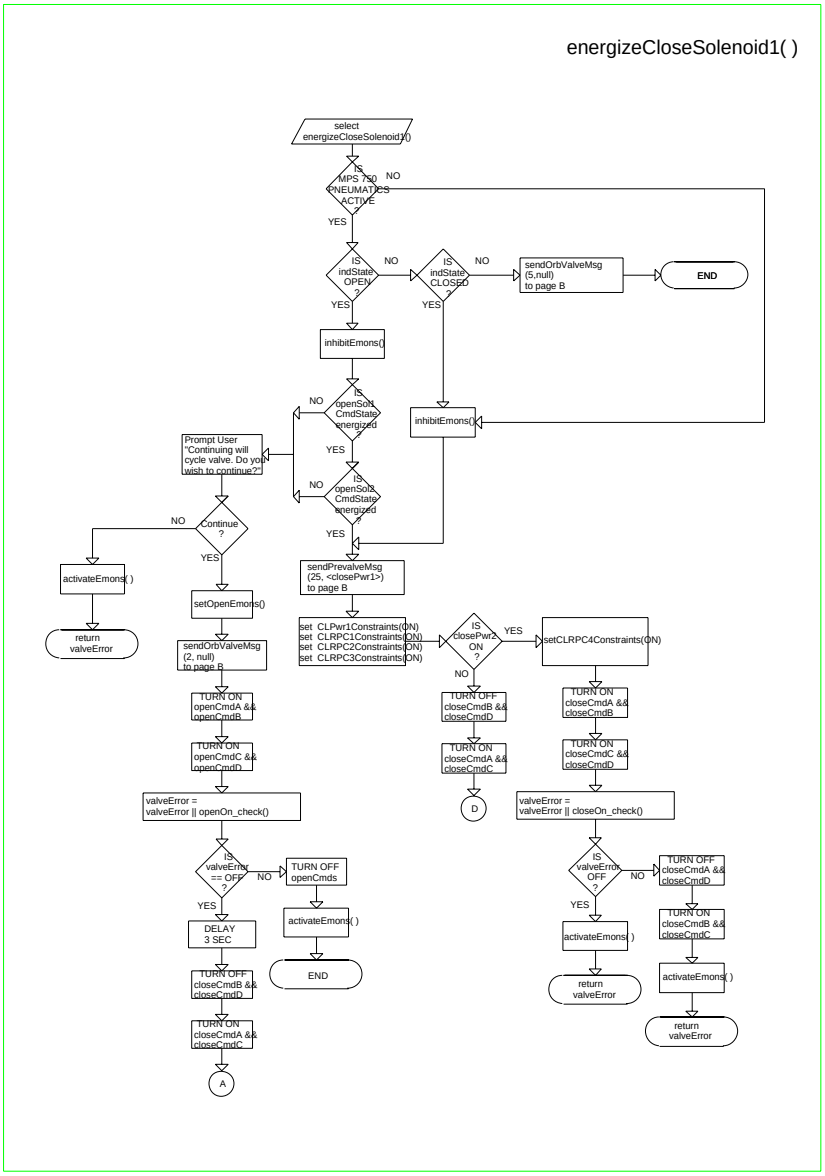


3.1.7.3.7. energizeCloseSolenoid1()

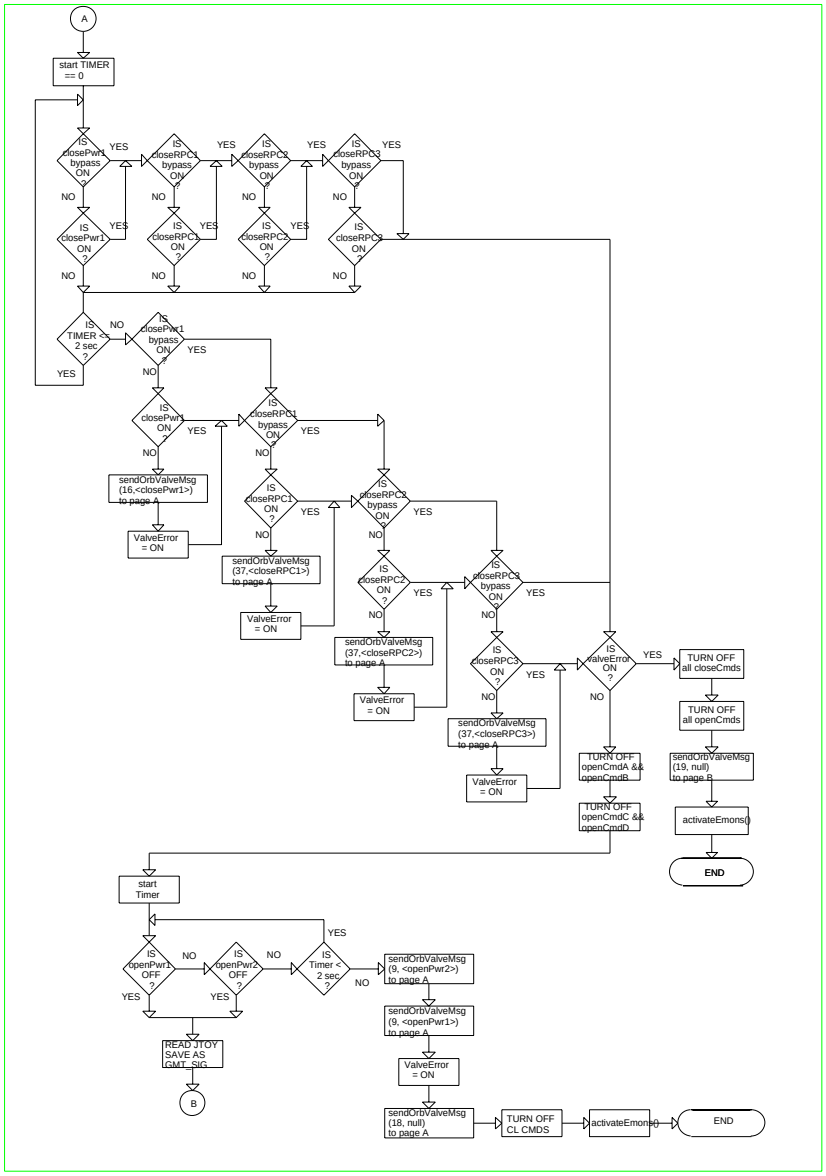
3.1.7.3.7.1. Descriptive Text

The flow charts contained below describe the logic to be used to code the method. The implementation of the details is left up to the coders and is acceptable as long as the overall logic is maintained. It is expected that cycle timing will be accomplished using the most accurate source. All messages will include a time stamp, and the appropriate FD description.

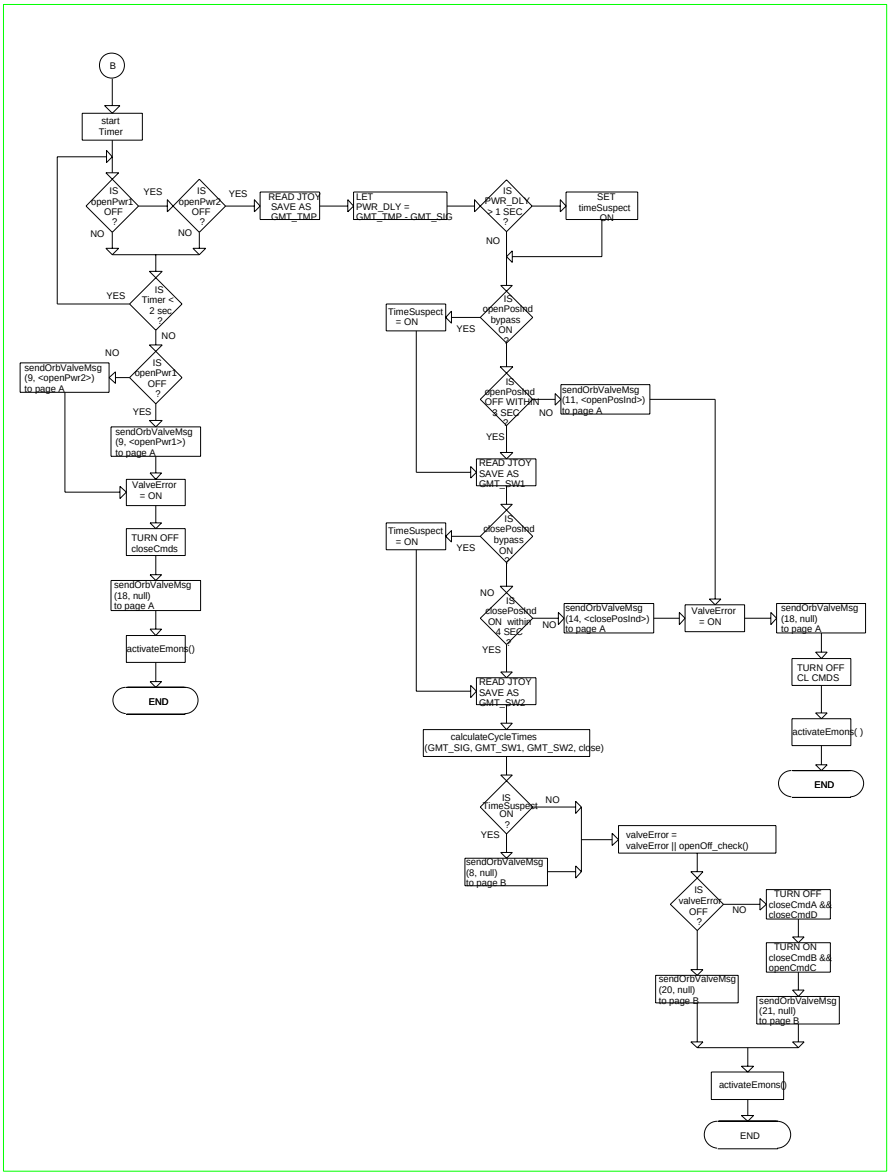
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



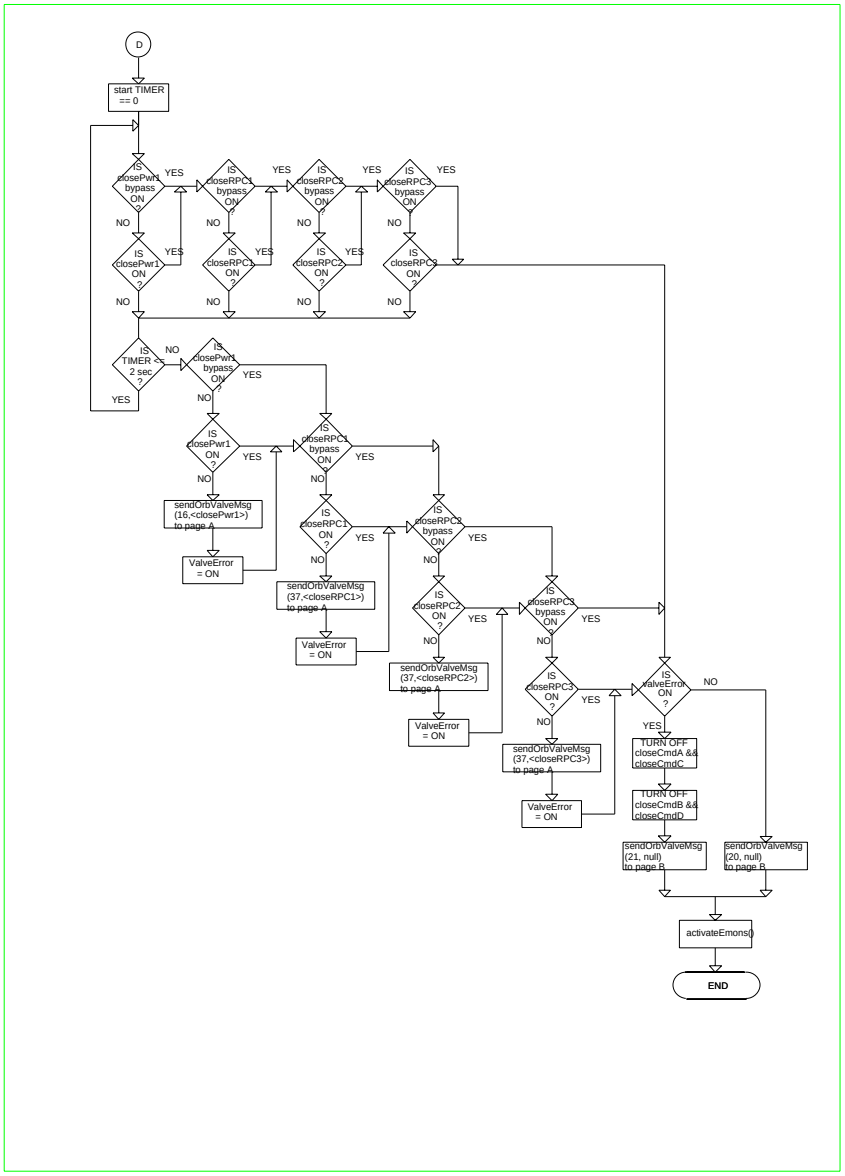
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

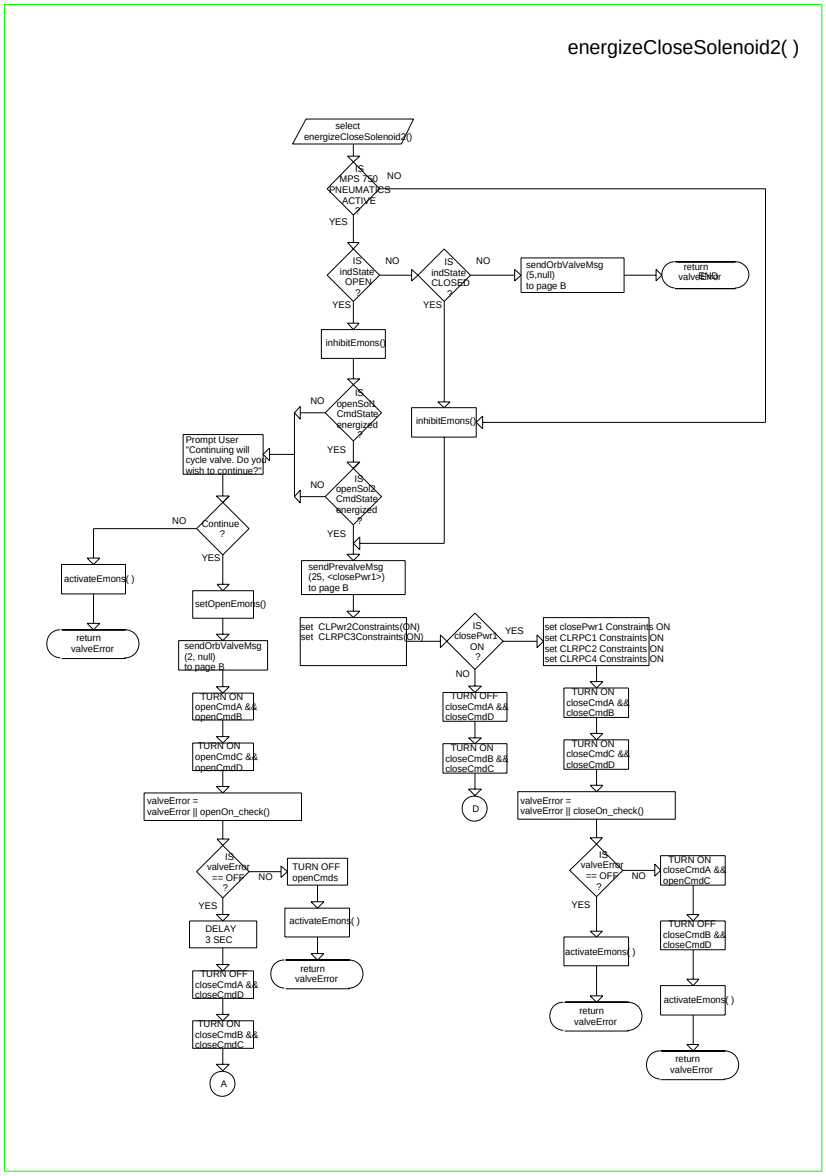


3.1.7.3.8. energizeCloseSolenoid2()

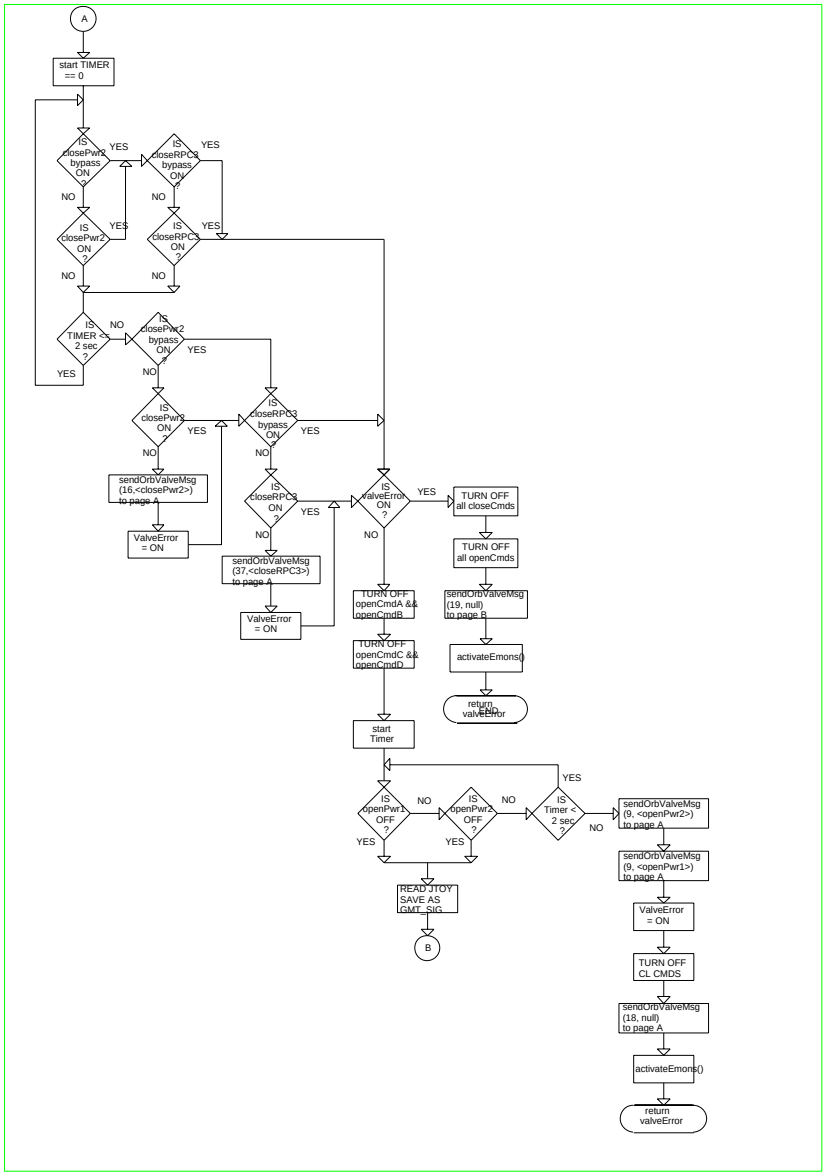
3.1.7.3.8.1. Descriptive Text

The flow charts contained below describe the logic to be used to code the method. The implementation of the details is left up to the coders and is acceptable as long as the overall logic is maintained. It is expected that cycle timing will be accomplished using the most accurate source. All messages will include a time stamp, and the appropriate FD description.

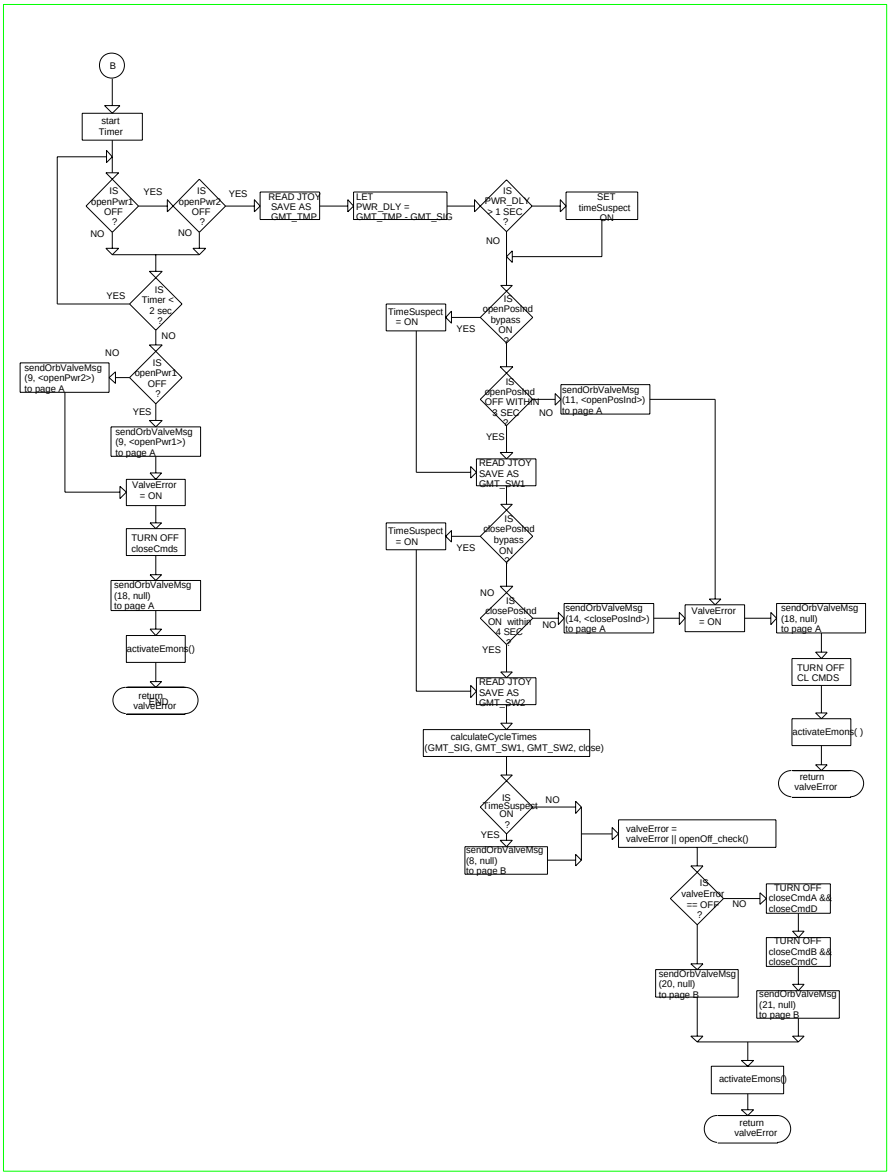
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



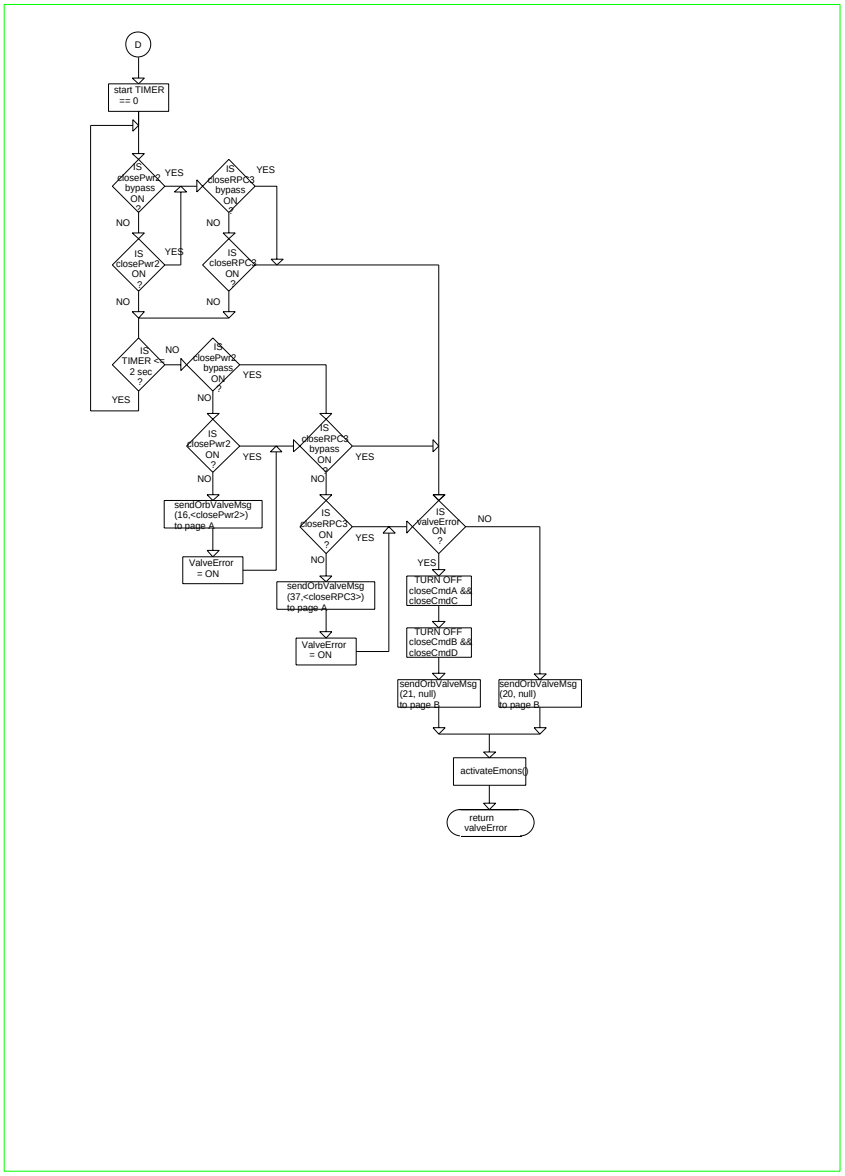
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

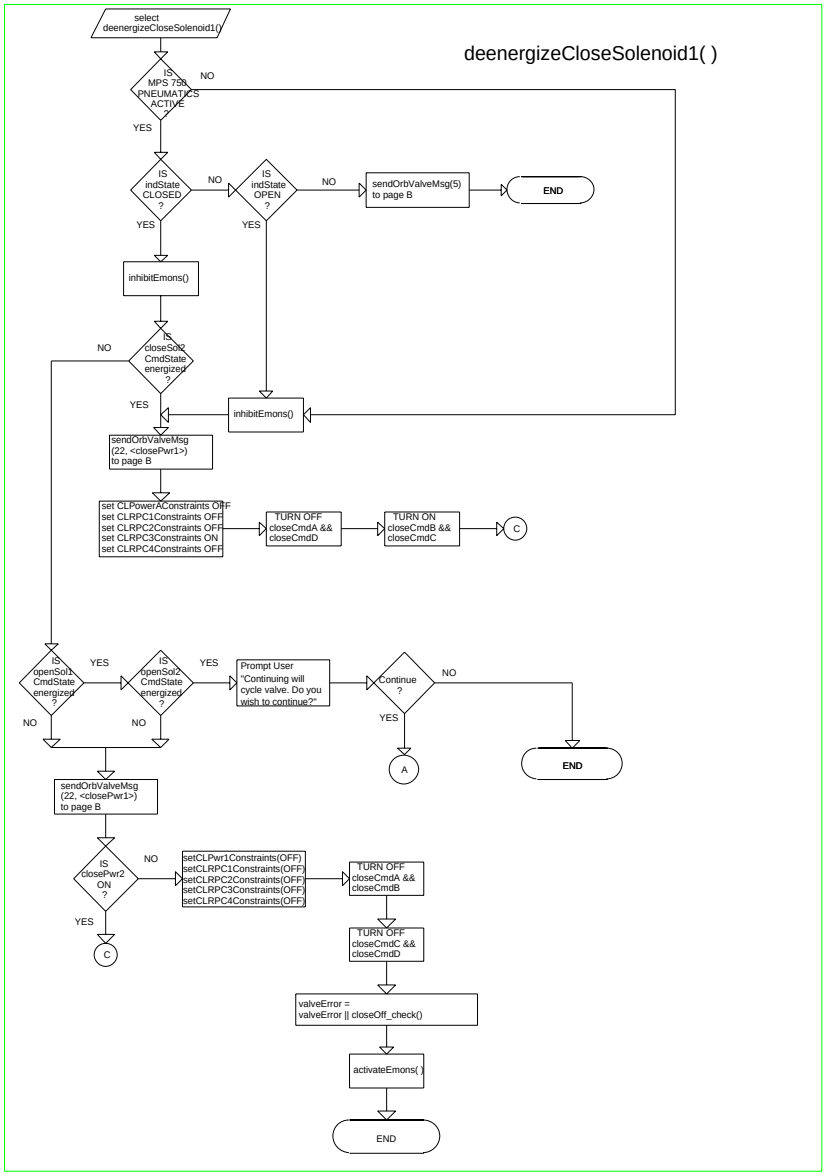


3.1.7.3.9. deenergizeCloseSolenoid1()

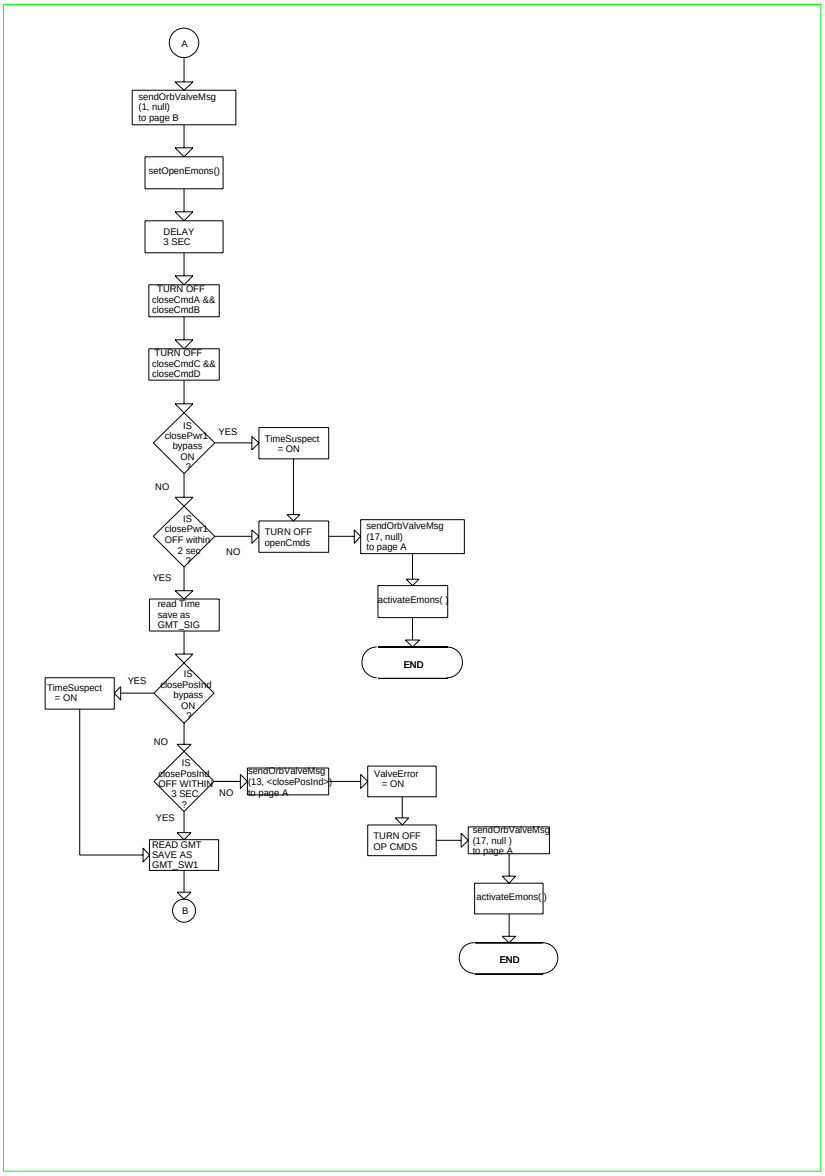
3.1.7.3.9.1. Descriptive Text

The flow charts contained below describe the logic to be used to code the method. The implementation of the details is left up to the coders and is acceptable as long as the overall logic is maintained. It is expected that cycle timing will be accomplished using the most accurate source. All messages will include a time stamp, and the appropriate FD description.

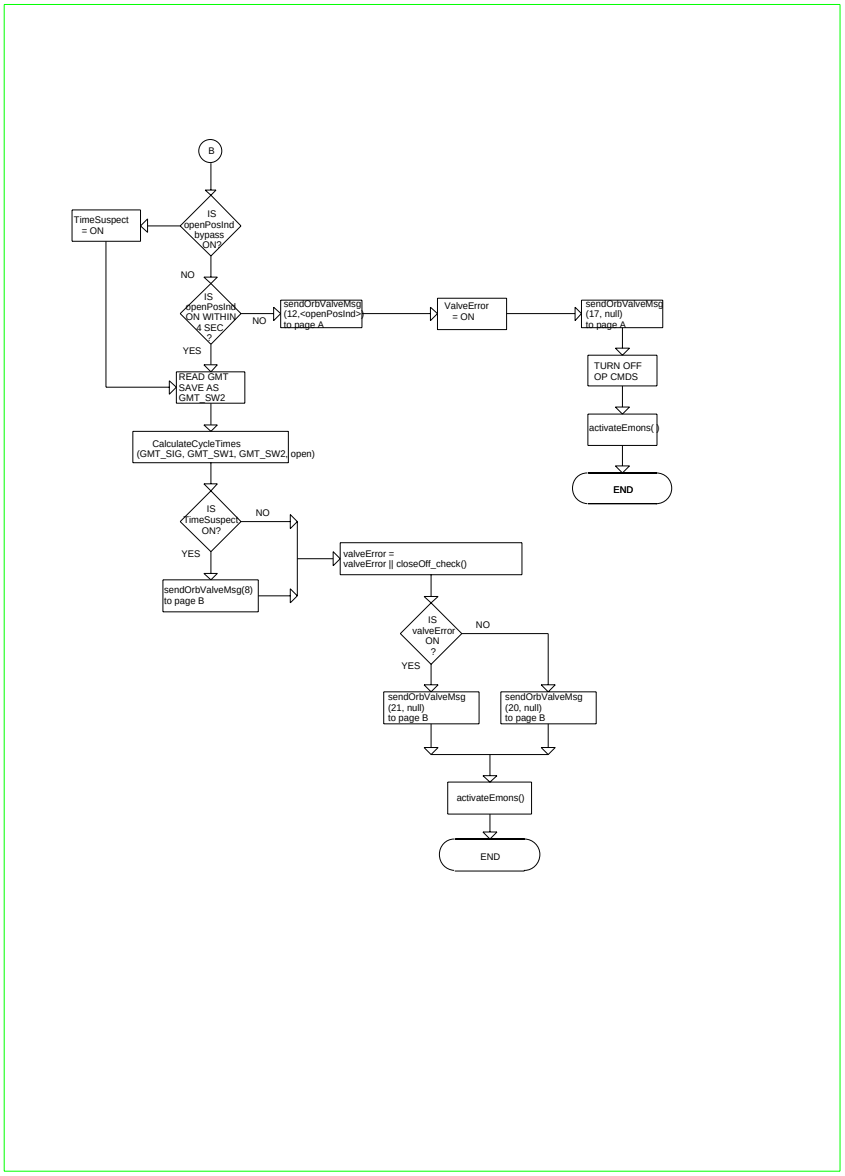
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



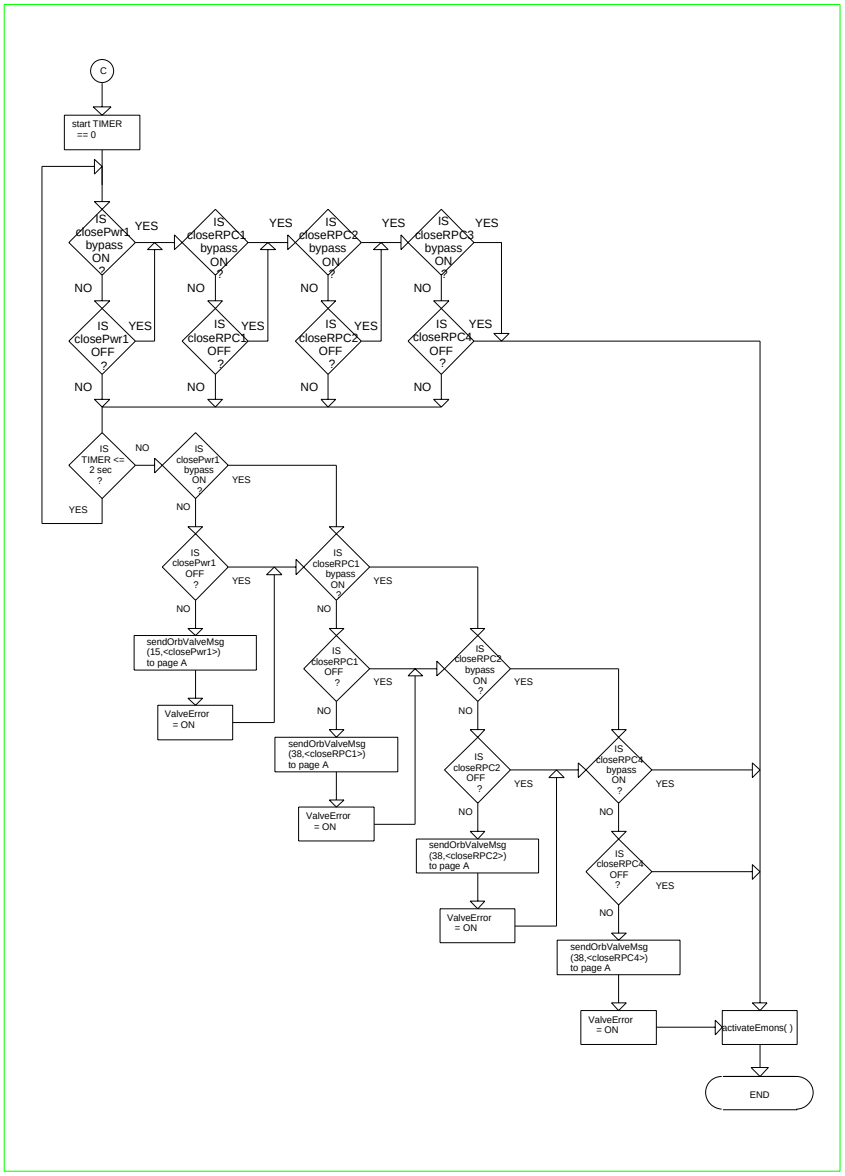
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

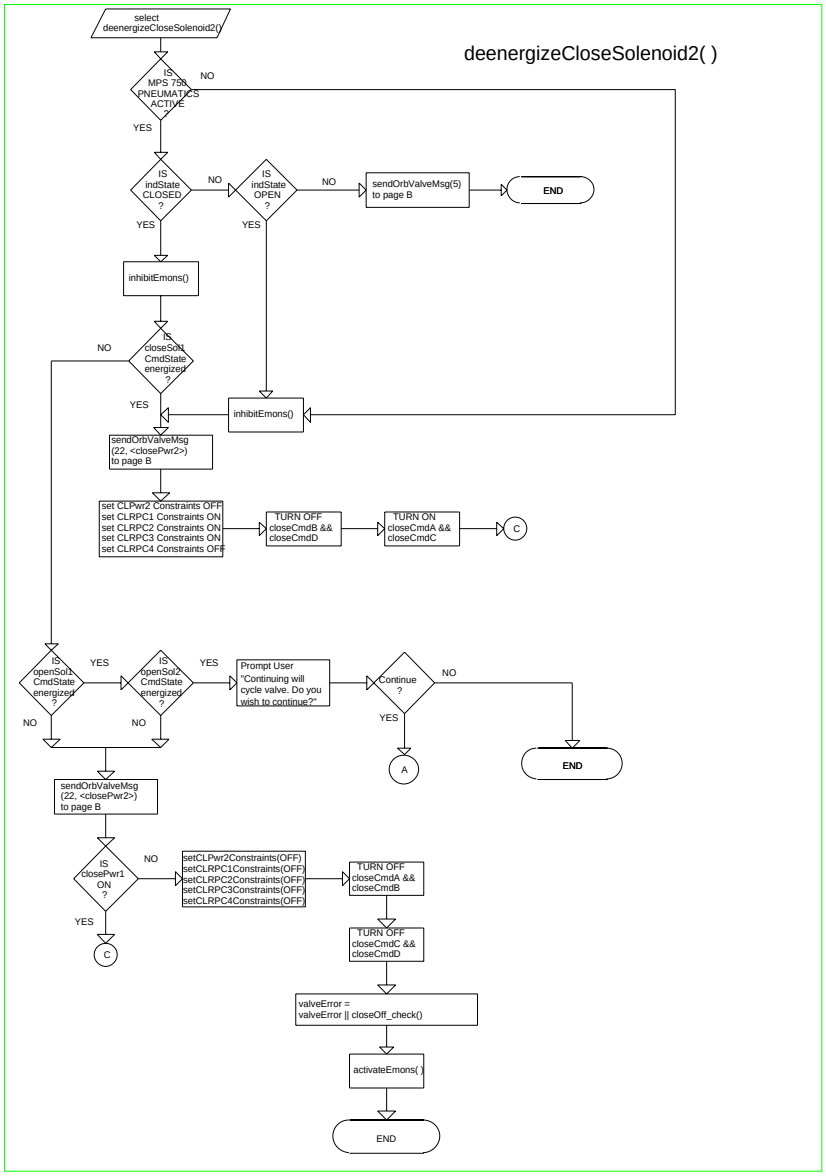


3.1.7.3.10. deenergizeCloseSolenoid2()

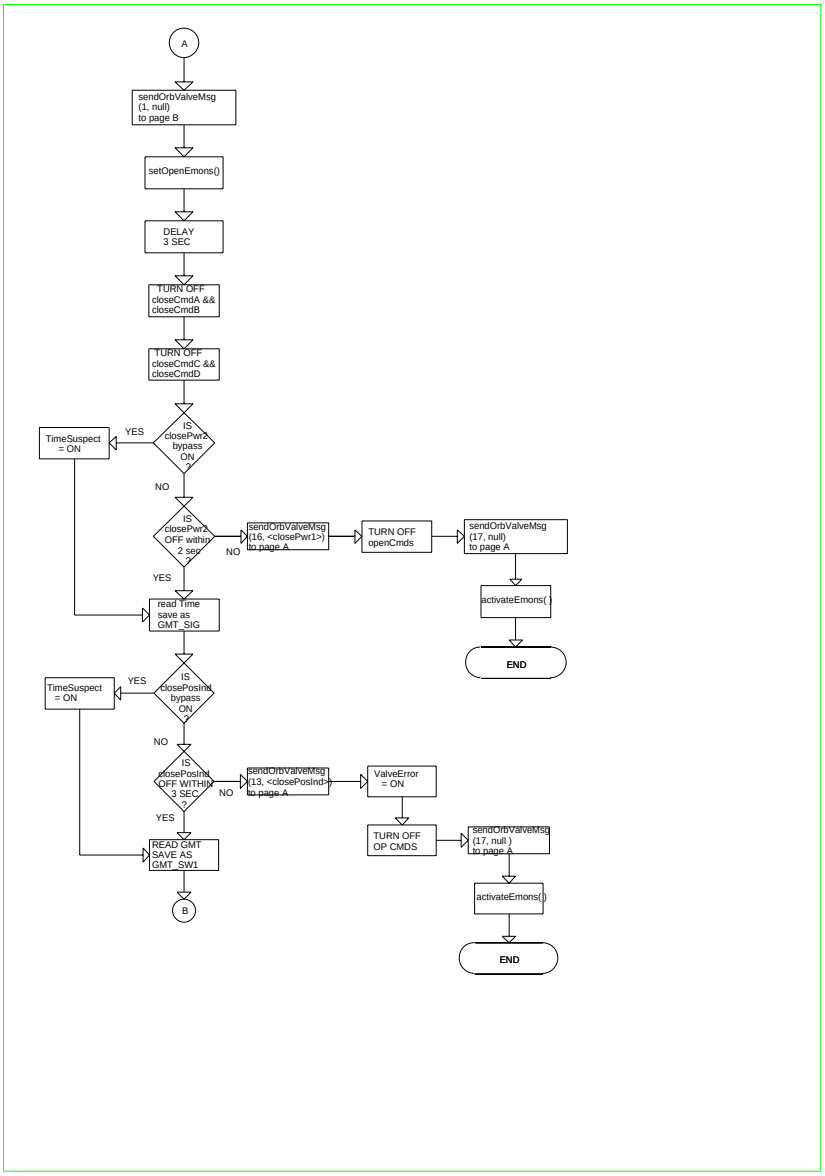
3.1.7.3.10.1. Descriptive Text

The flow charts contained below describe the logic to be used to code the method. The implementation of the details is left up to the coders and is acceptable as long as the overall logic is maintained. It is expected that cycle timing will be accomplished using the most accurate source. All messages will include a time stamp, and the appropriate FD description.

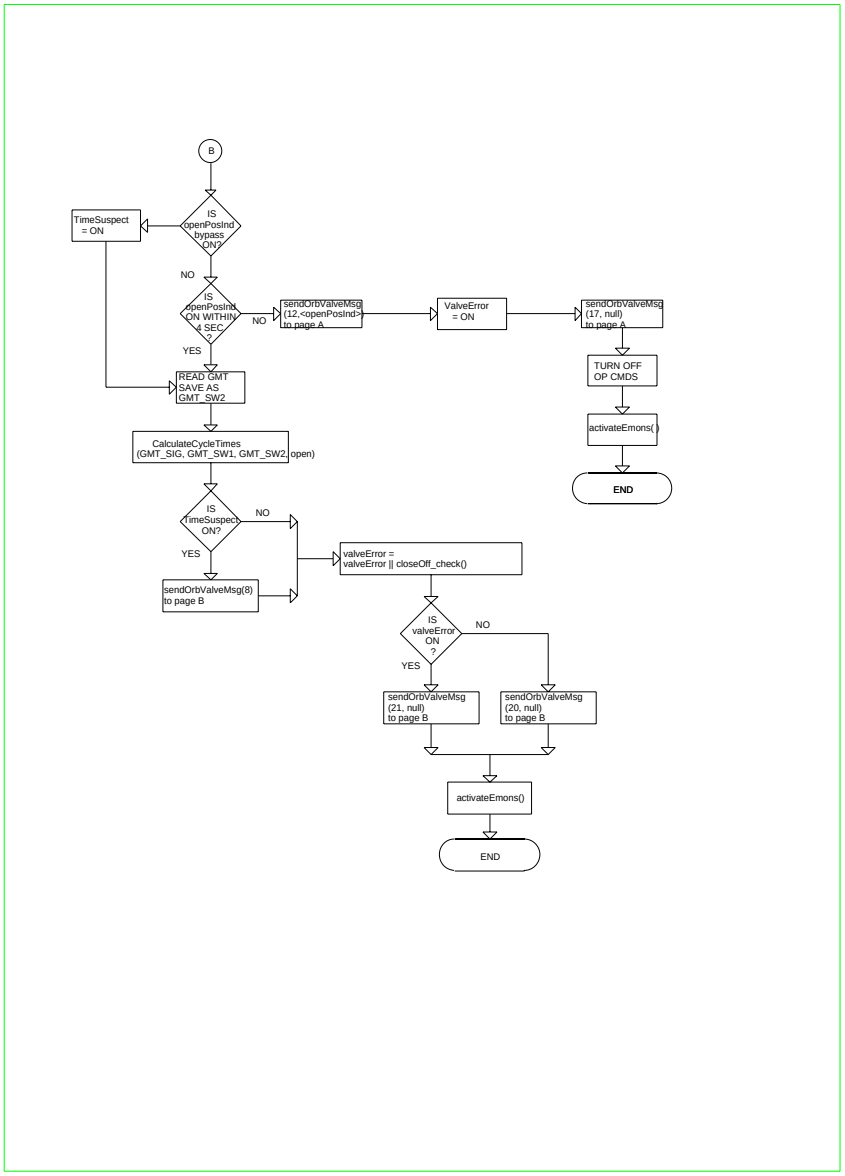
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



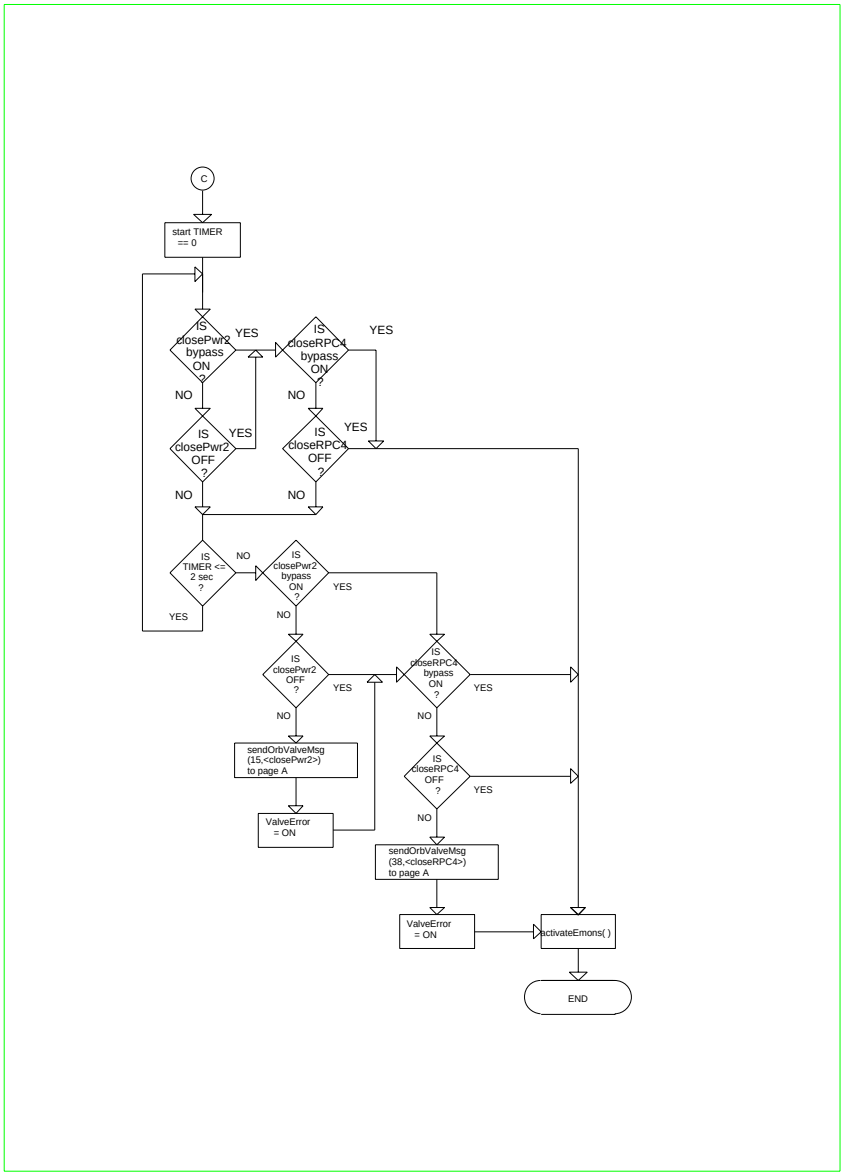
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

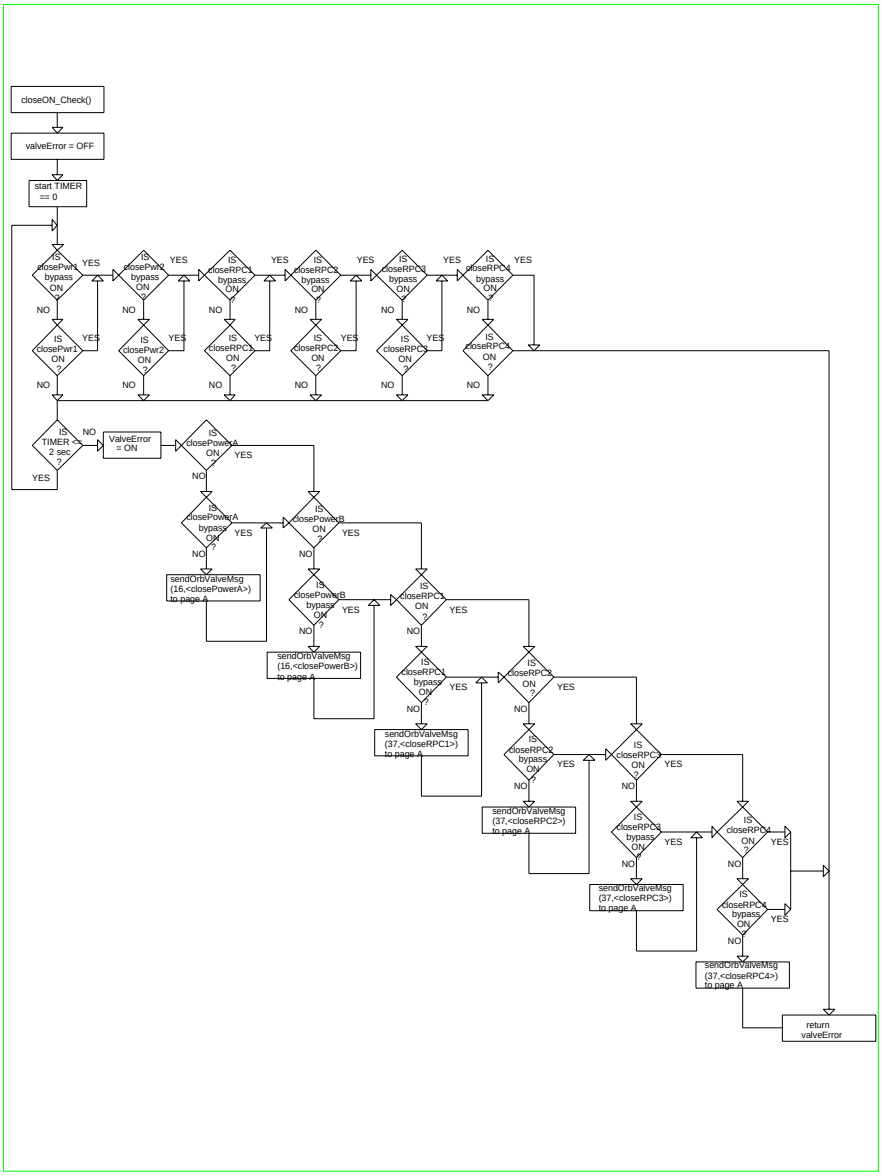


3.1.7.3.11. closeON_Check()

3.1.7.3.11.1. Descriptive Text

The closeON_Check() method is a private method that is called by the other valve cycle methods. The closeON_Check() method verifies that the unbypassed close powers and RPCs are energized within two seconds. The method sends appropiate messages for all powers or RPCs that do not energize within the allotted time. The flow charts shown below describe the logic to be used to code the method. The implementation of the details is left up to the coders and is acceptable as long as the overall logic is maintained. It is expected that cycle timing will be accomplished using the most accurate source. All messages will include a time stamp, and the appropriate FD description.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

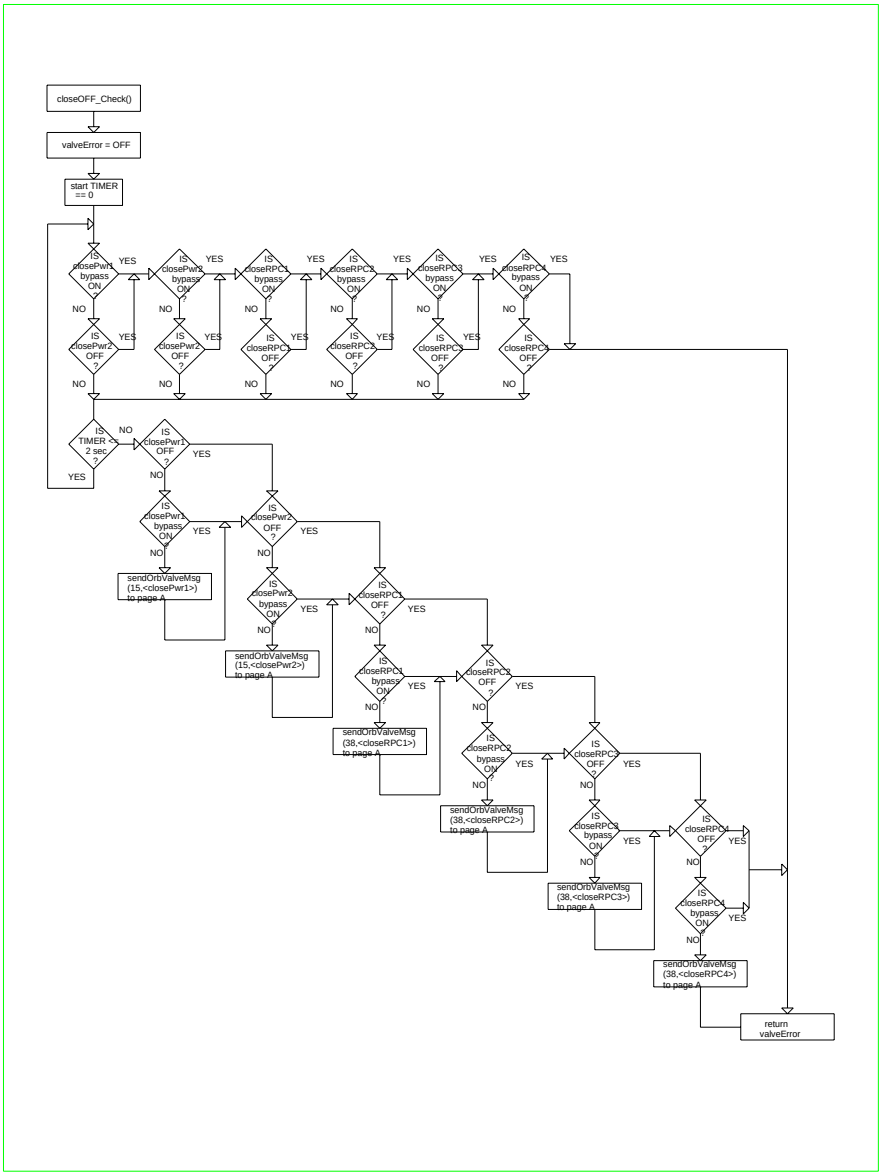


3.1.7.3.12. closeOff_Check()

3.1.7.3.12.1. Descriptive Text

The closeOff_Check() method is a private method that is called by the other valve cycle methods. The closeOff_Check() method verifies that the unbypassed close powers and RPCs are deenergized within two seconds. The method sends appropiate messages for all powers or RPCs that do not deenergize within the allotted time. The flow charts shown below describe the logic to be used to code the method. The implementation of the details is left up to the coders and is acceptable as long as the overall logic is maintained. It is expected that cycle timing will be accomplished using the most accurate source. All messages will include a time stamp, and the appropriate FD description.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

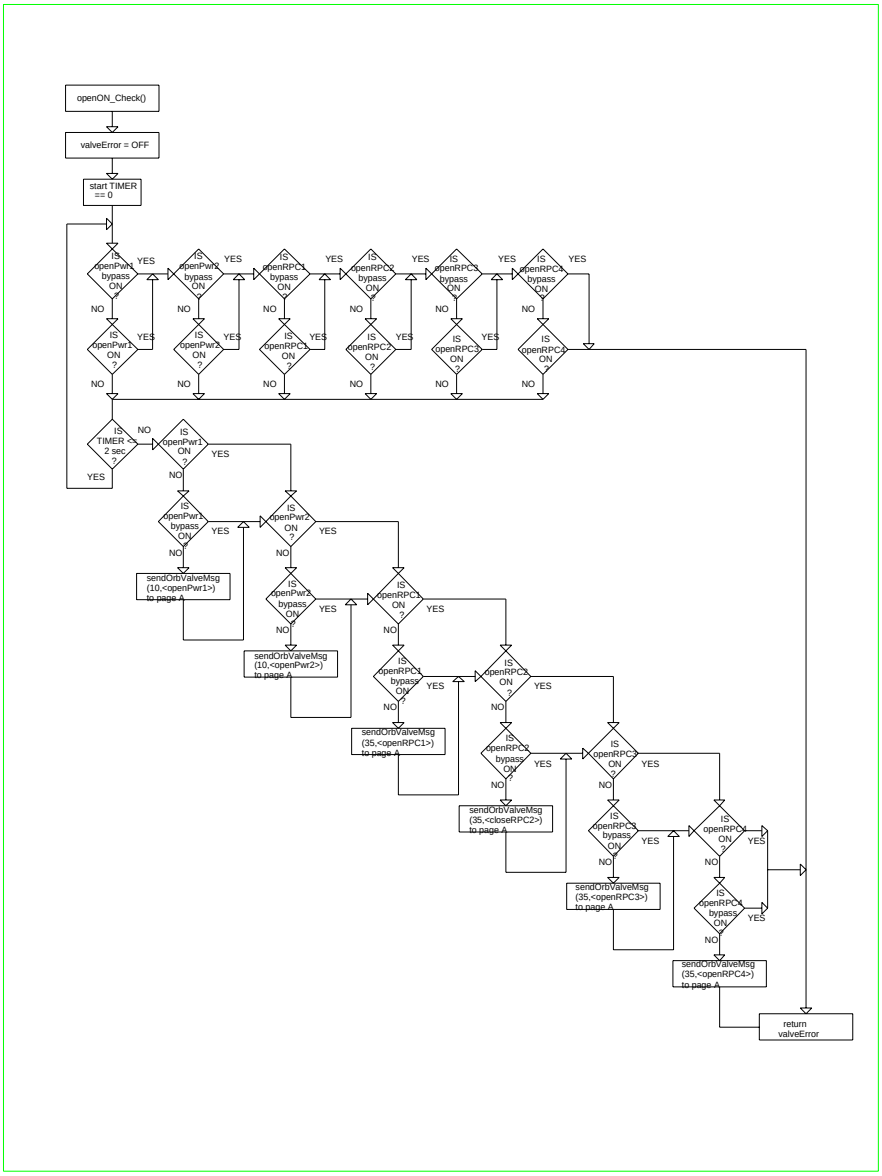


3.1.7.3.13. openON_check()

3.1.7.3.13.1. Descriptive Text

The openON_Check() method is a private method that is called by the other valve cycle methods. The openON_Check() method verifies that the unbypassed open powers and RPCs are energized within two seconds. The method sends appropiate messages for all open powers or RPCs that do not energize within the allotted time. The flow charts shown below describe the logic to be used to code the method. The implementation of the details is left up to the coders and is acceptable as long as the overall logic is maintained. It is expected that cycle timing will be accomplished using the most accurate source. All messages will include a time stamp, and the appropriate FD description.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

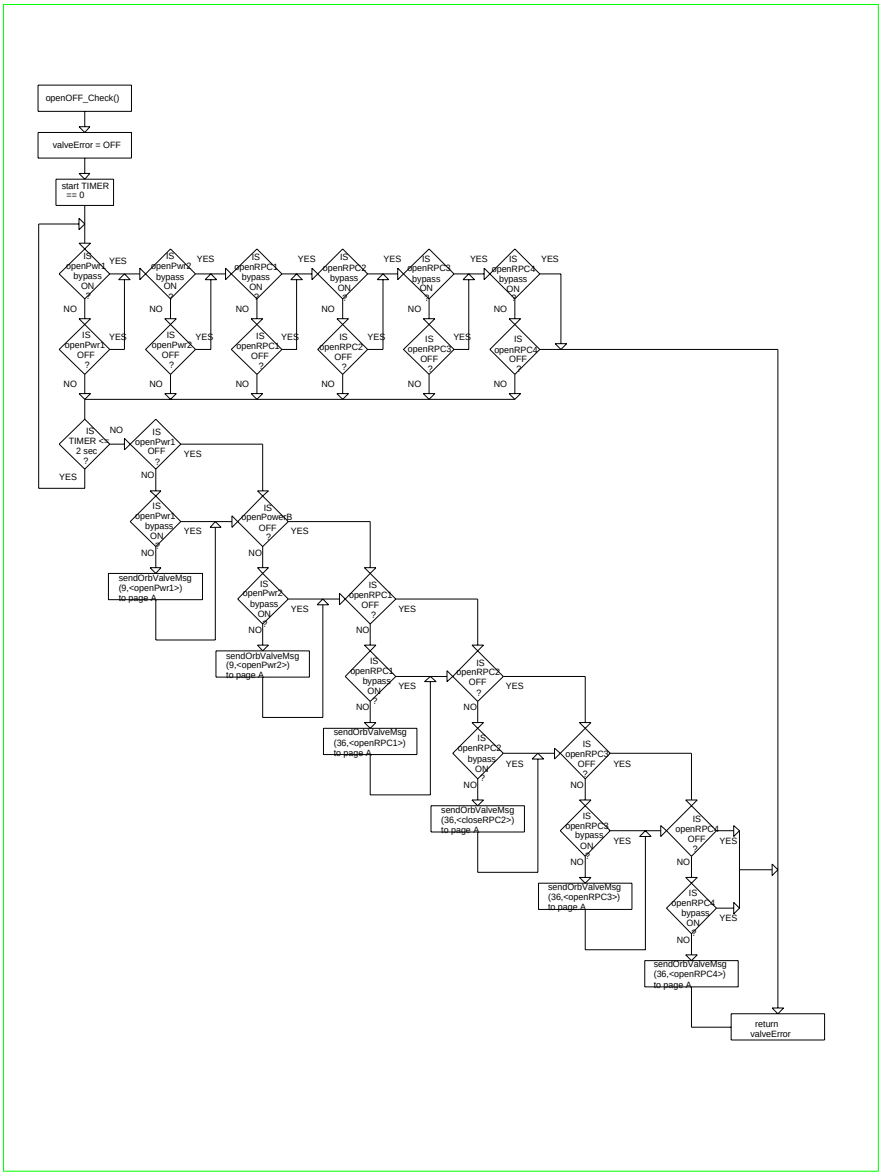


3.1.7.3.14. openOff_Check()

3.1.7.3.14.1. Descriptive Text

The openOff_Check() method is a private method that is called by the other valve cycle methods. The openOff_Check() method verifies that the unbypassed open powers and RPCs are deenergized within two seconds. The method sends appropiate messages for all close powers or RPCs that do not deenergize within the allotted time. The flow charts shown below describe the logic to be used to code the method. The implementation of the details is left up to the coders and is acceptable as long as the overall logic is maintained. It is expected that cycle timing will be accomplished using the most accurate source. All messages will include a time stamp, and the appropriate FD description.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



3.1.7.3.15. determineIndState()

3.1.7.3.15.1. Descriptive Text

This method will determine the valve's position based on its position indicators and associated indicator bypasses. The return value will be an enumerated type with possible values of OPEN, CLOSED, TRANSITION, and ERROR.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.7.3.15.2. Sequence

3.1.7.3.16. determineOpenSol1CmdState()

Indicator State				
open pos ind	**open pos byp**	**close pos ind**	**close pos byp**	**state**
OFF	OFF	ON	OFF	closed
OFF	OFF	----	ON	closed
----	ON	ON	OFF	closed
ON	OFF	OFF	OFF	open
----	ON	OFF	OFF	open
ON	OFF	----	ON	open
OFF	OFF	OFF	OFF	transition
----	ON	----	ON	error
ON	OFF	ON	OFF	error

3.1.7.3.16.1. Descriptive Text

This method will determine the commanded state of open solenoid 1 based on the open power. The return value will be an enumerated type with possible values of ENERGIZED, DEENERGIZED, and ERROR.

3.1.7.3.16.2. Effectivity

Operates on change data.

3.1.7.3.16.3. Sequence

The code for determineOpenSol1CmdState( ) will assign the value given by the logic below to appropriate pseudo FD.

if <openSol1Pwr> = ON, then openSol1CmdState = energized

if <openSol1Pwr> = OFF, then openSol1CmdState = deenergized

if <openSol1Pwr>.Bypass = ON, then openSol1CmdState = error

3.1.7.3.17. determineOpenSol2CmdState()

3.1.7.3.17.1. Descriptive Text

This method will determine the commanded state of open solenoid 2 based on the open power. The return value will be an enumerated type with possible values of ENERGIZED, DEENERGIZED and ERROR.

3.1.7.3.17.2. Sequence

The code for determineOpenSol2CmdState( ) will assign the value given by the logic below to appropriate pseudo FD.

if <openSol2Pwr> = ON, then openSol2CmdState = energized

if <openSol2Pwr> = OFF, then openSol2CmdState = deenergized

if <openSol2Pwr>.Bypass = ON, then openSol2CmdState = error

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.7.3.18. determineCloseSol1CmdState()

3.1.7.3.18.1. Descriptive Text

This method will determine the commanded state of close solenoid 1 based on the open power. The return value will be an enumerated type with possible values of ENERGIZED, DEENERGIZED and ERROR.

3.1.7.3.18.2. Sequence

The code for determineCloseSol1CmdState( ) will assign the value given by the logic below to appropriate pseudo FD.

if <closeSol1Pwr> = ON, then closeSol1CmdState = energized

if <closeSol1Pwr> = OFF, then closeSol1CmdState = deenergized

if <closeSol1Pwr>.Bypass = ON, then closeSol1CmdState = error

3.1.7.3.19. determineCloseSol2CmdState()

3.1.7.3.19.1. Descriptive Text

This method will determine the commanded state of close solenoid 2 based on the open power. The return value will be an enumerated type with possible values of ENERGIZED, DEENERGIZED.

3.1.7.3.19.2. Effectivity

Operates on change data.

3.1.7.3.19.3. Sequence

The code for determineCloseSol2CmdState( ) will assign the value given by the logic below to appropriate pseudo FD.

if <closeSol2Pwr> = ON, then closeSol2CmdState = energized

if <closeSol2Pwr> = OFF, then closeSol2CmdState = deenergized

if <closeSol1Pwr>.Bypass = ON, then closeSol1CmdState = error

3.1.7.3.20. determineCmdState()

3.1.7.3.20.1. Descriptive Text

This method will determine the valve's overall commanded state based on closeSol1CmdState, closeSol2CmdState, openSol1CmdState, and openSol2CmdState. The return value will be an enumerated type with possible values of CMD_CLOSED, CMD_OPEN, UNCOMMANDED, CMD_SNUBBED, and ERROR.

3.1.7.3.20.2. Effectivity

Operates on change data.

3.1.7.3.20.3. Sequence

Case:

<openSol1CmdState> == ENERGIZED
<openSol2CmdState> == ENERGIZED
<closeSol1CmdState> == DEENERGIZED
<closeSol2CmdState> == DEENERGIZED
cmdState = CMD_OPEN;

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

<openSol1CmdState> == ENERGIZED
<openSol2CmdState> == ENERGIZED
<closeSol1CmdState> == ENERGIZED
<closeSol2CmdState> == DEENERGIZED
cmdState =CMD_SNUBBED;

<openSol1CmdState> == ENERGIZED
<openSol2CmdState> == ENERGIZED
<closeSol1CmdState> == DEENERGIZED
<closeSol2CmdState> == ENERGIZED
cmdState = CMD_SNUBBED;

<openSol1CmdState> == ENERGIZED
<openSol2CmdState> == ENERGIZED
<closeSol1CmdState> == ENERGIZED
<closeSol2CmdState> == ENERGIZED
cmdState = CMD_SNUBBED;

<openSol1CmdState> == DEENERGIZED
<openSol2CmdState> == ENERGIZED
<closeSol1CmdState> == ENERGIZED
<closeSol2CmdState> == ENERGIZED
cmdState = CMD_CLOSED;

<openSol1CmdState> == ENERGIZED
<openSol2CmdState> == DEENERGIZED
<closeSol1CmdState> == ENERGIZED
<closeSol2CmdState> == ENERGIZED
cmdState = CMD_CLOSED;

<openSol1CmdState> == DEENERGIZED
<openSol2CmdState> == DEENERGIZED
<closeSol1CmdState> == ENERGIZED
<closeSol2CmdState> == ENERGIZED
cmdState = CMD_CLOSED;

<openSol1CmdState> == DEENERGIZED
<openSol2CmdState> == DEENERGIZED
<closeSol1CmdState> == DEENERGIZED
<closeSol2CmdState> == DEENERGIZED
cmdState = UNCOMMANDED;

End Case:

3.1.7.3.21. determineState()

3.1.7.3.21.1. Descriptive Text

3.1.7.3.21.2. Sequence

case:

<cmdState> == CMD_CLOSED
<indState> == CLOSED state=CLOSED;

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
<cmdState> == CMD_CLOSED
<indState> == OPEN                  state=OPEN_ERROR;

<cmdState> == CMD_CLOSED
<indState> == TRANSITION            state=TRANSITION;

<cmdState> == CMD_OPEN
<indState> == CLOSED                state=CLOSED_ERROR;

<cmdState> == CMD_OPEN
<indState> == OPEN                  state=OPEN;

<cmdState> == CMD_OPEN
<indState> == TRANSITION            state=TRANSITION;

<cmdState> == CMD_SNUBBED
<indState> == CLOSED                state=CLOSED_ERROR;

<cmdState> == CMD_SNUBBED
<indState> == OPEN                  state=OPEN_ERROR;

<cmdState> == CMD_SNUBBED
<indState> == TRANSITION            state=ERROR;

<cmdState> == UNCOMMANDED
<indState> == CLOSED                state=CLOSED_ERROR;

<cmdState> == UNCOMMANDED
<indState> == OPEN                  state=OPEN_ERROR;

<cmdState> == UNCOMMANDED
<indState> == TRANSITION            state=ERROR;


<indState> == ERROR                 state = ERROR;

CASE ELSE                           state = ERROR;



End Case;
```

3.1.7.3.22. getIndState()

3.1.7.3.22.1. Descriptive Text

This method will return the Enumerated Indicator State of the valve

3.1.7.3.22.2. Sequence

```
getIndState()
{
        return indState;
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.1.7.3.23. getOpenSolCmdState()

3.1.7.3.23.1. Descriptive Text

This method will return the Enumerated Command State of the valve.

3.1.7.3.23.2. Sequence

```
getOpenSolCmdState()
{
        return openSolCmdState;
}
```

3.1.7.3.24. getCloseSolCmdState()

3.1.7.3.24.1. Descriptive Text

This method will return the Enumerated Command State of the valve.

3.1.7.3.24.2. Sequence

```
getCloseSolCmdState()
{
        return closeSolCmdState;
}
```

3.1.7.3.25. verifyCommandsOff()

3.1.7.3.25.1. Descriptive Text

This function will verify that the open and close solenoid commands are off by performing a Byte Test 4 for each command.

3.1.7.3.25.2. Sequence

```
verifyCommandsOff()
{
open command status 1 = byteTest4 <openCMDA>
        {
        verify  open command status 1 == OFF
        else print message to page A :   "Byte Test 4 shows", valveID, "open command" <openCMDA> "to be ON"

open command status 2 = byteTest4 <openCMDB>
        {
        verify  open command status 2 == OFF
        else print message to page A :   "Byte Test 4 shows", valveID, "open command" <openCMDB> "to be ON"

open command status 3 = byteTest4 <openCMDC>
        {
        verify  open command status 3 == OFF
        else print message to page A :   "Byte Test 4 shows", valveID, "open command" <openCMDC> "to be ON"

open command status 4 = byteTest4 <openCMDD>
        {
        verify  open command status 4 == OFF
        else print message to page A :   "Byte Test 4 shows", valveID, "open command" <openCMDD> "to be ON"

close command status 1 = byteTest4 <closeCMDA>
        {
        verify  close command status 1 == OFF
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
        else print message to page A :   "Byte Test 4 shows", valveID, "close command" <closeCMDA> "to be
ON"

close command status 2 = byteTest4 <closeCMDB>
        {
        verify  close command status 2 == OFF
        else print message to page A :   "Byte Test 4 shows", valveID, "close command" <closeCMDB> "to be
ON"

close command status 3 = byteTest4 <closeCMDC>
        {
        verify  close command status 3 == OFF
        else print message to page A :   "Byte Test 4 shows", valveID, "close command" <closeCMDC> "to be
ON"

close command status 4 = byteTest4 <closeCMDD>
        {
        verify  close command status 4 == OFF
        else print message to page A :   "Byte Test 4 shows", valveID, "close command" <closeCMDD> "to be
ON"

        }
mainstage command status 1 = byteTest4 <mainstageCMDA>
        {
        verify  mainstage command status 1 == OFF
        else print message to page A :   "Byte Test 4 shows", valveID, "mainstage command" <mainstageCMDA>
"to be ON"

        }
mainstage command status 2 = byteTest4 <mainstageCMDB>
        {
        verify mainstage command status 2  == OFF
        else print message to page A :   "Byte Test 4 shows", valveID, "mainstage command" <mainstageCMDB>
"to be ON"

        }
mainstage command status 3 = byteTest4 <mainstageCMDC>
        {
        verify mainstage command status 3 == OFF
        else print message to page A :   "Byte Test 4 shows", valveID, "mainstage command" <mainstageCMDC>
"to be ON"

        }
{
        verify  close command status 1 && close command status 2 && close command status 3 && close
command
        status 4 && open command status 1 && open command status 2 && open command status 3 && open
command status 4
        && mainstage command status 1 && mainstage command status 2 && mainstage command status 3
         == OFF
        Return TRUE;

        Else return FALSE;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
}
end sequence;
```

3.1.7.3.26. getCmdState()

3.1.7.3.26.1. Descriptive Text

This method will return the Enumerated Command State of the valve.

3.1.7.3.26.2. Sequence

```
getCmdState()
{
        return cmdState;
}
```

3.1.7.3.27. getState()

3.1.7.3.27.1. Descriptive Text

This method will return the overall Enumerated State of the valve.

3.1.7.3.27.2. Sequence

```
getState()
{
        return state;
}
```

3.1.7.3.28. autoCycle()

3.1.7.3.28.1. Descriptive Text

The following will be called by the Panel Class autoValveCheckout method to calculate and record the valve cycle times.

3.1.7.3.28.2. Sequence

```
Set autoValveCheckoutError = OFF;
Set autocycle = ON;

If indState = OPEN then
        {
        //Close using closeSol 1//

        If <openSol1CmdState> = Energized
        deenergizeOpenSol1();
        If <openSol2CmdState> = Energized
        deenergizeOpenSol2();

        energizeCloseSolenoid1();

        if valveCycleError is ON, Set autoValveCheckoutError = ON;

        open();

        if valveCycleError is ON, Set autoValveCheckoutError = ON;

        //Close using closeSol 2//

        If <openSol1CmdState> = Energized
        deenergizeOpenSol1();
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
        If <openSol2CmdState> = Energized
        deenergizeOpenSol2();

        energizeCloseSolenoid2();

        if valveCycleError is ON, Set autoValveCheckoutError = ON;


        open();

        if valveCycleError is ON, Set autoValveCheckoutError = ON;

        }
Else
        {
        open();

        if valveCycleError is ON, Set autoValveCheckoutError = ON;

        //Close using closeSol 1//

        If <openSol1CmdState> = Energized
        deenergizeOpenSol1();
        If <openSol2CmdState> = Energized
        deenergizeOpenSol2();

        energizeCloseSolenoid1();

        if valveCycleError is ON, Set autoValveCheckoutError = ON;

        open();

        if valveCycleError is ON, Set autoValveCheckoutError = ON;

        //Close using closeSol 2//

        If <openSol1CmdState> = Energized
        deenergizeOpenSol1();
        If <openSol2CmdState> = Energized
        deenergizeOpenSol2();

        energizeCloseSolenoid2();

        if valveCycleError is ON, Set autoValveCheckoutError = ON;

        set autocycle OFF
        }

Return autoValveCheckoutError;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.1.7.3.29. verifyTminus3BT4()//

3.1.7.3.29.1. Description

This method performs a byte test 4 of prevalve commands to verify they are in a proper T-3 hour configuration. LO2 prevalves are closed prior to start of load.

3.1.7.3.29.2. Sequence

begin sequence;

```
set commanderrorcheck OFF
{
perform ByteTest4 of <openCMDA>
        verify ByteTest4 = OFF
        else send message to page A
                "LO2 Prevalve" <openCMDA> "is not in T-3 hour configuration."
                and set commanderrorcheck ON

perform ByteTest4 of <openCMDB>
        verify ByteTest4 = OFF
        else send message to page A
                "LO2 Prevalve" <openCMDB> "is not in T-3 hour configuration."
                and set commanderrorcheck ON

perform ByteTest4 of <openCMDC>
        verify ByteTest4 = OFF
        else send message to page A
                "LO2 Prevalve" <openCMDC> "is not in T-3 hour configuration."
                and set commanderrorcheck ON

perform ByteTest4 of <openCMDD>
        verify ByteTest4 = OFF
        else send message to page A
                "LO2 Prevalve" <openCMDD> "is not in T-3 hour configuration."
                and set commanderrorcheck ON
}
{
perform ByteTest4 of <closeCMDA>
        verify ByteTest4 = ON
        else send message to page A
                "LO2 Prevalve" <closeCMDA> "is not in T-3 hour configuration."
                and set commanderrorcheck ON

perform ByteTest4 of <closeCMDB>
        verify ByteTest4 = ON
        else send message to page A
                "LO2 Prevalve" <closeCMDB> "is not in T-3 hour configuration."
                and set commanderrorcheck ON

perform ByteTest4 of <closeCMDC>
        verify ByteTest4 = ON
        else send message to page A
                "LO2 Prevalve" <closeCMDC> "is not in T-3 hour configuration."
                and set commanderrorcheck ON
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
perform ByteTest4 of <closeCMDD>
        verify ByteTest4 = ON
        else send message to page A
                "LO2 Prevalve" <closeCMDD> "is not in T-3 hour configuration."
                and set commanderrorcheck ON

perform ByteTest4 of <mainstageCMDA>
        verify ByteTest4 = OFF
        else send message to page A
                "LO2 Prevalve" <mainstageCMDA> "is not in T-3 hour configuration."
                and set commanderrorcheck ON

perform ByteTest4 of <mainstageCMDB>
        verify ByteTest4 = OFF
        else send message to page A
                "LO2 Prevalve" <mainstageCMDB> "is not in T-3 hour configuration."
                and set commanderrorcheck ON

perform ByteTest4 of <mainstageCMDC>
        verify ByteTest4 = OFF
        else send message to page A
                "LO2 Prevalve" <mainstageCMDC> "is not in T-3 hour configuration."
                and set commanderrorcheck ON

}
return commanderrorcheck

end sequence;
```

3.1.7.3.30. verifyTminus6BT4()//

3.1.7.3.30.1. Description

This method performs a byte test 4 of prevalve commands to verify they are in a proper T-3 hour configuration. LH2 prevalves are closed (with recirc pumps active).

3.1.7.3.30.2. Sequence

```
begin sequence;

set commanderrorcheck OFF
{
perform ByteTest4 of <openCMDA>
        verify ByteTest4 = OFF
        else send message to page A
                "LO2 Prevalve" <openCMDA> "is not in T-6 hour configuration."
                and set commanderrorcheck ON

perform ByteTest4 of <openCMDB>
        verify ByteTest4 = OFF
        else send message to page A
                "LO2 Prevalve" <openCMDB> "is not in T-6 hour configuration."
                and set commanderrorcheck ON

perform ByteTest4 of <openCMDC>
        verify ByteTest4 = OFF
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
        else send message to page A
                "LO2 Prevalve" <openCMDC> "is not in T-6 hour configuration."
                and set commanderrorcheck ON

perform ByteTest4 of <openCMDD>
        verify ByteTest4 = OFF
        else send message to page A
                "LO2 Prevalve" <openCMDD> "is not in T-6 hour configuration."
                and set commanderrorcheck ON
}
{
perform ByteTest4 of <closeCMDA>
        verify ByteTest4 = ON
        else send message to page A
                "LO2 Prevalve" <closeCMDA> "is not in T-6 hour configuration."
                and set commanderrorcheck ON

perform ByteTest4 of <closeCMDB>
        verify ByteTest4 = ON
        else send message to page A
                "LO2 Prevalve" <closeCMDB> "is not in T-6 hour configuration."
                and set commanderrorcheck ON

perform ByteTest4 of <closeCMDC>
        verify ByteTest4 = ON
        else send message to page A
                "LO2 Prevalve" <closeCMDC> "is not in T-6 hour configuration."
                and set commanderrorcheck ON

perform ByteTest4 of <closeCMDD>
        verify ByteTest4 = ON
        else send message to page A
                "LO2 Prevalve" <closeCMDD> "is not in T-6 hour configuration."
                and set commanderrorcheck ON

{
perform ByteTest4 of <mainstageCMDA>
        verify ByteTest4 = OFF
        else send message to page A
                "LO2 Prevalve" <mainstageCMDA> "is not in T-6 hour configuration."
                and set commanderrorcheck ON

perform ByteTest4 of <mainstageCMDB>
        verify ByteTest4 = OFF
        else send message to page A
                "LO2 Prevalve" <mainstageCMDB> "is not in T-6 hour configuration."
                and set commanderrorcheck ON

perform ByteTest4 of <mainstageCMDC>
        verify ByteTest4 = OFF
        else send message to page A
                "LO2 Prevalve" <mainstageCMDC> "is not in T-6 hour configuration."
                and set commanderrorcheck ON
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
}

return commanderrorcheck

end sequence;
```

3.1.7.3.31. activateConstraints()

3.1.7.3.31.1. Descriptive Text

This method will activate constraint monitoring of all unbypassed measurement of this valve.

3.1.7.3.31.2. Sequence

```
activateConstraints()
        {
        if(<openPosInd>.Byp == OFF)
                {
                activate constraint monitoring for <openPosInd>
                }

        if(<openPwr1>.Byp == OFF)
                {
                activate constraint monitoring for <openPwr1>
                }
        if(<openPwr2>.Byp == OFF)
                {
                activate constraint monitoring for <openPwr2>
                }
        if(<openRPC1>.Byp == OFF)
                {
                activate constraint monitoring for <openRPC1>
                }
        if(<openRPC2>.Byp == OFF)
                {
                activate constraint monitoring for <openRPC2>
                }
        if(<openRPC3>.Byp == OFF)
                {
                activate constraint monitoring for <openRPC3>
                }
        if(<openRPC4>.Byp == OFF)
                {
                activate constraint monitoring for <openRPC4>
                }
        if(<closePwr1>.Byp == OFF)
                {
                activate constraint monitoring for <closePwr1>
                }
        if(<closePwr2>.Byp == OFF)
                {
                activate constraint monitoring for <closePwr2>
                }
        if(<closePosInd>.Byp == OFF)
                {
                activate constraint monitoring for <closePosInd>
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
        }
    if(<closeRPC1>.Byp == OFF)
        {
        activate constraint monitoring for <closeRPC1>
        }
    if(<closeRPC2>.Byp == OFF)
        {
        activate constraint monitoring for <closeRPC2>
        }
    if(<closeRPC3>.Byp == OFF)
        {
        activate constraint monitoring for <closeRPC3>
        }
    if(<closeRPC4>.Byp == OFF)
        {
        activate constraint monitoring for <closeRPC4>
        }
```

3.1.7.3.32. inhibitConstraints()

3.1.7.3.32.1. Descriptive Text

This method will inhibit constraint monitoring of all measurements related to this valve.

3.1.7.3.32.2. Sequence

```
inhibitConstraints()
    {
    inhibit constraint monitoring of the following:
    <openPosInd>
    <openRPC1>
    <openRPC2>
    <openRPC3>
    <openRPC4>
    <openPwr1>
    <openPwr2>
    <closePosInd>
    <closePwr1>
    <closePwr2>
    <closeRPC1>
    <closeRPC2>
    <closeRPC3>
    <closeRPC4>
    }
```

3.1.7.3.33. setOpenConstraints()

3.1.7.3.33.1. Descriptive Text

This method changes the constraint criteria of the measurements associated with this valve to those expected with the valve in the open position.

3.1.7.3.33.2. Sequence

```
    void setOpenConstraints()
    {
        CHANGE <openPwr1> CONSTRAINT CRITERIA TO ON;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

CHANGE <openPwr2> CONSTRAINT CRITERIA TO ON;

CHANGE <openPosInd> CONSTRAINT CRITERIA TO ON;

CHANGE <openRPC1> CONSTRAINT CRITERIA TO ON;

CHANGE <openRPC2> CONSTRAINT CRITERIA TO ON;

CHANGE <openRPC3> CONSTRAINT CRITERIA TO ON;

CHANGE <openRPC4> CONSTRAINT CRITERIA TO ON;

CHANGE <closePwr1> CONSTRAINT CRITERIA TO OFF;

CHANGE <closePwr2> CONSTRAINT CRITERIA TO OFF;

CHANGE <closePosInd> CONSTRAINT CRITERIA TO OFF;

CHANGE <closeRPC1> CONSTRAINT CRITERIA TO OFF;

CHANGE <closeRPC2> CONSTRAINT CRITERIA TO OFF;

CHANGE <closeRPC3> CONSTRAINT CRITERIA TO OFF;

CHANGE <closeRPC4> CONSTRAINT CRITERIA TO OFF;

}

3.1.7.3.34. setOpenPosConstraints()

3.1.7.3.34.1. Descriptive Text

This method changes the constraint criteria of the measurements associated with this valve to those expected with the valve in the open position with no power (pneumatics) applied.

3.1.7.3.34.2. Sequence

setOpenPosConstraints()

{

CHANGE <openPwr1> CONSTRAINT CRITERIA TO OFF;

CHANGE <openPwr2> CONSTRAINT CRITERIA TO OFF;

CHANGE <openPosInd> CONSTRAINT CRITERIA TO ON;

CHANGE <openRPC1> CONSTRAINT CRITERIA TO OFF;

CHANGE <openRPC2> CONSTRAINT CRITERIA TO OFF;

CHANGE <openRPC3> CONSTRAINT CRITERIA TO OFF;

CHANGE <openRPC4> CONSTRAINT CRITERIA TO OFF;

CHANGE <closePwr1> CONSTRAINT CRITERIA TO OFF;

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

CHANGE <closePwr2> CONSTRAINT CRITERIA TO OFF;

CHANGE <closePosInd> CONSTRAINT CRITERIA TO OFF;

CHANGE <closeRPC1> CONSTRAINT CRITERIA TO OFF;

CHANGE <closeRPC2> CONSTRAINT CRITERIA TO OFF;

CHANGE <closeRPC3> CONSTRAINT CRITERIA TO OFF;

CHANGE <closeRPC4> CONSTRAINT CRITERIA TO OFF;

}

3.1.7.3.35. setClosedConstraints()

3.1.7.3.35.1. Descriptive Text

This method changes the constraint criteria of the measurements associated with this valve to those expected with the valve in the closed position.

3.1.7.3.35.2. Sequence

void setCloseConstraints()
{

CHANGE <openPwr1> CONSTRAINT CRITERIA TO OFF;

CHANGE <openPwr2> CONSTRAINT CRITERIA TO OFF;

CHANGE <openPosInd> CONSTRAINT CRITERIA TO OFF;

CHANGE <openRPC1> CONSTRAINT CRITERIA TO OFF;

CHANGE <openRPC2> CONSTRAINT CRITERIA TO OFF;

CHANGE <openRPC3> CONSTRAINT CRITERIA TO OFF;

CHANGE <openRPC4> CONSTRAINT CRITERIA TO OFF;

CHANGE <closePwr1> CONSTRAINT CRITERIA TO ON;

CHANGE <closePwr2> CONSTRAINT CRITERIA TO ON;

CHANGE <closePosInd> CONSTRAINT CRITERIA TO ON;

CHANGE <closeRPC1> CONSTRAINT CRITERIA TO ON;

CHANGE <closeRPC2> CONSTRAINT CRITERIA TO ON;

CHANGE <closeRPC3> CONSTRAINT CRITERIA TO ON;

CHANGE <closeRPC4> CONSTRAINT CRITERIA TO ON;

}

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.7.3.36. setClosedPosConstraints()

3.1.7.3.36.1. Description

This method changes the constraint criteria of the measurements associated with this valve to those expected with the valve in the closed position with no power (pneumatics) applied.

3.1.7.3.36.2. Sequence

setClosedPosConstraints()

{

CHANGE <openPwr1> CONSTRAINT CRITERIA TO OFF;

CHANGE <openPwr2> CONSTRAINT CRITERIA TO OFF;

CHANGE <openPosInd> CONSTRAINT CRITERIA TO OFF;

CHANGE <openRPC1> CONSTRAINT CRITERIA TO OFF;

CHANGE <openRPC2> CONSTRAINT CRITERIA TO OFF;

CHANGE <openRPC3> CONSTRAINT CRITERIA TO OFF;

CHANGE <openRPC4> CONSTRAINT CRITERIA TO OFF;

CHANGE <closePwr1> CONSTRAINT CRITERIA TO OFF;

CHANGE <closePwr2> CONSTRAINT CRITERIA TO OFF;

CHANGE <closePosInd> CONSTRAINT CRITERIA TO ON;

CHANGE <closeRPC1> CONSTRAINT CRITERIA TO OFF;

CHANGE <closeRPC2> CONSTRAINT CRITERIA TO OFF;

CHANGE <closeRPC3> CONSTRAINT CRITERIA TO OFF;

CHANGE <closeRPC4> CONSTRAINT CRITERIA TO OFF;

}

3.1.7.4. User Interfaces

3.1.7.5. Safety Requirements

3.1.8. MPS_Valve_ORB_LH2HiPt_Bleed

3.1.8.1. Description

The MPS LH2 high point bleed valve (MPS_PV22) is a normally closed monostable ball valve. The valve is spring loaded to the closed position, requiring MPS 750 pneumatics to override spring force to cycle the valve to the open position. The valve has two open commands, 1 close and 1 open position indicators, and 1 open power indicator. Functionally, the LH2 high point bleed valve is open during vehicle tanking (to prevent the formation of a hydrogen gas bubble at the top of the feedline) and closed during ascent to provide redundancy (with T-0 QD PD17) to loss of propellent overboard during ascent. The valve inherits from the MPS_ORB_Valve class.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.8.1.1. Instance Index

Instance	Display User	Other Users
MPS_PV22	MPS	

3.1.8.1.2. Class Diagram

<insert class diagram here>

3.1.8.1.3. Attributes and Aggregations

Attribute	Data Type	Initial Value	Publish / Persistence	Client Access	Description
openCmdA	DS_ON_OFF				
openCmdB	DS_ON_OFF				
openPositionInd	DM_ON_OFF				
closePositionInd	DM_ON_OFF				
openPower	DM_ON_OFF				

3.1.8.1.4. Methods Index

Operation	Description
open()	This method performs necessary functions to cycle valve to the open position.
close()	This method performs necessary functions to cycle valve to the closed position.
energizeSolenoid()	This method performs necessary functions to energize the opening solenoid with MPS pneumatics secured. If pneumatics is active, the method invokes the open method after prompting operator that a valve cycle will result.
deenergizeSolenoid()	This method performs necessary functions to deenergize the opening solenoid with MPS pneumatics secured. If pneumatics is active, the method invokes the close method after prompting operator that a valve cycle will result.
determineCmdState()	This method will determine the enumerated command state of the valve based on the opening solenoid power indication and will return either: ENERGIZED DEENERGIZED ERROR
getCmdState()	This method will return the enumerated command state of the valve

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

determineIndState()	This method will determine the position state of the valve based on open/close indicators and will return either: OPEN CLOSED TRANSITION ERROR
getIndState()	This method will return valve indicator Enumerated state.
determineState()//Phase 3	This method will determine the overall Enumerated State based on the values of indState and CmdState. The compiled state will be used on phase 3 integrated displays. Possible values are: CLOSED CLOSED_ERROR OPEN OPEN_ERROR TRANSITION ERROR
getState()//Phase 3	This method returns the overall enumerated state of the valve.
verifyCommandsOff()	This method performs a byte test 4 of valve open commands to verify no commands are issued during system securing operations.
autoCycle()	This method will perform valve open/close autocycle operations.
inhibitConstraints()	This method will inhibit constraint notification during valve cycle operations.
setOpenConstraints()	This method will establish constraint criteria to that expected of an open valve.
setCloseConstraints()	This method will establish constraint criteria to that expected of a closed valve.
activateConstraints()	This method will activate constraints for either the close or open positions when in standby mode (uncommanded present state).
verifyTminus6BT4()//Phase 3	PAD method required to support load. Verifies command status for 6 hour configuration.
verifyTminus3BT4()//Phase 3	PAD method required to support load. Verifies command status for 3 hour configuration.

3.1.8.1.5. Associations

Association	Type	Class	Description
MPS_Helium_Subsystem			MPS 750 pneumatic Status.
MPS_Subsystem			Cryo flag status.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

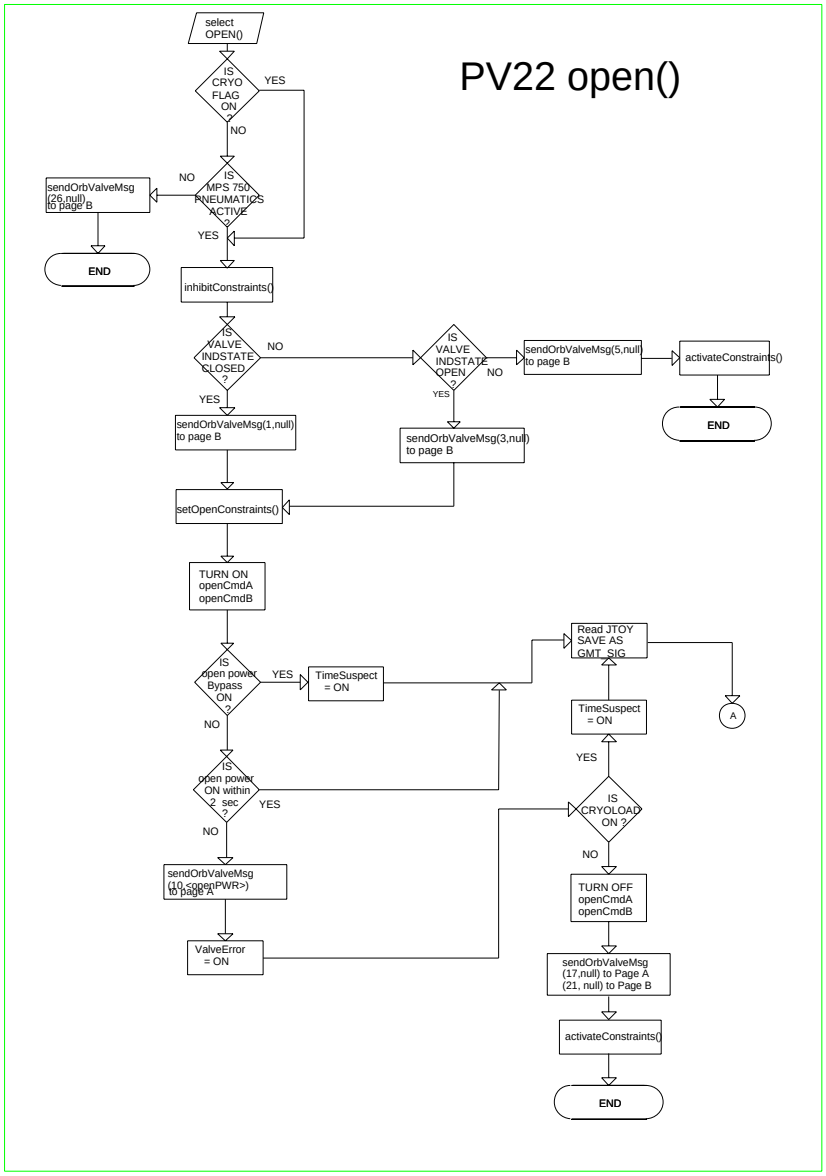
3.1.8.2. Operations

3.1.8.2.1. open()

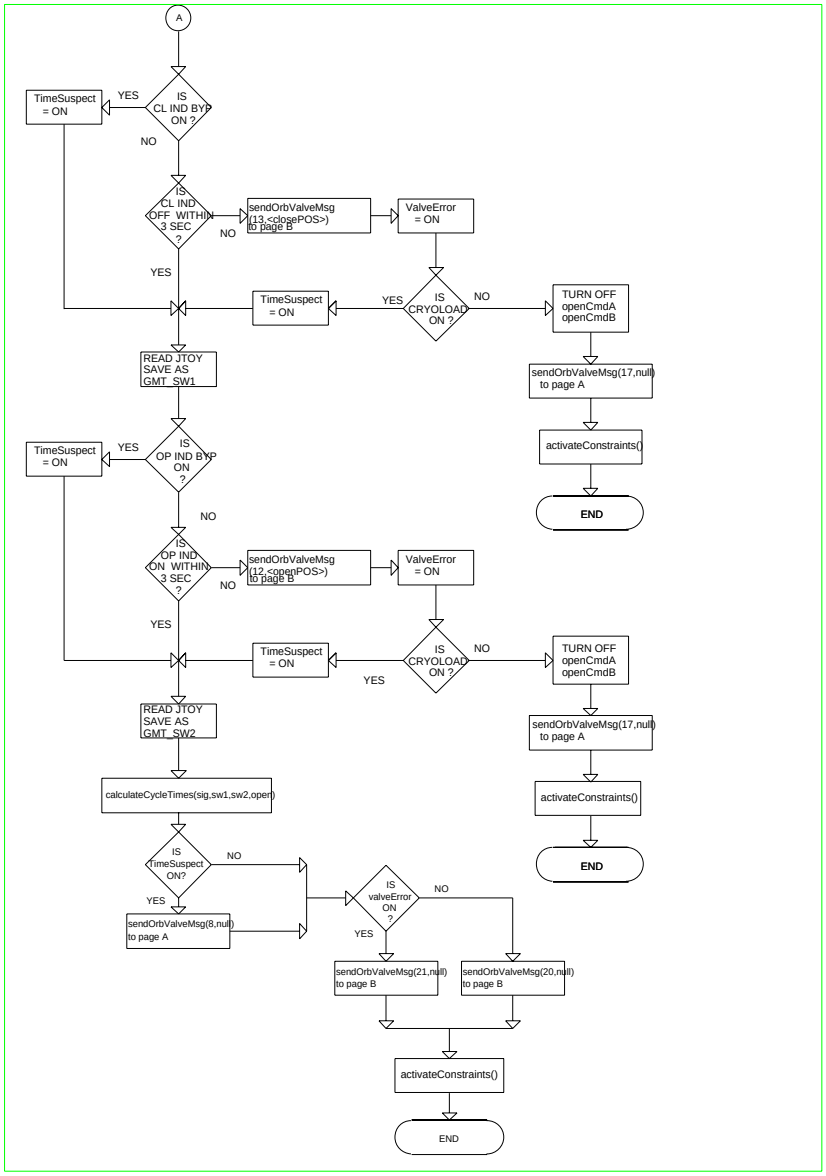
3.1.8.2.1.1. Descriptive Text

This method performs cycling LH2 HighPoint Bleed Valve to the open position. The flow chart provides the necessary logic for valve cycle, including checks for the proper response of power indicators, and proper response of position indicators. The valve cycles in a fail safe mode and 'wimps-out' (all commands removed) at the first indication of hardware failure, unless the orbiter is in cryogenic load. If in cryo load, the valve cycles in a 'damn-the-torpedos' mode and completes all steps necessary to achieve the valve cycle, regardless of failures which are detected during the cycle. Valve signal to switch and switch to switch timing is also calculated. ValveError and TimeSuspect are initialized to OFF and Valve cycle times are set to 0 prior to starting valve cycle methods. If ValveError gets set ON during method run, cycle time FD data health should be set to invalid to aid in identifying suspect cycle data in trend analysis.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

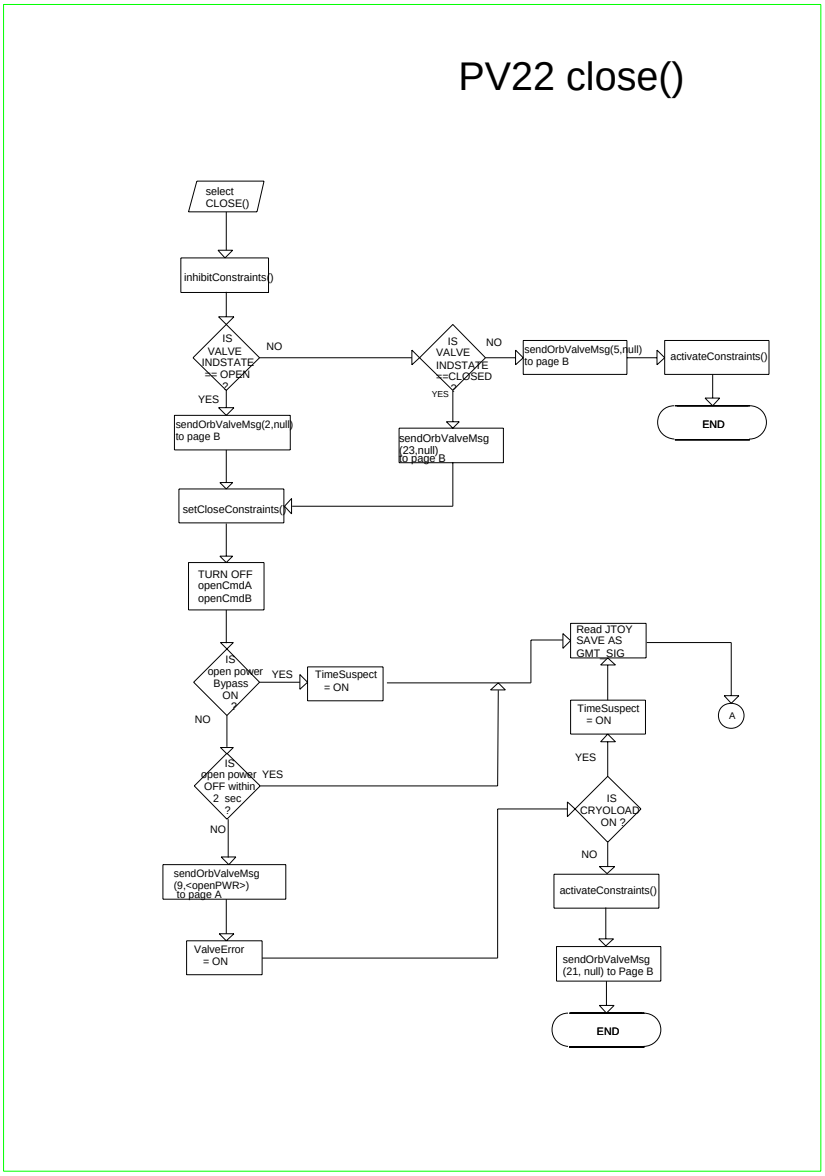


3.1.8.2.2. close()

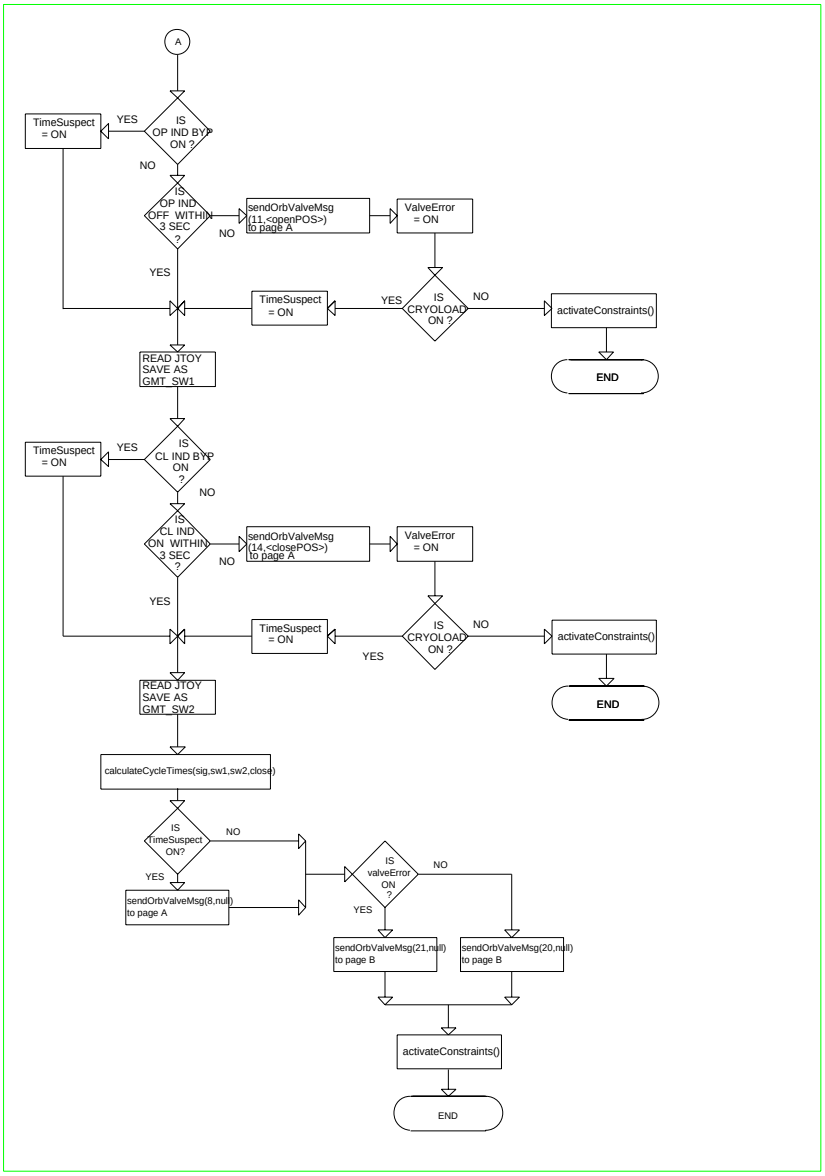
3.1.8.2.2.1. Descriptive Text

This method performs cycling LH2 HighPoint Bleed Valve to the closed position. The flow chart provides the necessary logic for valve cycle, including checks for the proper response of power indicators, and proper response of position indicators. The valve cycles in a fail safe mode and 'wimps-out' (all commands removed) at the first indication of hardware failure, unless the orbiter is in cryogenic load. If in cryo load, the valve cycles in a 'damn-the-torpedos' mode and completes all steps necessary to achieve the valve cycle, regardless of failures which are detected during the cycle. Valve signal to switch and switch to switch timing is also calculated. ValveError and TimeSuspect are initialized to OFF and Valve cycle times are set to 0 prior to starting valve cycle methods. If ValveError gets set ON during method run, cycle time FD data health should be set to invalid to aid in identifying suspect cycle data in trend analysis.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

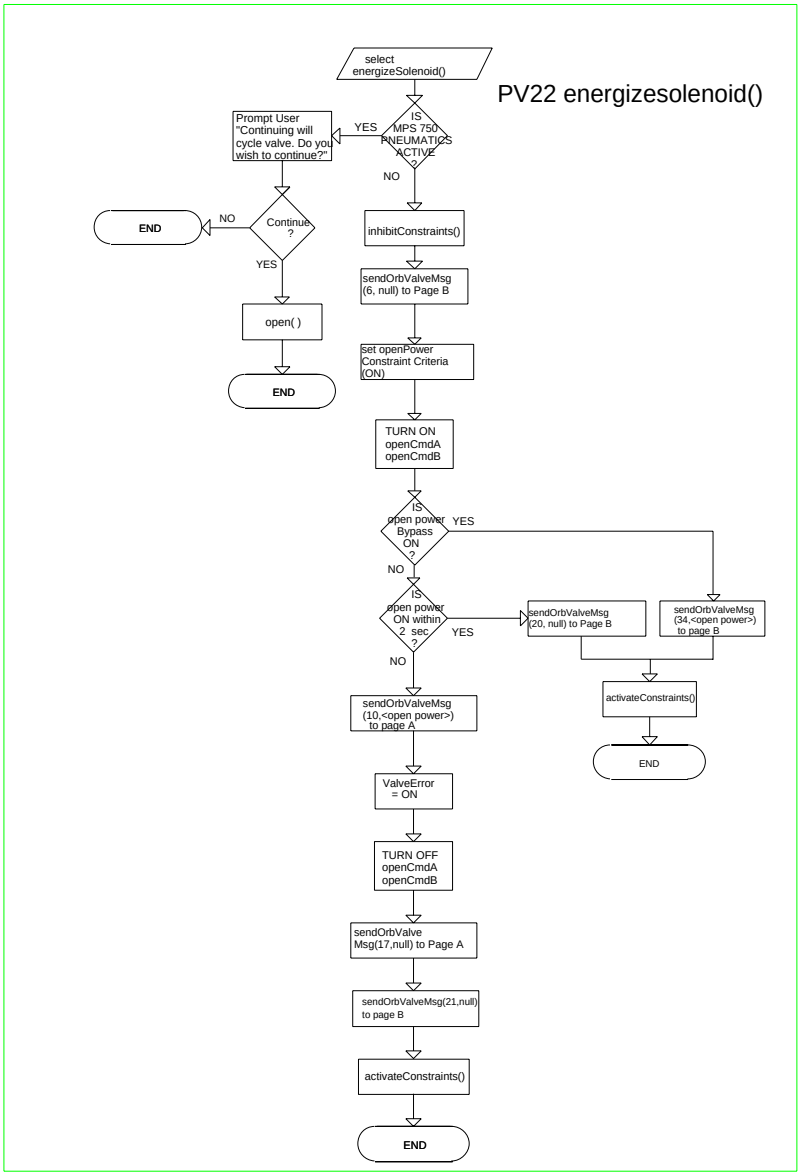


3.1.8.2.3. energizeSolenoid()

3.1.8.2.3.1. Descriptive Text

This method performs applies power to the open solenoid. If MPS 750 pneumatics is active the open() method is called. The flow chart provides the necessary logic for the method, including checks for the proper response of power indicators. The valve cycles in a fail safe mode and 'wimps-out' (all commands removed) at the first indication of hardware failure. ValveError is to be set to OFF when method is initialized.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

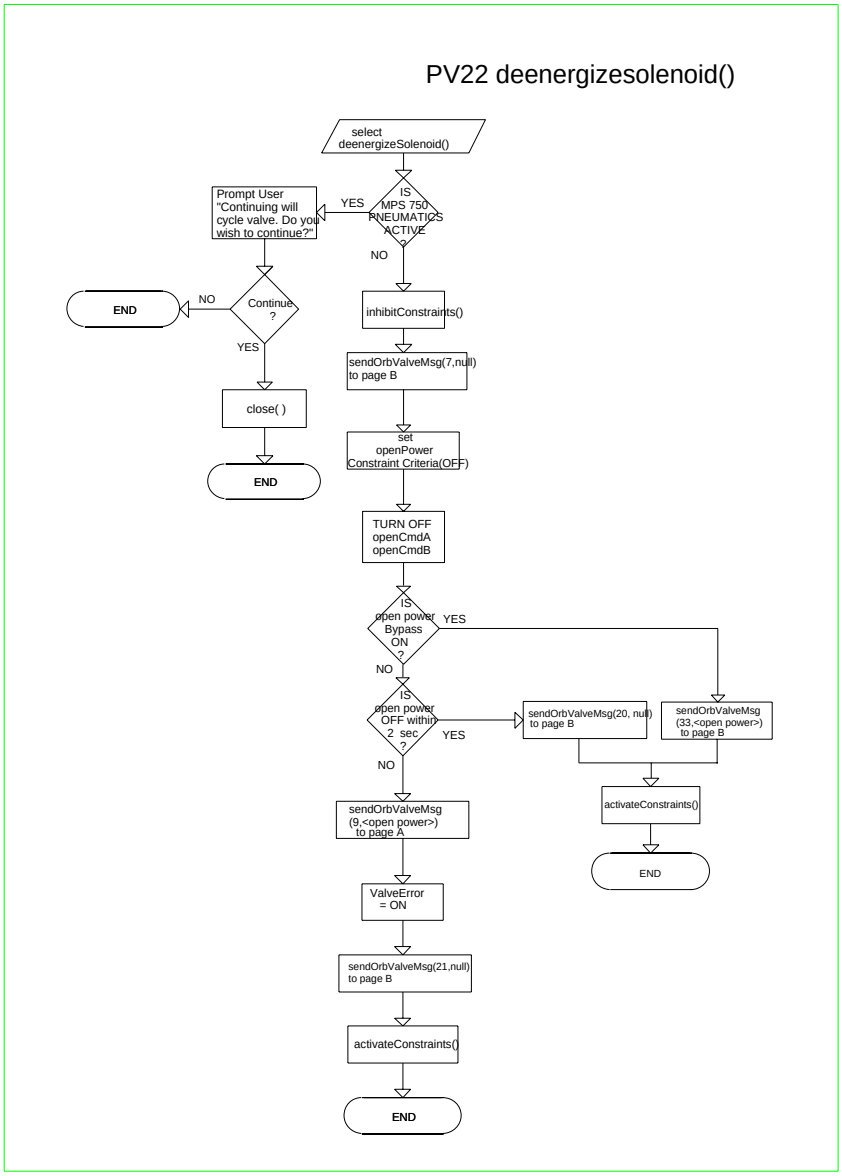


3.1.8.2.4. deenergizeSolenoid()

3.1.8.2.4.1. Descriptive Text

This method performs removes power from the open solenoid. If MPS 750 pneumatics is active the close() method is called. The flow chart provides the necessary logic for the method, including checks for the proper response of power indicators. The valve cycles in a fail safe mode and 'wimps-out' (all commands removed) at the first indication of hardware failure. ValveError is to be set to OFF when method is initialized.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



3.1.8.2.5. determineIndState()

3.1.8.2.5.1. Descriptive Text

This method will determine the valve's position based on its position indicators and associated indicator bypasses. The return value will be an enumerated type with possible values of OPEN, CLOSED, TRANSITION, and ERROR.

3.1.8.2.5.2. Effectivity

Operates on change data.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.8.2.5.3. Sequence

Indicator State				
open pos ind	**open pos byp**	**close pos ind**	**close pos byp**	**state**
OFF	OFF	ON	OFF	CLOSED
OFF	OFF	----	ON	CLOSED
----	ON	ON	OFF	CLOSED
ON	OFF	OFF	OFF	OPEN
----	ON	OFF	OFF	OPEN
ON	OFF	----	ON	OPEN
OFF	OFF	OFF	OFF	TRANSITION
----	ON	----	ON	ERROR
ON	OFF	ON	OFF	ERROR

3.1.8.2.6. determineCmdState()

3.1.8.2.6.1. Descriptive Text

This method will determine the valve's commanded state based on the valve open power FD. The return value will be an enumerated type with possible values of ENERGIZED, DEENERGIZED and ERROR.

3.1.8.2.6.2. Effectivity

Operates on change data.

3.1.8.2.6.3. Sequence

Case:

<openPwr> ON : cmdState = ENERGIZED;

<openPwr> OFF : cmdState = DEENERGIZED;

<openPwr>.Bypass ON : cmdState = ERROR;

End Case:

3.1.8.2.7. determineState()//Phase 3 Method

3.1.8.2.7.1. Descriptive Text

3.1.8.2.7.2. Sequence

case:

(<cmdState> == DEENERGIZED AND <indState> == closed) : state= CLOSED;

(<cmdState> == DEENERGIZED AND <indState> == open) : state= OPEN_ERROR;

(<cmdState> == DEENERGIZED AND <indState> == TRANSITION) : state = TRANSITION;

(<cmdState> == DEENERGIZED AND <indState> == error) : state = ERROR;

(<cmdState> == ENERGIZED AND <indState> == closed) : state= CLOSED_ERROR;

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

(<cmdState> == ENERGIZED AND <indState> == open) : state= OPEN;

(<cmdState> == ENERGIZED AND <indState> == TRANSITION) : state = TRANSITION;

(<cmdState> == ENERGIZED AND <indState> == error) : state = ERROR;

End Case;

3.1.8.2.8. getIndState()

3.1.8.2.8.1. Descriptive Text

This method will return the Enumerated Indicator State of the valve_

3.1.8.2.8.2. Sequence

```
getIndState()
{
	return indState;
}
```

3.1.8.2.9. getCmdState()

3.1.8.2.9.1. Descriptive Text

This method will return the Enumerated Command State of the valve.

3.1.8.2.9.2. Sequence

```
getCmdState()
{
	return cmdState;
}
```

3.1.8.2.10. getState()//Phase 3 Method

3.1.8.2.10.1. Descriptive Text

This method will return the overall Enumerated State of the valve.

3.1.8.2.10.2. Sequence

```
getState()
{
	return state;
}
```

3.1.8.2.11. autoCycle()

3.1.8.2.11.1. Descriptive Text

The following will be called by the Panel Class autoValveCheckout method to calculate and record the valve cycle times.

3.1.8.2.11.2. Sequence

Set autoValveCheckoutError = OFF;

open();

if valveCycleError is ON, Set autoValveCheckoutError = ON;

close();

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

if valveCycleError is ON, Set autoValveCheckoutError = ON;

Return autoValveCheckoutError;

3.1.8.2.12. activateConstraints()

3.1.8.2.12.1. Descriptive Text

This method will activate constraint monitoring of all unbypassed measurement of this valve.

3.1.8.2.12.2. Sequence

```
activateConstraints()
        {
        if(<openInd>.Byp == OFF)
                {
                activate constraint monitoring for <openInd>
                }
        if(<openPwr>.Byp == OFF)
                {
                activate constraint monitoring for <openPwr>
                }
        if(<closeInd>.Byp == OFF)
                {
                activate constraint monitoring for <closeInd>
                }
        }
```

3.1.8.2.13. inhibitConstraints()

3.1.8.2.13.1. Descriptive Text

This method will inhibit constraint monitoring of all measurements related to this valve.

3.1.8.2.13.2. Sequence

```
inhibitConstraints()
        {
        inhibit constraint monitoring of the following:
        <openInd>
        <closeInd>
        <openPwr>
        }
```

3.1.8.2.14. setOpenConstraints()

3.1.8.2.14.1. Descriptive Text

This method changes the constraint criteria of the measurements associated with this valve to those expected with the valve in the open position.

3.1.8.2.14.2. Sequence

```
        void setOpenConstraints()
        {
                CHANGE <openInd> CONSTRAINT CRITERIA TO ON;

               CHANGE <openPwr>  CONSTRAINT CRITERIA TO ON;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

CHANGE <closeInd> CONSTRAINT CRITERIA TO OFF;

}

3.1.8.2.15. setClosedConstraints()

3.1.8.2.15.1. Descriptive Text

This method changes the constraint criteria of the measurements associated with this valve to those expected with the valve in the closed position.

3.1.8.2.15.2. Sequence

void setCloseConstraints()
{
CHANGE <openInd> CONSTRAINT CRITERIA TO OFF;

CHANGE <openPwr> CONSTRAINT CRITERIA TO OFF;

CHANGE <closeInd> CONSTRAINT CRITERIA TO ON;

}

3.1.8.2.16. verifyCommandsOff()

3.1.8.2.16.1. Descriptive Text

This function will verify that both the open and close commands are OFF

3.1.8.2.16.2. Sequence

verifyCommandsOff()
{
open command status 1 = byteTest4 <Open Command A>
open command status 2 = byteTest4 <Open Command B>
{
verify open command status 1 == OFF
else send message to page A: valveID "open command A" <openCMDA>" is ON, should be OFF"
}
{
verify open command status 2 == OFF
else send message to page A: valveID "open command B" <openCMDB>" is ON, should be OFF"
}
{
verify open command status 1 && 2 == OFF
Return TRUE
else Return FALSE
}

3.1.8.2.17. verifyTminus3BT4()//Phase 3 Method

3.1.8.2.17.1. Description

This method verifies commands are ON during cryo load.

3.1.8.2.17.2. Sequence

verifyCommandsOff()
{
open command status 1 = byteTest4 <Open Command A>

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
open command status 2 = byteTest4 <Open Command B>
        {
        verify  open command status 1 == ON
        else send message to page A: valveID "open command A" <openCMDA>" is OFF, should be ON"
        }
        {
        verify  open command status 2 == OFF
        else send message to page A: valveID "open command B" <openCMDB>" is OFF, should be ON"
        }
{
verify  open command status 1 && 2 == OFF
    Return TRUE
else Return FALSE
}
```

3.1.8.2.18. verifyTminus6BT4()//Phase 3 Method

3.1.8.2.18.1. Description

This method verifies PV22 commands are OFF prior to cryogenic load.

3.1.8.2.18.2. Sequence

```
verifyCommandsOff()
{
open command status 1 = byteTest4 <Open Command A>
open command status 2 = byteTest4 <Open Command B>
        {
        verify  open command status 1 == OFF
        else send message to page A: valveID "open command A" <openCMDA>" is ON, should be OFF"
        }
        {
        verify  open command status 2 == OFF
        else send message to page A: valveID "open command B" <openCMDB>" is ON, should be OFF"
        }
{
verify  open command status 1 && 2 == OFF
    Return TRUE
else Return FALSE
}
```

3.1.9. MPS_Valve_ORB_Topping

3.1.9.1. Description

The MPS LH2 topping valve (MPS_PV13) is a normally closed monostable ball valve. The valve is spring loaded to the closed position, requiring MPS 750 pneumatics to override spring force to cycle the valve to the open position. The valve has one open command, 1 close and 1 open position indicators, and 1 open power indicator. Functionally the valve provides redundancy to loss of propellant from the LH2 recirc manifold/topping line overboard (with the LH2 outboard fill and drain valve) during ascent, and the capability to isolate the topping line and SSME bleed systems during tanking. The valve inherits from the MPS_ORB_Valve class.

3.1.9.1.1. Instance Index

Instance	Display User	Other Users

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

MPS_PV13	MPS	

3.1.9.1.2. Class Diagram

<insert class diagram here>

3.1.9.1.3. Attributes and Aggregations

Attribute	Data Type	Initial Value	Publish / Persistence	Client Access	Description
openCmd	DS_ON_OFF				
openPosInd	DM_ON_OFF				
closePosInd	DM_ON_OFF				
openPower	DM_ON_OFF				

3.1.9.1.4. Operations Index

Operation	Description
open()	This method performs the operations necessary to open the valve.
close()	This mthod performs the operations necessary to close the valve.
energizeOpenSolenoid()	This method performs necessary functions to energize the opening solenoid with MPS pneumatics secured. If pneumatics is active, the method invokes the open method after prompting operator of valve cycle.
deenergizeOpenSolenoid()	This method performs necessary functions to deenergize the opening solenoid with MPS pneumatics secured. If pneumatics is active, the method invokes the close method after prompting operator of valve cycle.
determineIndState()	This method looks at the position indicators of the valve and returns an enumerated state based on a logic table of either: OPEN CLOSED TRANSITION ERROR
determineCmdState()	This method looks at the open power indicator of the valve and returns an enumerated state based on a logic table of either: ENERGIZED DEENERGIZED ERROR
determineState()//Phase 3	This method looks at the cmdState and indState pseudo-FDs of the valve and returns an enumerated state based on a logic table. Possible values are: OPEN OPEN_ERROR

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

	CLOSED CLOSED_ERROR TRANSITION ERROR
getIndState()	This method returns the enumerated indicator state.
getCmdState()	This method returns the enumerated command state.
getState()//Phase 3	This method returns the enumerated overall state.
inhibitConstraints()	This method inhibits constraint monitoring of all FD measurements associated with this valve.
activateConstraints()	This method activates constraint monitoring of all unbypassed FD measurements associated with this valve.
autoCycle()	This method runs the open and close methods of the valve and returns either failure or success.
setOpenConstraints()	This method establishes the constraint criteria expected for the valve in the open position.
setCloseConstraints()	This method establishes the constraint criteria expected for the valve in the closed position.
verifyCommandsOff()	This method verifies open command to the valve is off by performing a byteTest4.
verifyTminus6BT4()//	Pad method required to support load. Verifies command status for 6 hour configuration.
verifyTminus3BT4()//	Pad method required to support load. Verifies command status for 3 hour configuration.

3.1.9.1.5. Associations

Association	Type	Class	Description
MPS_Helium_Subsystem			MPS 750 pneumatic status.
MPS_Subsystem			Cryo flag status.

3.1.9.2. Operations

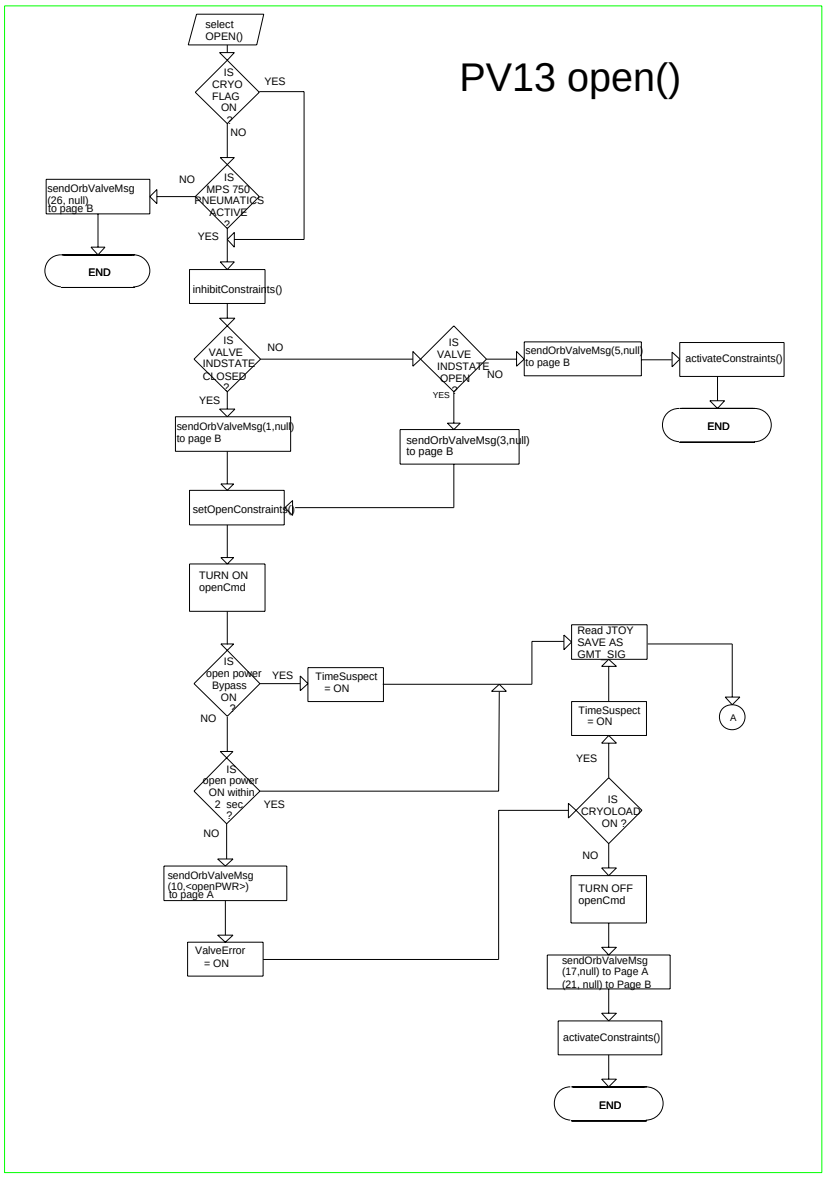
3.1.9.2.1. open()

3.1.9.2.1.1. Descriptive Text

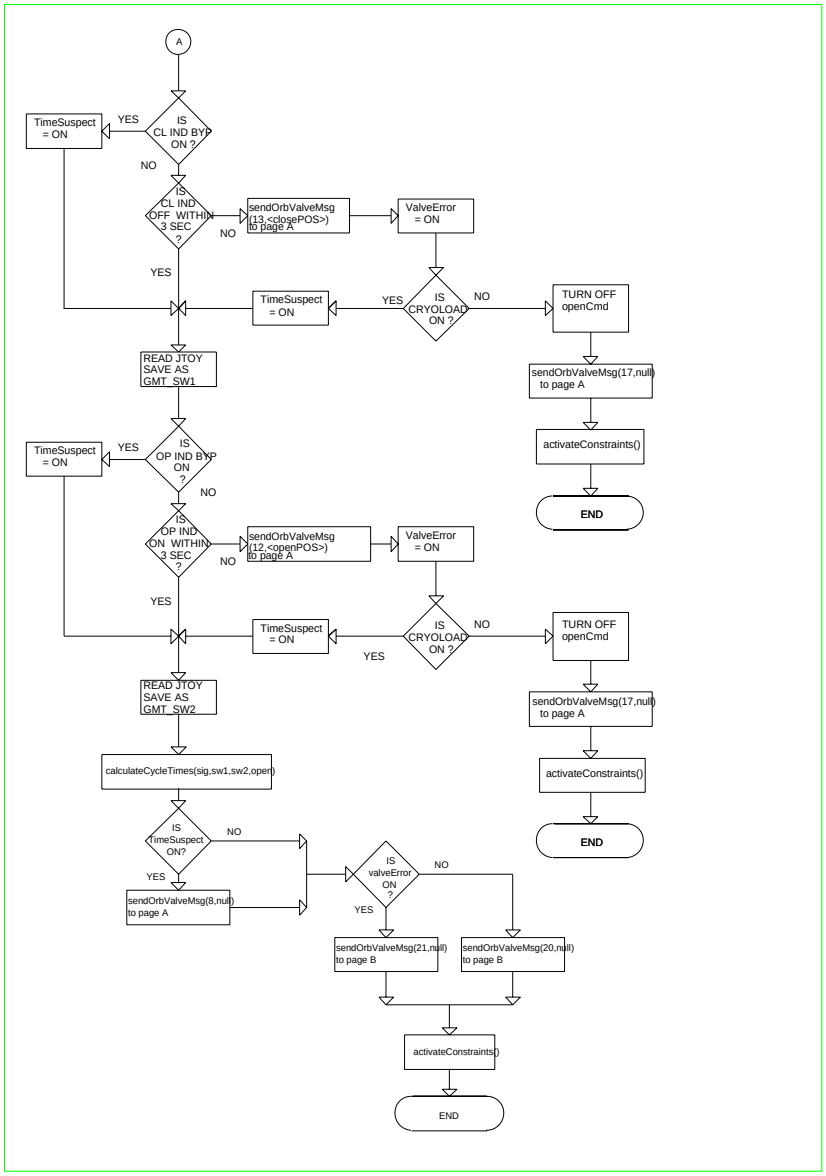
This method performs cycling LH2 Topping Valve to the open position. The flow chart provides the necessary logic for valve cycle, including checks for the proper response of open power indicator, and proper response of position

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

indicators. The valve cycles in a fail safe mode and 'wimps-out' (all commands removed) at the first indication of hardware failure, unless the orbiter is in cryogenic load. If in cryo load, the valve cycles in a 'damn-the-torpedos' mode and completes all steps necessary to achieve the valve cycle, regardless of failures which are detected during the cycle. Valve signal to switch and switch timing is also called by the cycle method. ValveError and TimeSuspect are initialized to OFF and Valve cycle times are set to 0 prior to starting valve cycle methods. If ValveError gets set ON during method run, cycle time FD data health should be set to invalid to aid in identifying suspect cycle data in trend analysis.



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

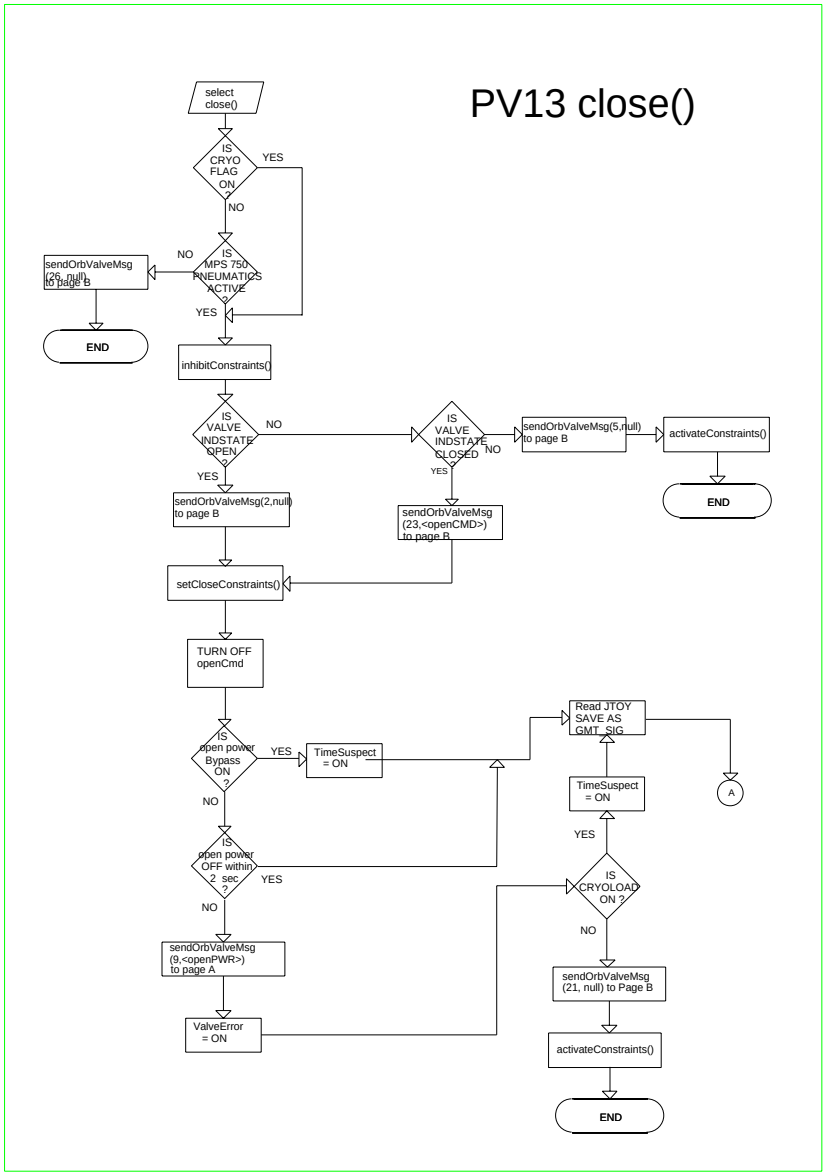


3.1.9.2.2. close()

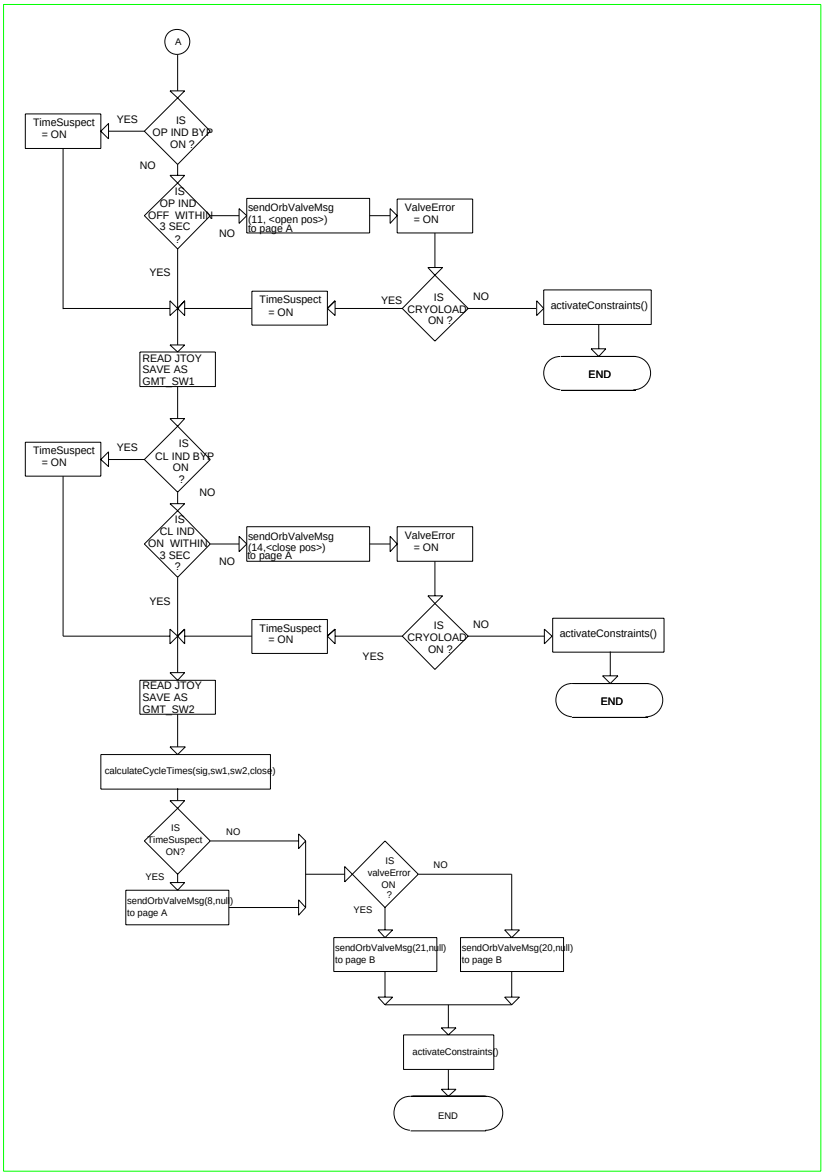
3.1.9.2.2.1. Descriptive Text

This method performs cycling LH2 Topping Valve to the closed position. The flow chart provides the necessary logic for valve cycle, including checks for the proper response of power indicators, and proper response of position indicators. The valve cycles in a fail safe mode and 'wimps-out' (all commands removed) at the first indication of hardware failure, unless the orbiter is in cryogenic load. If in cryo load, the valve cycles in a 'damn-the-torpedos' mode and completes all steps necessary to achieve the valve cycle, regardless of failures which are detected during the cycle. Valve signal to switch and switch timing is also called by the cycyle method. ValveError and TimeSuspect are initialized to OFF and Valve cycle times are set to 0 prior to starting valve cycle methods. If ValveError gets set ON during method run, cycle time FD data health should be set to invalid to aid in identifying suspect cycle data in trend analysis.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

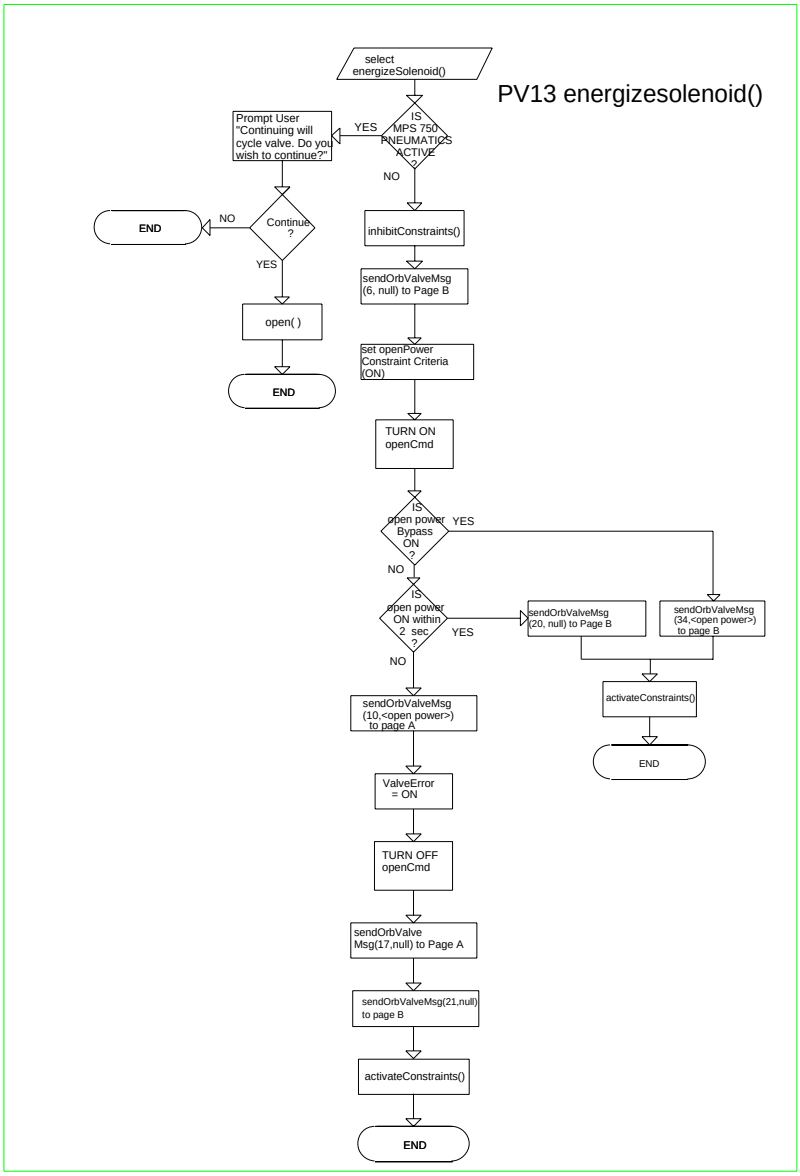


3.1.9.2.3. energizeOpenSolenoid()

3.1.9.2.3.1. Descriptive Text

This method performs applies power to the open solenoid. If MPS 750 pneumatics is active and the operator responds via prompt to continue with cycling the valve, the open() method is called. The flow chart provides the necessary logic for the method, including checks for the proper response of the open power indicator. The valve cycles in a fail safe mode and 'wimps-out' (all commands removed) at the first indication of hardware failure

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

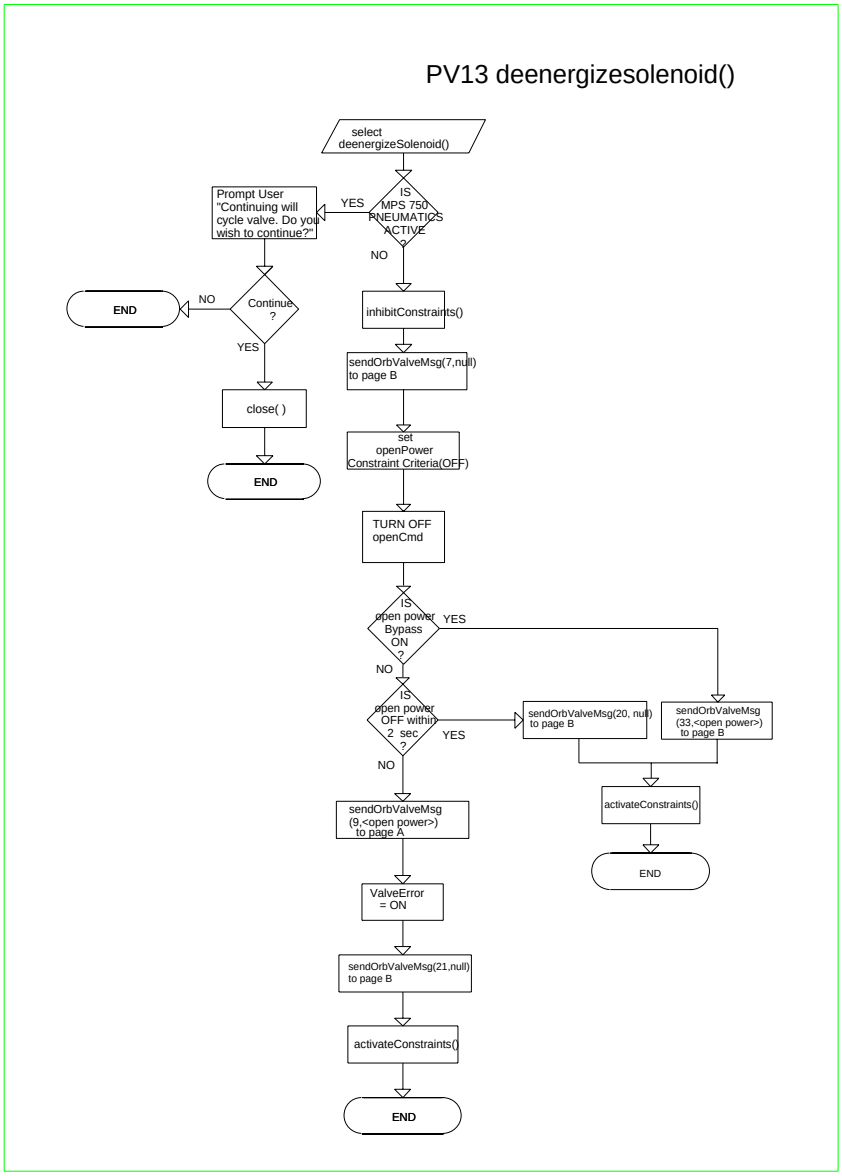


3.1.9.2.4. deenergizeOpenSolenoid()

3.1.9.2.4.1. Descriptive Text

This method performs removes power from the open solenoid. If MPS 750 pneumatics is active and the operator responds via prompt to continue with cycling the valve, the close() method is called. The flow chart provides the necessary logic for the method, including checks for the proper response of the power indicator. The valve cycles in a fail safe mode and 'wimps-out' (all commands removed) at the first indication of hardware failure

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



3.1.9.2.5. determineIndState()

3.1.9.2.5.1. Descriptive Text

This method will determine the valve's position based on its position indicators and associated indicator bypasses. The return value will be an enumerated type with possible values of OPEN, CLOSED, TRANSITION, and ERROR.

3.1.9.2.5.2. Effectivity

Operates on change data.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

Indicator State				
open pos ind	open pos byp	close pos ind	close pos byp	state
OFF	OFF	ON	OFF	CLOSED
OFF	OFF	----	ON	CLOSED
----	ON	ON	OFF	CLOSED
ON	OFF	OFF	OFF	OPEN
----	ON	OFF	OFF	OPEN
ON	OFF	----	ON	OPEN
OFF	OFF	OFF	OFF	TRANSITION
----	ON	----	ON	ERROR
ON	OFF	ON	OFF	ERROR

3.1.9.2.6. determineCmdState()

3.1.9.2.6.1. Descriptive Text

This method will determine the valve's commanded state based on the valve open power FD. The return value will be an enumerated type with possible values of ENERGIZED, DEENERGIZED and ERROR.

3.1.9.2.6.2. Effectivity

Operates on change data.

3.1.9.2.6.3. Sequence

Case:

<openPwr> ON : cmdState = ENERGIZED;

<openPwr> OFF : cmdState = DEENERGIZED;

<openPwr>.Bypass ON : cmdState = ERROR;

End Case:

3.1.9.2.7. determineState()//Phase 3

3.1.9.2.7.1. Descriptive Text

3.1.9.2.7.2. Sequence

case:

(<cmdState> == DEENERGIZED && <indState> == closed) : state= CLOSED;

(<cmdState> == DEENERGIZED && <indState> == open) : state= OPEN_ERROR;

(<cmdState> == DEENERGIZED && <indState> == TRANSITION) : state = TRANSITION;

(<cmdState> == DEENERGIZED && <indState> == error) : state = ERROR;

(<cmdState> == ENERGIZED && <indState> == closed) : state= CLOSED_ERROR;

(<cmdState> == ENERGIZED && <indState> == open) : state= OPEN;

(<cmdState> == ENERGIZED && <indState> == TRANSITION) : state = TRANSITION;

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

(<cmdState> == ENERGIZED && <indState> == error) : state = ERROR;

DEFAULT CASE : state = ERROR;

End Case;

3.1.9.2.8. getIndState()

3.1.9.2.8.1. Descriptive Text

This method will return the Enumerated Indicator State of the valve_

3.1.9.2.8.2. Sequence

```
getIndState()
{
        return indState;
}
```

3.1.9.2.9. getCmdState()

3.1.9.2.9.1. Descriptive Text

This method will return the Enumerated Command State of the valve.

3.1.9.2.9.2. Sequence

```
getCmdState()
{
        return cmdState;
}
```

3.1.9.2.10. getState()//Phase 3

3.1.9.2.10.1. Descriptive Text

This method will return the overall Enumerated State of the valve.

3.1.9.2.10.2. Sequence

```
getState()
{
        return state;
}
```

3.1.9.2.11. autoCycle()

3.1.9.2.11.1. Descriptive Text

The following will be called by the MPS_LH2_Subsystem Class autoValveCheckout method to calculate and record the valve cycle times.

3.1.9.2.11.2. Sequence

Set autoValveCheckoutError = OFF;

open();

if valveCycleError is ON, Set autoValveCheckoutError = ON;

close();

if valveCycleError is ON, Set autoValveCheckoutError = ON;

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

Return autoValveCheckoutError;

3.1.9.2.12. activateConstraints()

3.1.9.2.12.1. Descriptive Text

This method will activate constraint monitoring of all unbypassed measurement of this valve.

3.1.9.2.12.2. Sequence

```
activateConstraints()
        {
        if(<openInd>.Byp == OFF)
                {
                activate constraint monitoring for <OP IND>
                }
        if(<openPwr>.Byp == OFF)
                {
                activate constraint monitoring for <OP PWR>
                }
        if(<closeInd>.Byp == OFF)
                {
                activate constraint monitoring for <CL IND>
                }
        }
```

3.1.9.2.13. inhibitConstraints()

3.1.9.2.13.1. Descriptive Text

This method will inhibit constraint monitoring of all measurements related to this valve.

3.1.9.2.13.2. Sequence

```
inhibitConstraints()
        {
        inhibit constraint monitoring of the following:
        <OP IND>
        <OP PWR>
        <CL IND>
        }
```

3.1.9.2.14. setOpenConstraints()

3.1.9.2.14.1. Descriptive Text

This method changes the constraint criteria of the measurements associated with this valve to those expected with the valve in the open position.

3.1.9.2.14.2. Sequence

```
        void setOpenConstraints()
        {
                CHANGE <OP IND> CONSTRAINT CRITERIA TO ON;

                CHANGE <OP PWR>  CONSTRAINT CRITERIA TO ON;

                CHANGE <CL IND> CONSTRAINT CRITERIA TO OFF;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

}

3.1.9.2.15. setClosedConstraints()

3.1.9.2.15.1. Descriptive Text

This method changes the constraint criteria of the measurements associated with this valve to those expected with the valve in the closed position.

3.1.9.2.15.2. Sequence

```
void setCloseConstraints()
{
        CHANGE <OP IND> CONSTRAINT CRITERIA TO OFF;

        CHANGE <OP PWR>  CONSTRAINT CRITERIA TO OFF;

        CHANGE <CL IND> CONSTRAINT CRITERIA TO ON;

}
```

3.1.9.2.16. verifyCommandsOff()

3.1.9.2.16.1. Descriptive Text

This function will verify that the open command is OFF.

3.1.9.2.16.2. Sequence

```
verifyCommandsOff()
{
open command status = byteTest4 <Open Command>
        {
        verify  open command status == OFF
        else print message to page A :   "Byte Test 4 shows", valveID, "open command" <openCMD> "to be ON"
        and Return TRUE

        Else return FALSE
        }
}
```

3.1.9.2.17. verifyTminus6BT4()//Phase 3 Method

3.1.9.2.17.1. Description

This method supports T-6 hour byte test 4 of MPS_PV13 open command. PV13 is closed (uncommanded) during this check at this point in cryo load.

```
verifyCommandsOff()
{
open command status = byteTest4 <Open Command>
        {
        verify  open command status == OFF
        else print message to page A :   "Byte Test 4 shows", valveID, "open command" <openCMD> "to be ON"
        and Return TRUE

        Else return FALSE
        }
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.1.9.2.18. verifyTminus3BT4()//Phase 3 Method

3.1.9.2.18.1. Description

This method supports T-3 hour byte test 4 of MPS_PV13 open command. PV13 is open (commanded open) during this check at this point in cryo load.

```
verifyCommandsOff()
{
open command status = byteTest4 <Open Command>
        {
        verify  open command status == ON
        else print message to page A :   "Byte Test 4 shows", valveID, "open command" <openCMD> "to be OFF"
        and Return TRUE

        Else return FALSE
        }
}
```

3.1.10. MPS_Valve_ORB_RTLS_Dump

3.1.10.1. Description

The MPS LH2 RTLS Dump valves (MPS_PV17, PV18) are normally closed monostable ball valves. The valves are spring loaded to the closed position, requiring MPS 750 pneumatics to override spring force to cycle to the open position. The valves each have three open commands, 1 close and 1 open position indicators, 1 open power indicator, and 2 open RPCs. Functionally, the RTLS valves provide redundancy (with the other in-line RTLS valve) against loss of propellant overboard during ascent and are cycled open by onboard software in a contingency RTLS profile to dump residual LH2 overboard after SSME shutdown and during re-entryt inerting operations. The valve inherits from the MPS_ORB_Valve class.

3.1.10.1.1. Instance Index

Instance	Display User	Other Users
MPS_PV17	MPS	
MPS_PV18	MPS	

3.1.10.1.2. Class Diagram

<insert class diagram here>

3.1.10.1.3. Attributes and Aggregations

Attribute	Data Type	Initial Value	Publish / Persistence	Client Access	Description
openCmdA	DS_ON_OFF			R,W	
openCmdB	DS_ON_OFF			R,W	
openCmdC	DS_ON_OFF			R,W	
openPosInd	DM_ON_OF			R	

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

	F				
closePosInd	DM_ON_OF F			R	
openRPC_A	DM_ON_OF F			R	
openRPC_B	DM_ON_OF F			R	
openPower	DM_ON_OF F			R	

3.1.10.1.4. Associations

Association	Type	Class	Description

3.1.10.1.5. Operations Index

Operation	Description
open()	This method performs the operations necessary to open the valve.
close()	This method performs the operations necessary to close the valve.
energizeOpenSolenoid()	This method performs the operations necessary to energize the opening solenoid with MPS pneumatics secured. If pneumatics is active, the method invokes the open method after prompting operator of valve cycle.
deenergizeOpenSolenoid	This method performs the operations necessary to deenergize the opening solenoid with MPS pneumatics secured. If pneumatics is active, the method invokes the close method after prompting operator of valve cycle.
openOnCheck()	This method verifies that the open power and RPCs are energized within two seconds. If the power and RPCs are not powered within the alloted time then the method sends appropriate messages and sets valveError ON.
openOffCheck()	This method verifies that the open power and RPCs are deenergized within two seconds. If the power and RPCs are still powered after the alloted time then the method sends appropriate messages and sets valveError ON.
determineIndState()	This method looks at the position indicators of the valve and returns an enumerated state based on a logic table.
determineCmdState()	This method looks at the open power indicator of the valve and returns an enumerated state based on a logic table.
determineState()	This method looks at the cmdState and indState pseudo-FDs of the valve and returns an enumerated state based on a logic table.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

getIndState()	returns the enumerated indicator state from the CVT.
getCmdState()	returns the enumerated command state from the CVT.
getState()	returns the enumerated overall state from the CVT.
inhibitConstraints()	This method inhibits constraint monitoring of all values associated with this valve.
activateConstraints()	This method activates constraint monitoring of all unbypassed values associated with this valve.
autoCycle()	This method runs the open and close methods of the valve and returns either failure or success.
setOpenConstraints()	This method establishes the constraint criteria expected for the valve in the open position.
setCloseConstraints()	This method establishes the constraint criteria expected for the valve in the closed position.
verifyCommandsOff()	This method will verify no command issued.
//verifyTminus3BT4()//	This method is required to support load and will verify no commands issued at T-3 hour configuration.
//verifyTminus6BT4()//	This method is required to support load and will verify no commands issued at T-6 hour configuration.

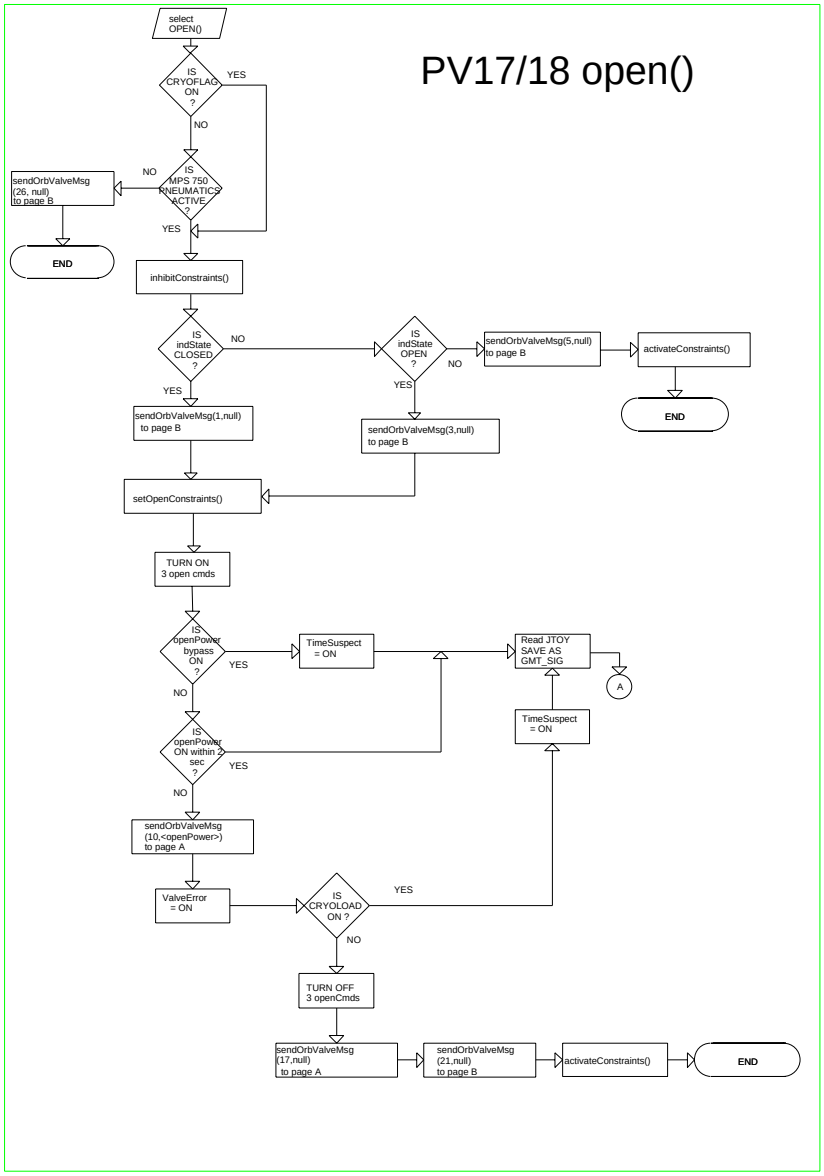
3.1.10.2. Operations

3.1.10.2.1. open()

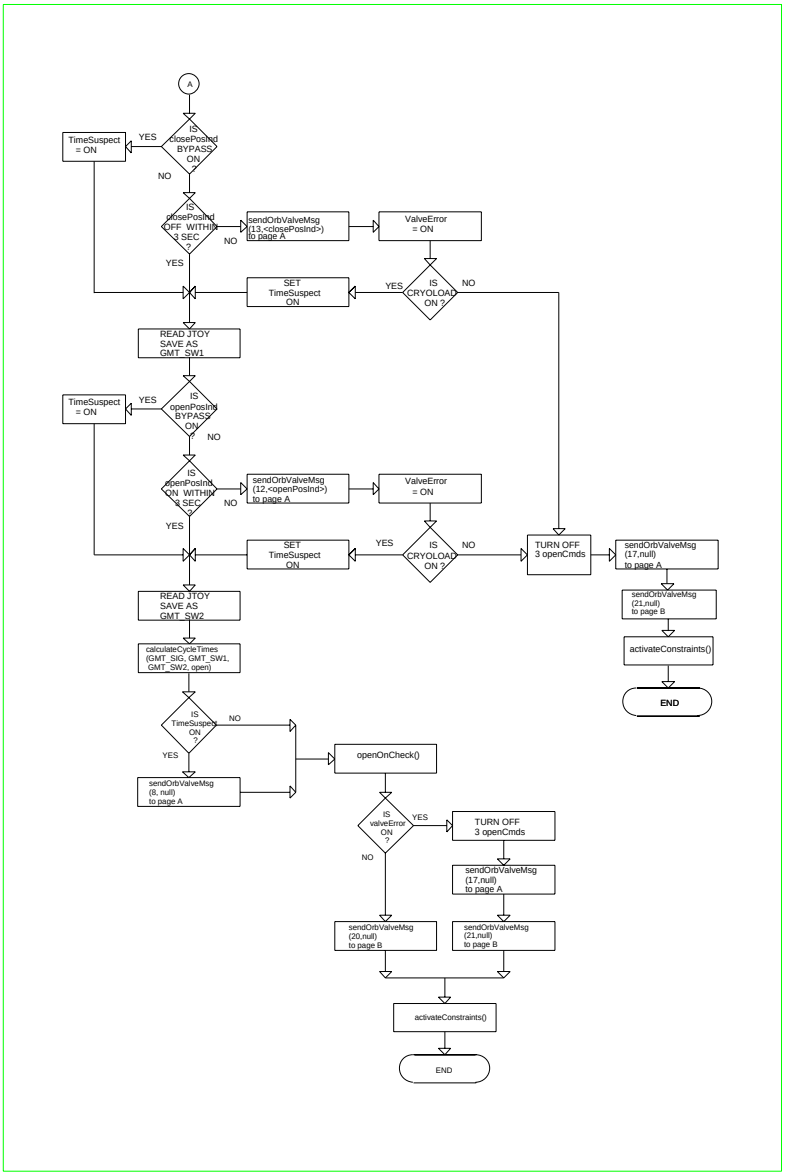
3.1.10.2.1.1. Descriptive Text

This method performs cycling of the RTLS Dump valves to the open position. The flow chart provides the necessary logic for valve cycles, including checks for proper response of power indications, and proper response of position indicators. The valve cycles in a fail safe mode and 'wimps-out' (commands removed) at the first indication of hardware failure, unless the orbiter is in cryogenic load. If in cryo load, the valve cycles in a 'damn-the-torpedos' mode and completes all steps necessary to achieve the valve cycle, regardless of failures which are detected during the cycle. Valve signal to switch and switch timing is also calculated.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

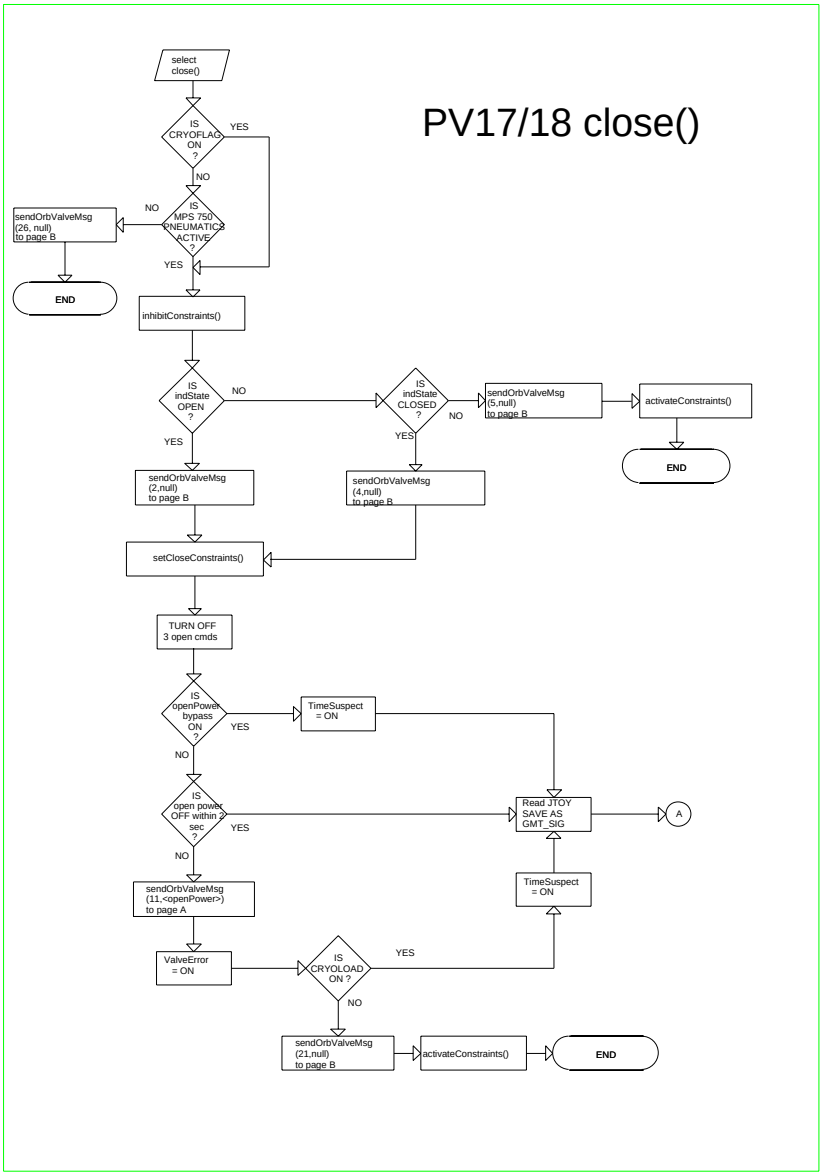


3.1.10.2.2. close()

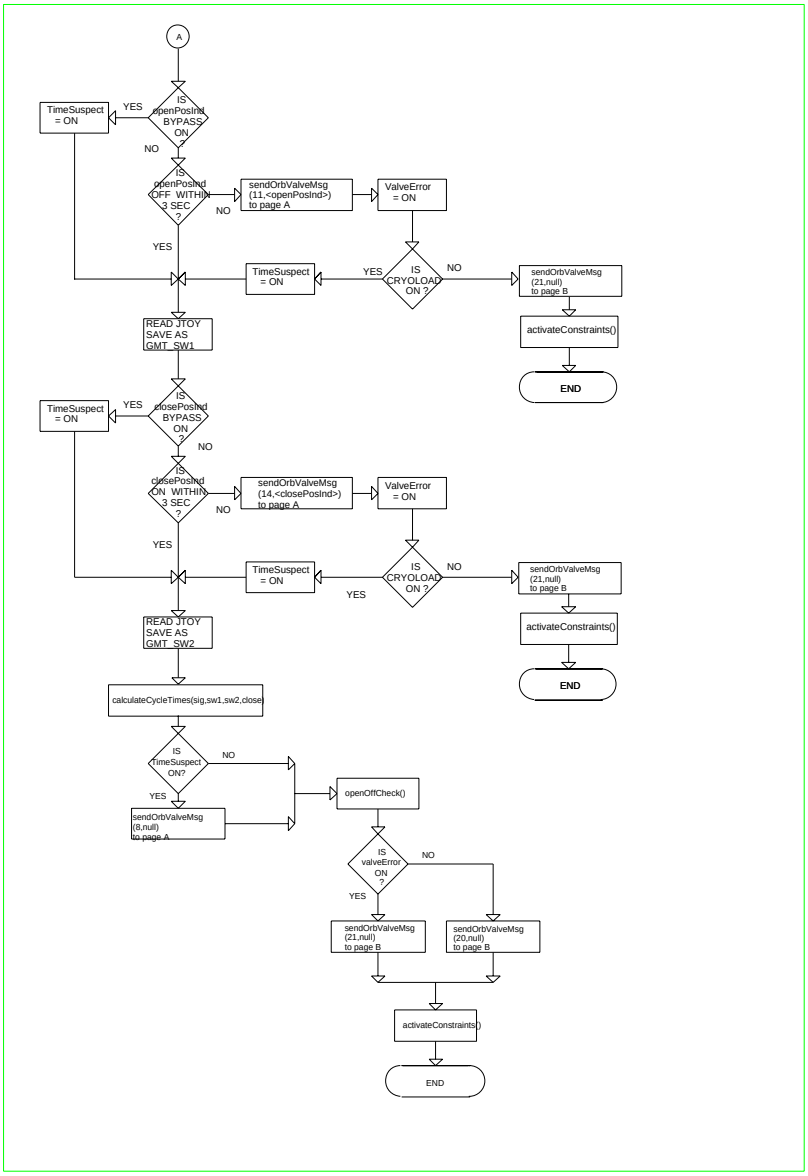
3.1.10.2.2.1. Descriptive Text

This method performs cycling of the RTLS Dump valves to the closed position. The flow chart provides the necessary logic for valve cycles, including checks for proper response of power indications, and proper response of position indicators. The valve cycles in a fail safe mode and 'wimps-out' (commands removed) at the first indication of hardware failure, unless the orbiter is in cryogenic load. If in cryo load, the valve cycles in a 'damn-the-torpedos' mode and completes all steps necessary to achieve the valve cycle, regardless of failures which are detected during the cycle. Valve signal to switch and switch timing is also calculated.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



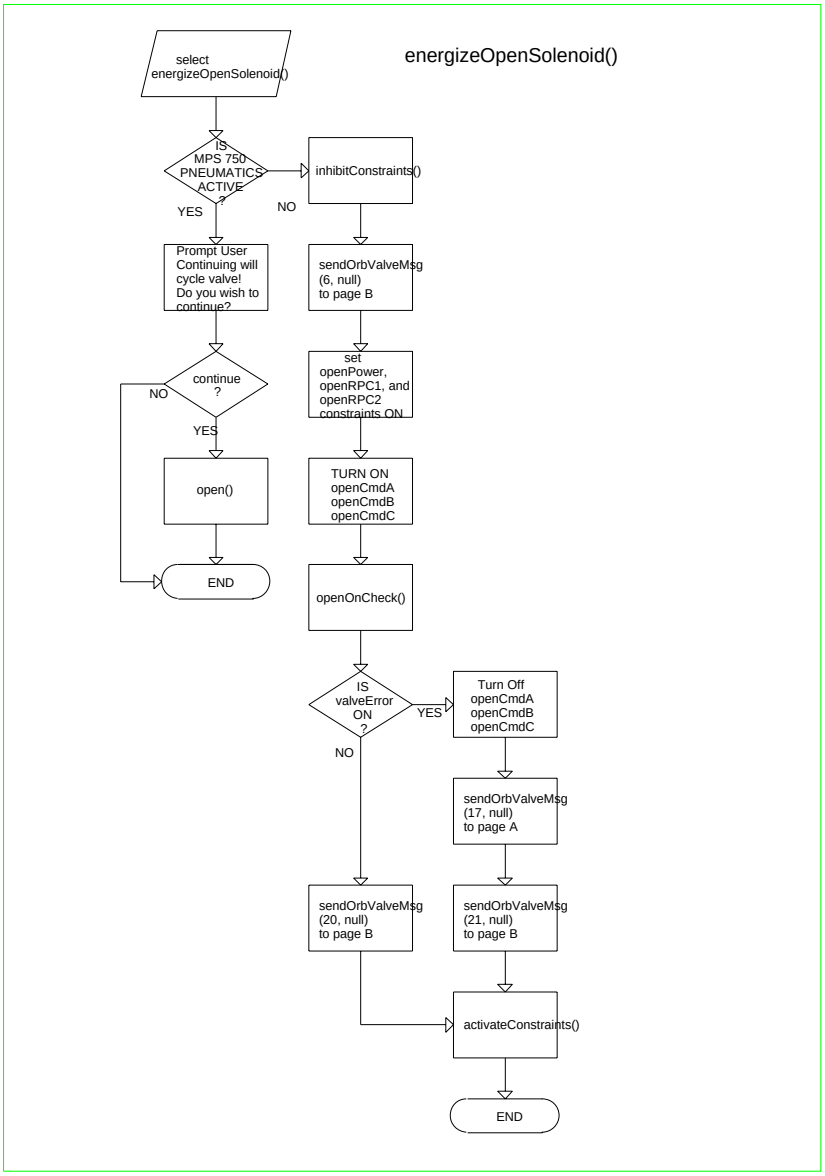
3.1.10.2.3. energizeOpenSolenoid()

3.1.10.2.3.1. Descriptive Text

This method performs applies power to the open solenoid. If MPS 750 pneumatics are active the open() method is called. The flow chart provides the necessary logic for the method, including checks for the proper response of power indicators. The valve cycles in a fail safe mode and 'wimps-out' (all commands removed) at the first indication of hardware failure

3.1.10.2.3.2. Sequence

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



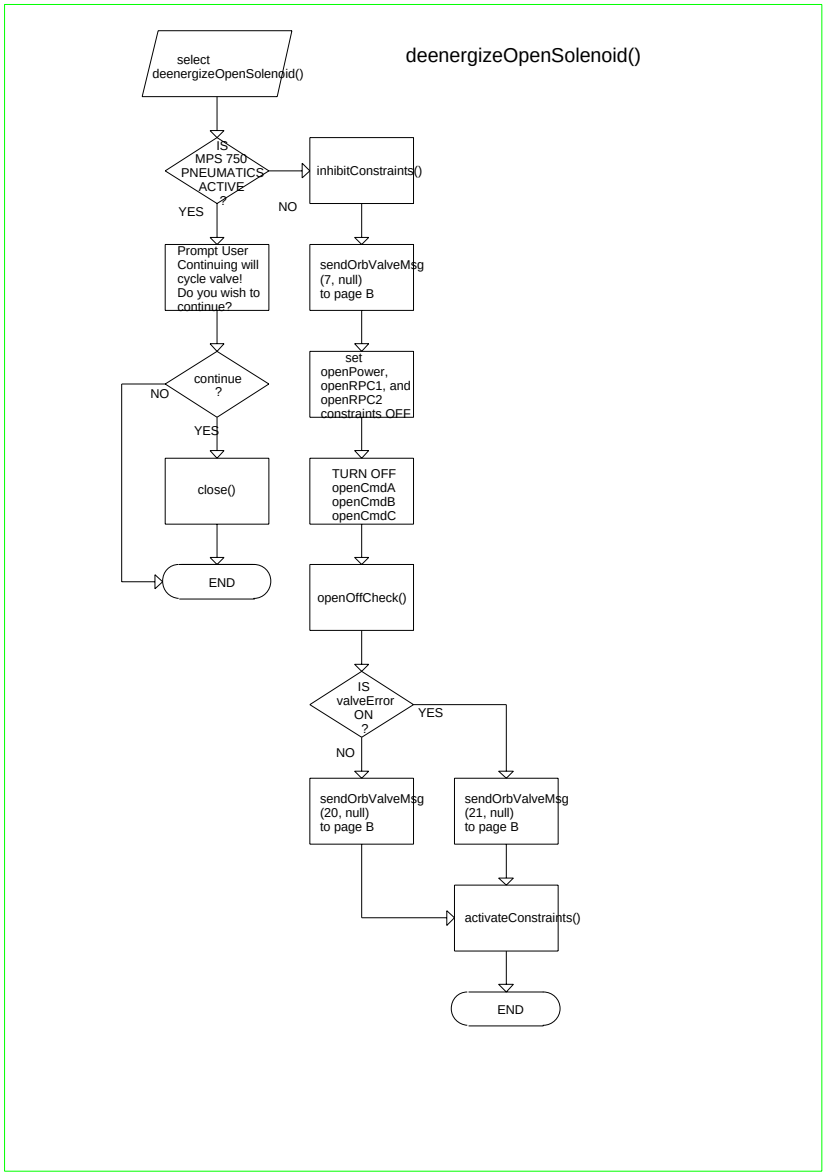
3.1.10.2.4. deenergizeOpenSolenoid()

3.1.10.2.4.1. Descriptive Text

This method performs removes power from the open solenoid. If MPS 750 pneumatics are active the close() method is called. The flow chart provides the necessary logic for the method, including checks for the proper response of power indicators. The valve cycles in a fail safe mode and 'wimps-out' (all commands removed) at the first indication of hardware failure The implementation of the details is left up to the coders and is acceptable as long as the overall logic is maintained. It is expected that cycle timing will be accomplished using the most accurate source. All messages will include a time stamp, and the appropriate FD description.

3.1.10.2.4.2. Sequence

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



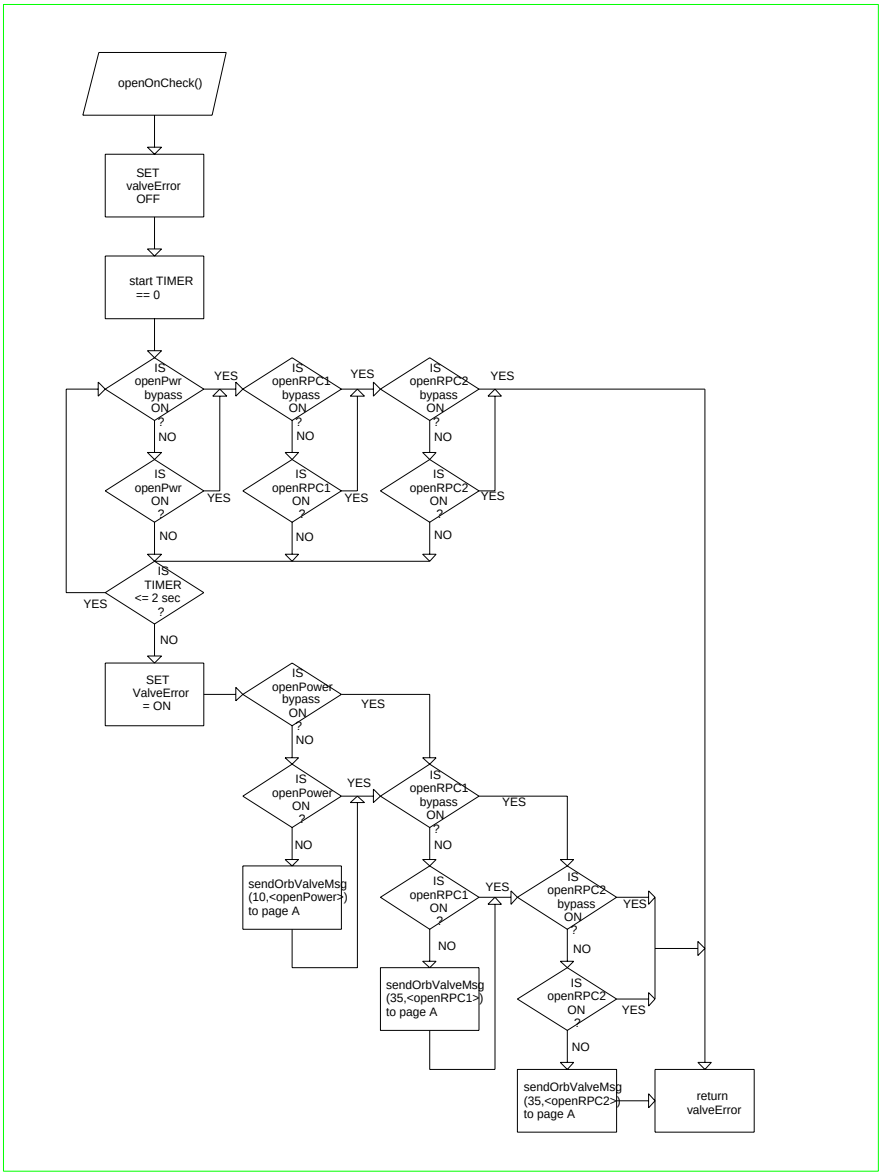
3.1.10.2.5. openOnCheck()

3.1.10.2.5.1. Descriptive Text

This method verifies that power and RPCs are on within 2 seconds. If any of the power or RPC indicators are not on within two seconds then the method prints appropriate messages and sets valveError on.

3.1.10.2.5.2. Sequence

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



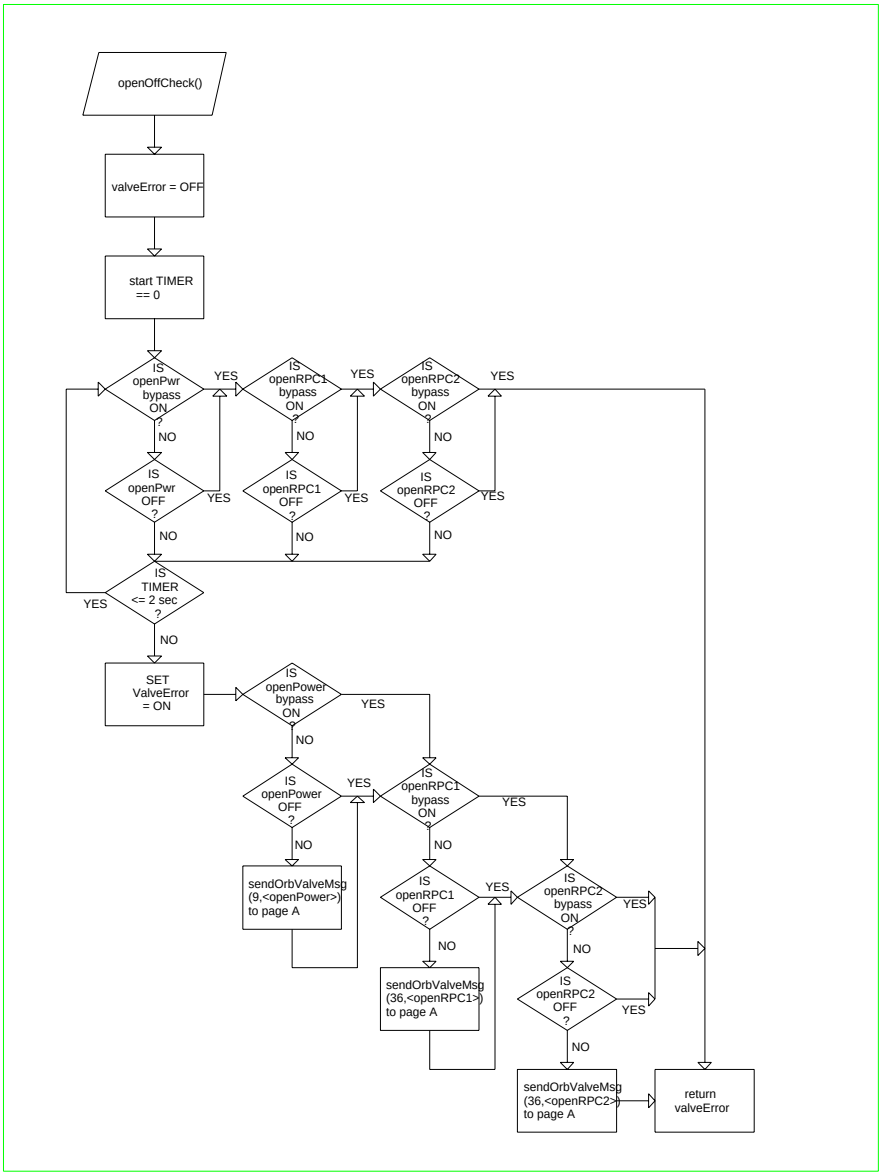
3.1.10.2.6. openOffCheck()

3.1.10.2.6.1. Descriptive Text

This method verifies that power and RPCs are off within 2 seconds. If any of the power or RPC indicators are not off within two seconds then the method prints appropriate messages and sets valveError on.

3.1.10.2.6.2. Sequence

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



3.1.10.2.7. determineIndState()

3.1.10.2.7.1. Descriptive Text

This method will determine the valve's position based on its position indicators and associated indicator bypasses. The return value will be an enumerated type with possible values of OPEN, CLOSED, TRANSITION, and ERROR.

3.1.10.2.7.2. Effectivity

Operates on change data.

3.1.10.2.7.3. Sequence

Case:

<OP POS IND> OFF AND

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

<OP POS IND>.BYP OFF AND
<CL POS IND> OFF AND
<CL POS IND>.BYP OFF : indState = TRANSITION;

<OP POS IND> OFF AND
<OP POS IND>.BYP OFF AND
<CL POS IND> ON AND
<CL POS IND>.BYP OFF : indState = CLOSED;

<OP POS IND>.BYP ON AND
<CL POS IND> ON AND
<CL POS IND>.BYP OFF : indState = CLOSED;

<OP POS IND> OFF AND
<OP POS IND>.BYP OFF AND
<CL POS IND>.BYP ON : indState = CLOSED;

<OP POS IND> ON AND
<OP POS IND>.BYP OFF AND
<CL POS IND> OFF AND
<CL POS IND>.BYP OFF : indState = OPEN;

<OP POS IND>.BYP ON AND
<CL POS IND> OFF AND
<CL POS IND>.BYP OFF : indState = OPEN;

<OP POS IND> ON AND
<OP POS IND>.BYP OFF AND
<CL POS IND>.BYP ON : indState = OPEN;

<OP POS IND> ON AND
<OP POS IND>.BYP OFF AND
<CL POS IND> ON AND
<CL POS IND>.BYP OFF : indState = ERROR;

<OP POS IND>.BYP ON AND
<CL POS IND>.BYP ON : indState = ERROR;

Default: indState = ERROR;

End Case:

3.1.10.2.8. determineCmdState()

3.1.10.2.8.1. Descriptive Text

This method will determine the valve's commanded state based on the valve open power and open RPC A and RPC B indicators. The return value will be an enumerated type with possible values of ENERGIZED, DEENERGIZED and ERROR.

3.1.10.2.8.2. Effectivity

Operates on change.

3.1.10.2.8.3. Sequence

Case:

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

<openPwr> OFF
<openPwr>.BYP OFF
<openRPC_A> OFF
<openRPC_A>.BYP OFF
<openRPC_B> OFF
<openRPC_B>.BYP OFF
<cmdState> = DEENERGIZED;

<openPwr> OFF
<openPwr>.BYP OFF
<openRPC_A> OFF
<openRPC_A>.BYP OFF
<openRPC_B>.BYP ON
<cmdState> = DEENERGIZED;

<openPwr> OFF
<openPwr>.BYP OFF
<openRPC_A>.BYP ON
<openRPC_B> OFF
<openRPC_B>.BYP OFF
<cmdState> = DEENERGIZED;

<openPwr> OFF
<openPwr>.BYP OFF
<openRPC_A>.BYP ON
<openRPC_B>.BYP ON
<cmdState> = DEENERGIZED;

<openPwr>.BYP ON
<openRPC_A> OFF
<openRPC_A>.BYP OFF
<openRPC_B> OFF
<openRPC_B>.BYP OFF
<cmdState> = DEENERGIZED;

<openPwr>.BYP ON
<openRPC_A> OFF
<openRPC_A>.BYP OFF
<openRPC_B>.BYP ON
<cmdState> = DEENERGIZED;

<openPwr>.BYP ON
<openRPC_A>.BYP ON
<openRPC_B> OFF
<openRPC_B>.BYP OFF
<cmdState> = DEENERGIZED;

<openPwr>.BYP ON
<openRPC_A>.BYP ON
<openRPC_B> ON
<openRPC_B>.BYP OFF
<cmdState> = ENERGIZED;

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

<openPwr>.BYP ON
<openRPC_A> ON
<openRPC_A>.BYP OFF
<openRPC_B>.BYP ON
<cmdState> = ENERGIZED;

<openPwr>.BYP ON
<openRPC_A> ON
<openRPC_A>.BYP OFF
<openRPC_B> ON
<openRPC_B>.BYP OFF
<cmdState> = ENERGIZED;

<openPwr> ON
<openPwr>.BYP OFF
<openRPC_A>.BYP ON
<openRPC_B>.BYP ON
<cmdState> = ENERGIZED;

<openPwr> ON
<openPwr>.BYP OFF
<openRPC_A>.BYP ON
<openRPC_B> ON
<openRPC_B>.BYP OFF
<cmdState> = ENERGIZED;

<openPwr> ON
<openPwr>.BYP OFF
<openRPC_A> ON
<openRPC_A>.BYP OFF
<openRPC_B>.BYP ON
<cmdState> = ENERGIZED;

<openPwr> ON
<openPwr>.BYP OFF
<openRPC_A> ON
<openRPC_A>.BYP OFF
<openRPC_B> ON
<openRPC_B>.BYP OFF
<cmdState> = ENERGIZED;

DEFAULT CASE ELSE ERROR;

End Case:

3.1.10.2.9. determineState()

3.1.10.2.9.1. Descriptive Text

3.1.10.2.9.2. Sequence

case:

(<cmdState> == DEENERGIZED AND <indState> == closed) : state=closed;

(<cmdState> == DEENERGIZED AND <indState> == open) : state=openError;

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

(<cmdState> == DEENERGIZED AND <indState> == TRANSITION) : state = TRANSITION;

(<cmdState> == DEENERGIZED AND <indState> == error) : state = error;

(<cmdState> == ENERGIZED AND <indState> == closed) : state=closedError;

(<cmdState> == ENERGIZED AND <indState> == open) : state=open;

(<cmdState> == ENERGIZED AND <indState> == TRANSITION) : state = TRANSITION;

(<cmdState> == ENERGIZED AND <indState> == error) : state = error;

End Case;

3.1.10.2.10. getIndState()

3.1.10.2.10.1. Descriptive Text

This method will return the Enumerated Indicator State of the valve_

3.1.10.2.10.2. Sequence

```
getIndState()
{
	return indState;
}
```

3.1.10.2.11. getCmdState()

3.1.10.2.11.1. Descriptive Text

This method will return the Enumerated Command State of the valve.

3.1.10.2.11.2. Sequence

```
getCmdState()
{
	return cmdState;
}
```

3.1.10.2.12. getState()

3.1.10.2.12.1. Descriptive Text

This method will return the overall Enumerated State of the valve.

3.1.10.2.12.2. Sequence

```
getState()
{
	return state;
}
```

3.1.10.2.13. autoCycle()

3.1.10.2.13.1. Descriptive Text

The following will be called by the Panel Class autoValveCheckout method to calculate and record the valve cycle times.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.10.2.13.2. Sequence

Set autoValveCheckoutError = OFF;

open();

if valveCycleError is ON, Set autoValveCheckoutError = ON;

close();

if valveCycleError is ON, Set autoValveCheckoutError = ON;

Return autoValveCheckoutError;

3.1.10.2.14. activateConstraints()

3.1.10.2.14.1. Descriptive Text

This method will activate constraint monitoring of all unbypassed measurement of this valve.

3.1.10.2.14.2. Sequence

```
activateConstraints()
        {
        if(<openPosInd>.BYP == OFF)
                {
                activate constraint monitoring for <openInd>
                }
        if(<openPwr>.BYP == OFF)
                {
                activate constraint monitoring for <openPwr>
                }
        if(<closePosInd>.BYP == OFF)
                {
                activate constraint monitoring for <closeInd>
                }
        if(<openRPC_A>.BYP == OFF)
                {
                activate constraint monitoring for <openRPC_1>
                }
        if(<openRPC_B>.BYP == OFF)
                {
                activate constraint monitoring for <openRPC_2>
                }

        }
```

3.1.10.2.15. inhibitConstraints()

3.1.10.2.15.1. Descriptive Text

This method will inhibit constraint monitoring of all measurements related to this valve.

3.1.10.2.15.2. Sequence

```
inhibitConstraints()
        {
        inhibit constraint monitoring of the following:
        <openInd>
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

<closeInd>
<openPwr>
<openRPC_A>
<openRPC_B>
}

3.1.10.2.16. setOpenConstraints()

3.1.10.2.16.1. Descriptive Text

This method changes the constraint criteria of the measurements associated with this valve to those expected with the valve in the open position.

3.1.10.2.16.2. Sequence

```
void setOpenConstraints()
{
        CHANGE (OP POS IND) CONSTRAINT CRITERIA TO ON;

        CHANGE (OP PWR)  CONSTRAINT CRITERIA TO ON;

        CHANGE (CL POS IND) CONSTRAINT CRITERIA TO OFF;

        CHANGE (OPEN RPC_A) CONSTRAINT CRITERIA TO ON;

        CHANGE (OPEN RPC_B) CONSTRAINT CRITERIA TO ON;
}
```

3.1.10.2.17. setClosedConstraints()

3.1.10.2.17.1. Descriptive Text

This method changes the constraint criteria of the measurements associated with this valve to those expected with the valve in the closed position.

3.1.10.2.17.2. Sequence

```
void setCloseConstraints()
{
        CHANGE (OP POS IND) CONSTRAINT CRITERIA TO OFF;

        CHANGE (OP PWR)  CONSTRAINT CRITERIA TO OFF;

        CHANGE (CL POS IND) CONSTRAINT CRITERIA TO ON;

        CHANGE (OPEN RPC_A) CONSTRAINT CRITERIA TO OFF;

        CHANGE (OPEN RPC_B) CONSTRAINT CRITERIA TO OFF;


}
```

3.1.10.2.18. verifyCommandsOff()

3.1.10.2.18.1. Descriptive Text

This function will verify that both the open and close commands are OFF

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.1.10.2.18.2. Sequence

```
verifyCommandsOff()
{
        open command status1 = byteTest4 <Open Command A>
        open command status2 = byteTest4 <Open Command B>
        open command status3 = byteTest4 <Open Command C>
}
{
        verify  open command status1 && open command status2  && open command status3 status == OFF

        return TRUE
        }
        ELSE return FALSE
}
```

3.1.11. MPS_Valve_ORB_LO2OvBrd_Bleed

3.1.11.1. Description

3.1.11.1.1. Instance Index

Instance	Display User	Other Users
MPS_PV19	MPL	

3.1.11.1.2. Class Diagram

<insert class diagram here>

3.1.11.1.3. Attributes and Aggregations

Attribute	Data Type	Initial Value	Publish / Persistence	Client Access	Description
closeCmdA	DS_ON_OFF				
closeCmdB	DS_ON_OFF				
closeCmdC	DS_ON_OFF				
openPositionInd	DM_ON_OFF				
closePositionIndA	DM_ON_OFF				
closePositionIndB	DM_ON_OFF				
closeRPC1	DM_ON_OFF				
closeRPC2	DM_ON_OFF				
closePower	DM_ON_OFF				

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.1.11.1.4. Associations

Association	Type	Class	Description

3.1.11.1.5. Operations Index

Operation	Description
open()	This method performs the operations necessary to open the valve.
close()	This mthod performs the operations necessary to close the valve.
determineIndState()	This method looks at the position indicators of the valve and returns an enumerated state based on a logic table.
determineCmdState()	This method returns an enumerated state based on valve power and RPC indicators.
determineState()	This method looks at the cmdState and indState pseudo-FDs of the valve and returns an enumerated state based on a logic table.
getIndState()	returns the enumerated indicator state.
getCmdState()	returns the enumerated command state.
getState()	returns the enumerated overall state.
inhibitConstraints()	This method inhibits constraint monitoring of all values associated with this valve.
activateConstraints()	This method activates constraint monitoring of all unbypassed values associated with this valve.
autoCycle()	This method runs the open and close methods of the valve and returns either failure or success.
setOpenConstraints()	This method establishes the constraint criteria expected for the valve in the open position.
setCloseConstraints()	This method establishes the constraint criteria expected for the valve in the closed position.
closeOnCheck()	This method checks for the close RPCs to be ON or bypassed and sends appropriate message.
closeOffCheck()	This method checks for the close RPCs to be OFF or bypassed and sends appropriate messages.
checkClosePosIndOn()	This method checks for the close indicators to be ON or bypassed and sends appropriate message.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

checkClosePosIndOff()	This method checks for the closeindicators to be OFF or bypassed and sends appropriate messages.
verifyCmdsOff()	This method checks to verify all commands are off.
//verifyTminus6BT4//	This method verifies T-6 hour configuration of commands (S/B OFF).
//verifyTminus3BT4//	This method verifies T-3 hour configuration of commands (S/B OFF).

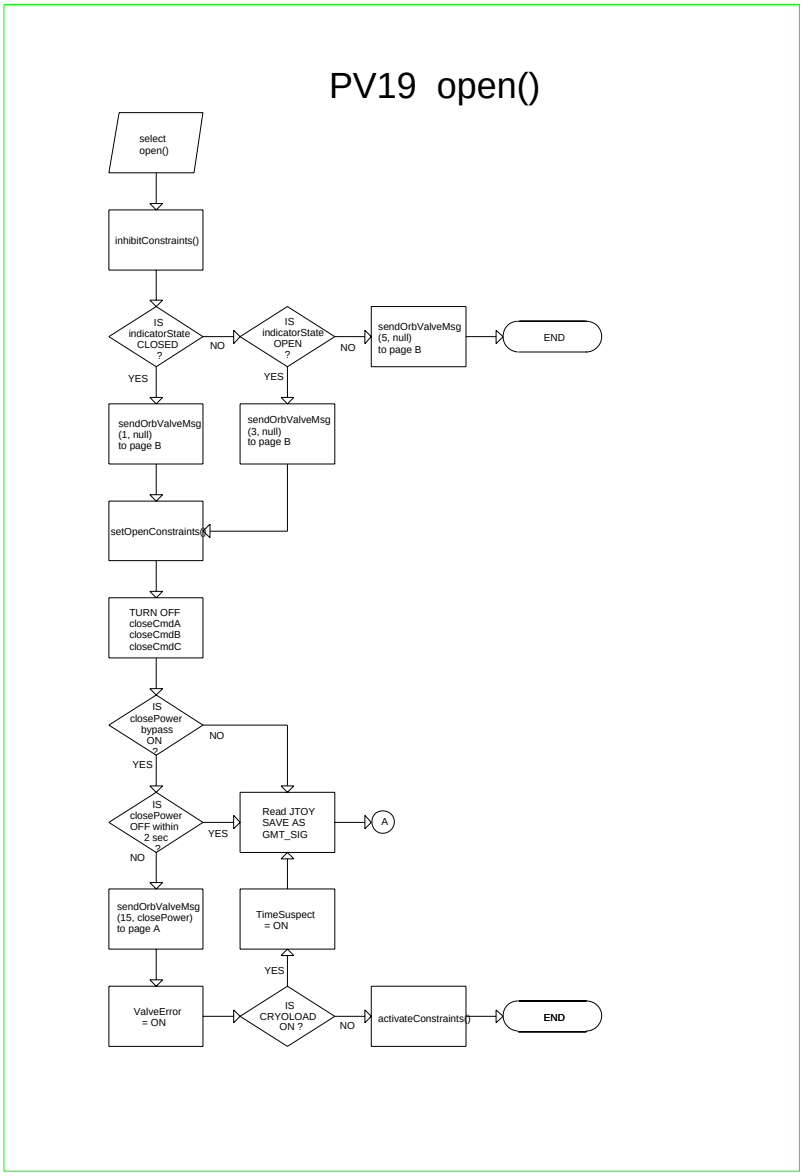
3.1.11.2. Operations

3.1.11.2.1. open()

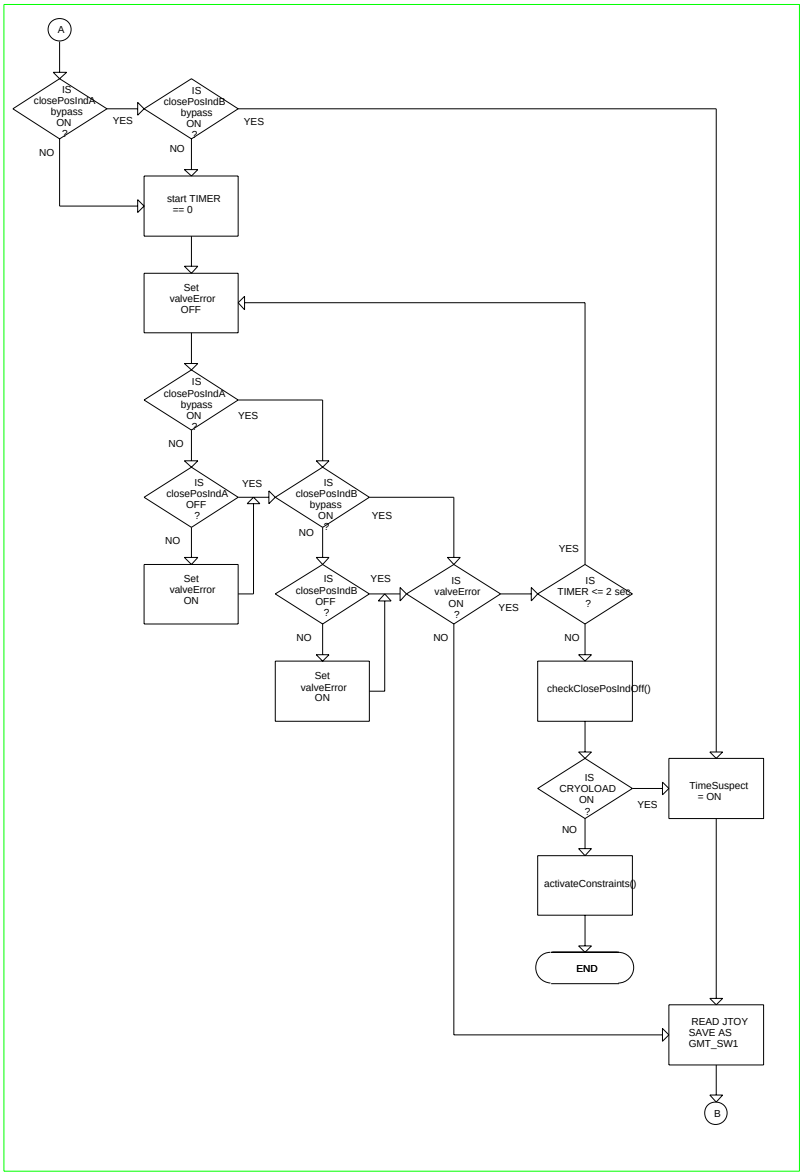
3.1.11.2.1.1. Descriptive Text

This method performs cycling LO2 Overboard Bleed Valve to the open position. The flow chart provides the necessary logic for valve cycle, including checks for the proper response of power indicators, and proper response of position indicators. The valve cycles in a fail safe mode and 'wimps-out' (all commands removed) at the first indication of hardware failure, unless the orbiter is in cryogenic load. If in cryo load, the valve cycles in a 'damn-the-torpedos' mode and completes all steps necessary to achieve the valve cycle, regardless of failures which are detected during the cycle. Valve signal to switch and switch timing is also calculated.

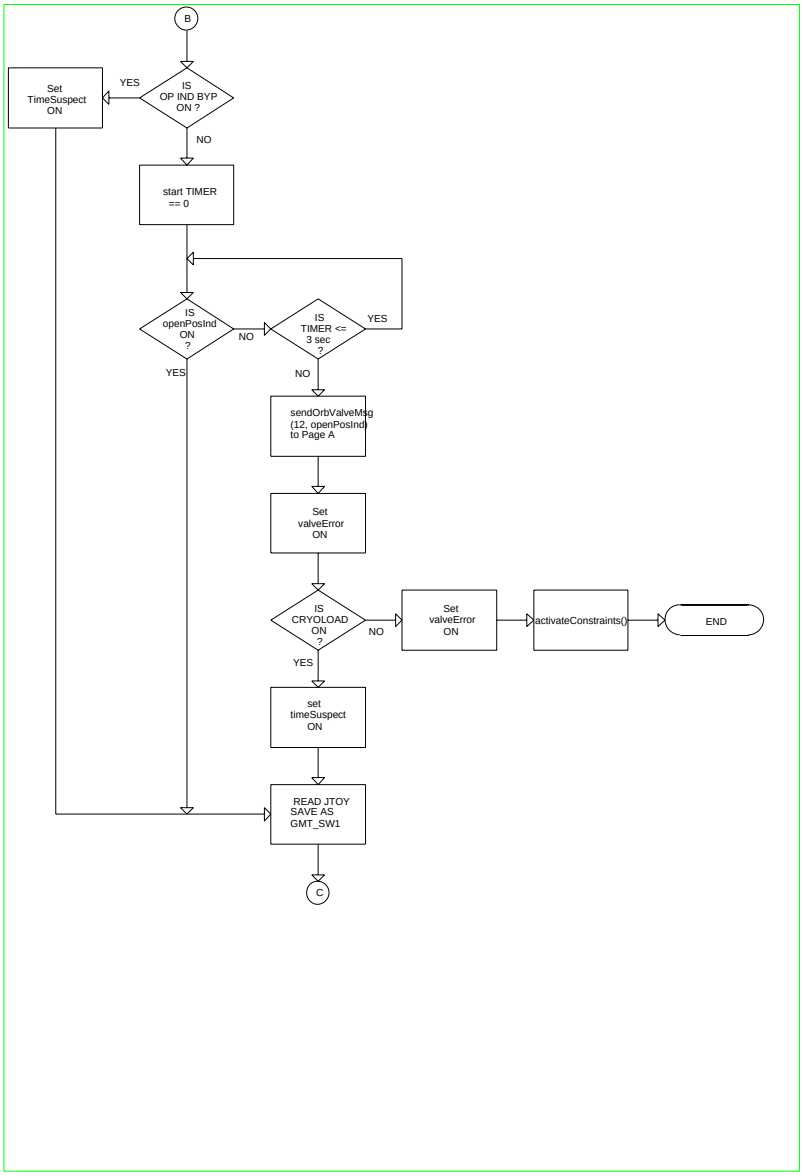
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



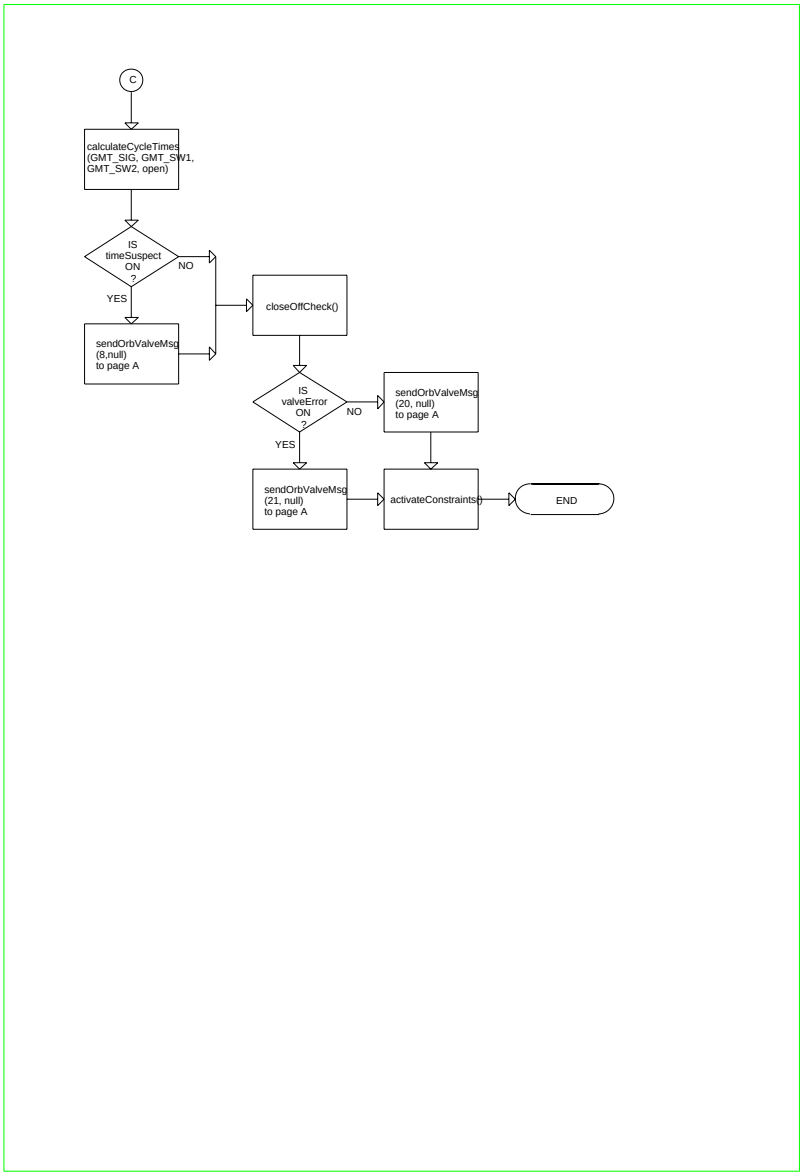
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

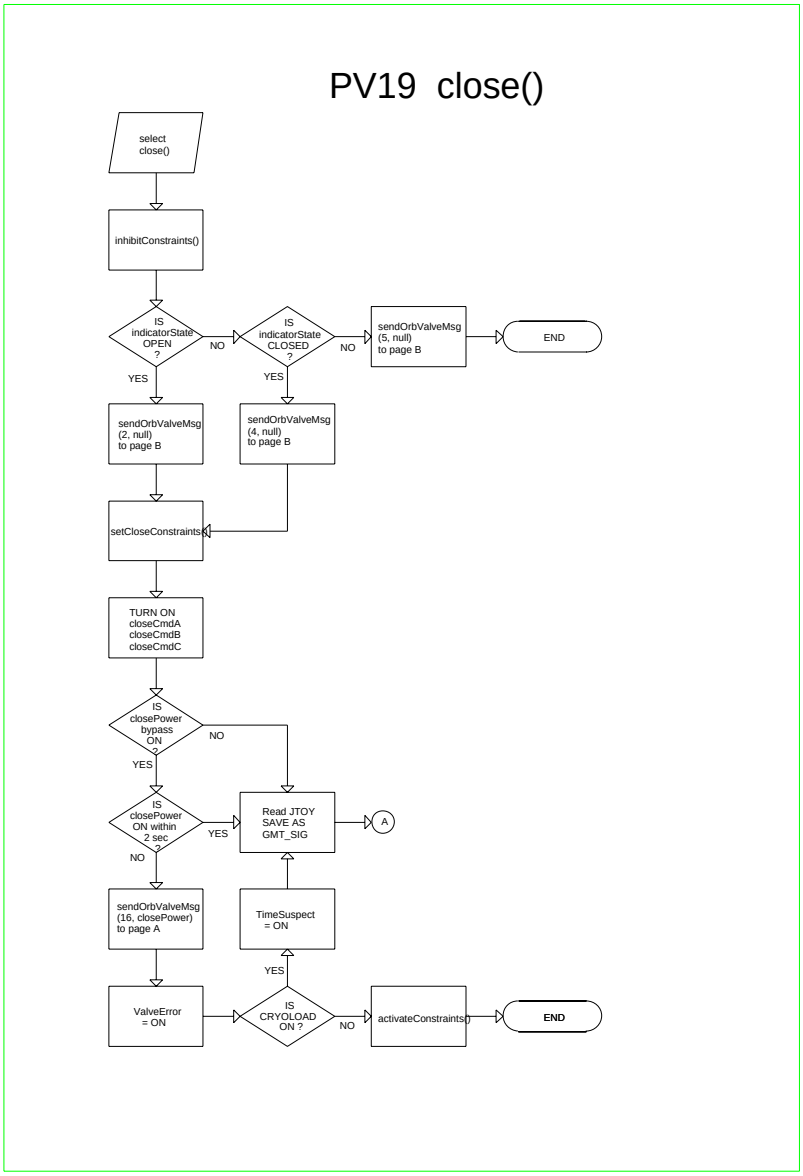


3.1.11.2.2. close()

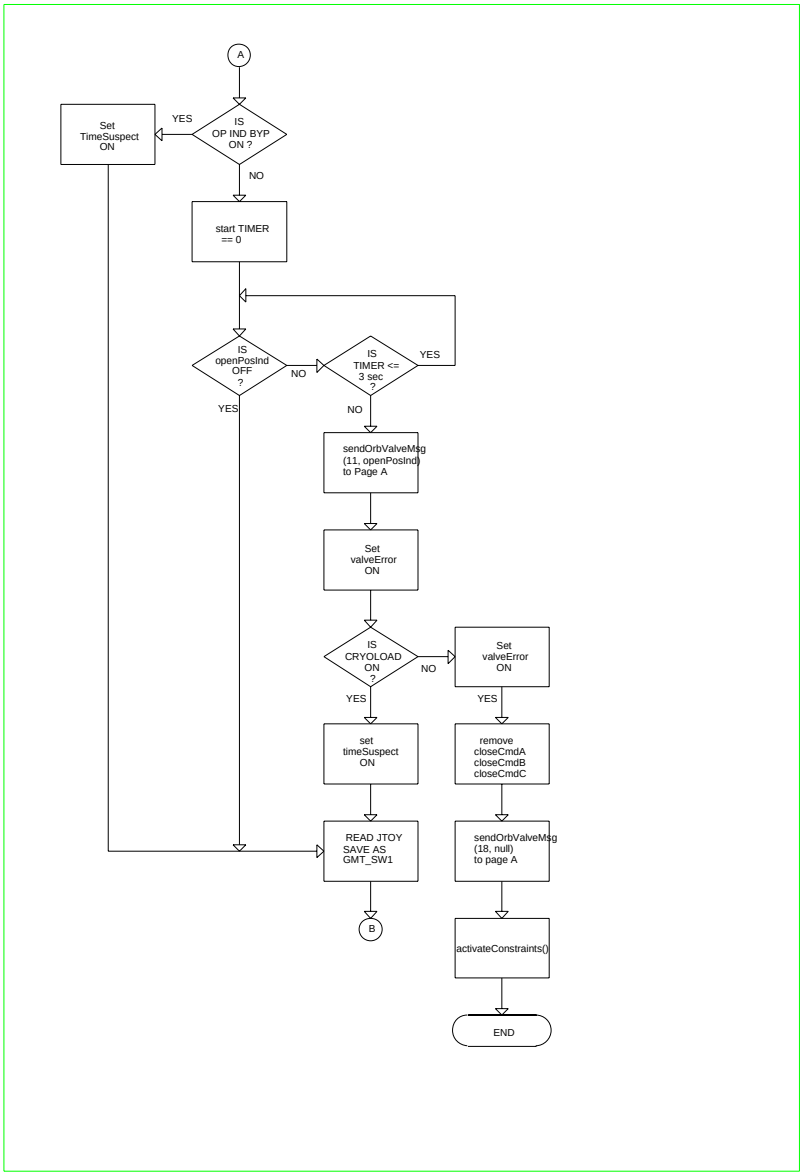
3.1.11.2.2.1. Descriptive Text

This method performs cycling LO2 Overboard Bleed Valve to the closed position. The flow chart provides the necessary logic for valve cycle, including checks for the proper response of power indicators, and proper response of position indicators. The valve cycles in a fail safe mode and 'wimps-out' (all commands removed) at the first indication of hardware failure, unless the orbiter is in cryogenic load. If in cryo load, the valve cycles in a 'damn-the-torpedos' mode and completes all steps necessary to achieve the valve cycle, regardless of failures which are detected during the cycle. Valve signal to switch and switch timing is also calculated.

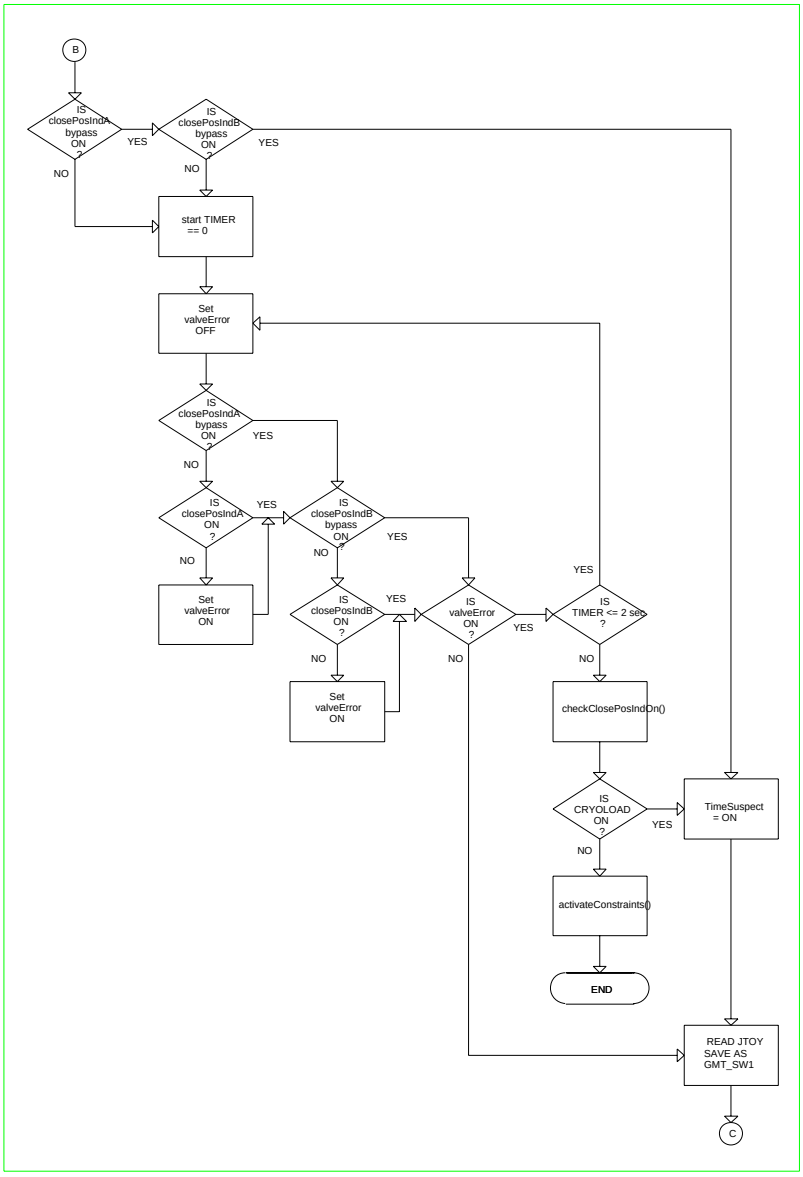
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



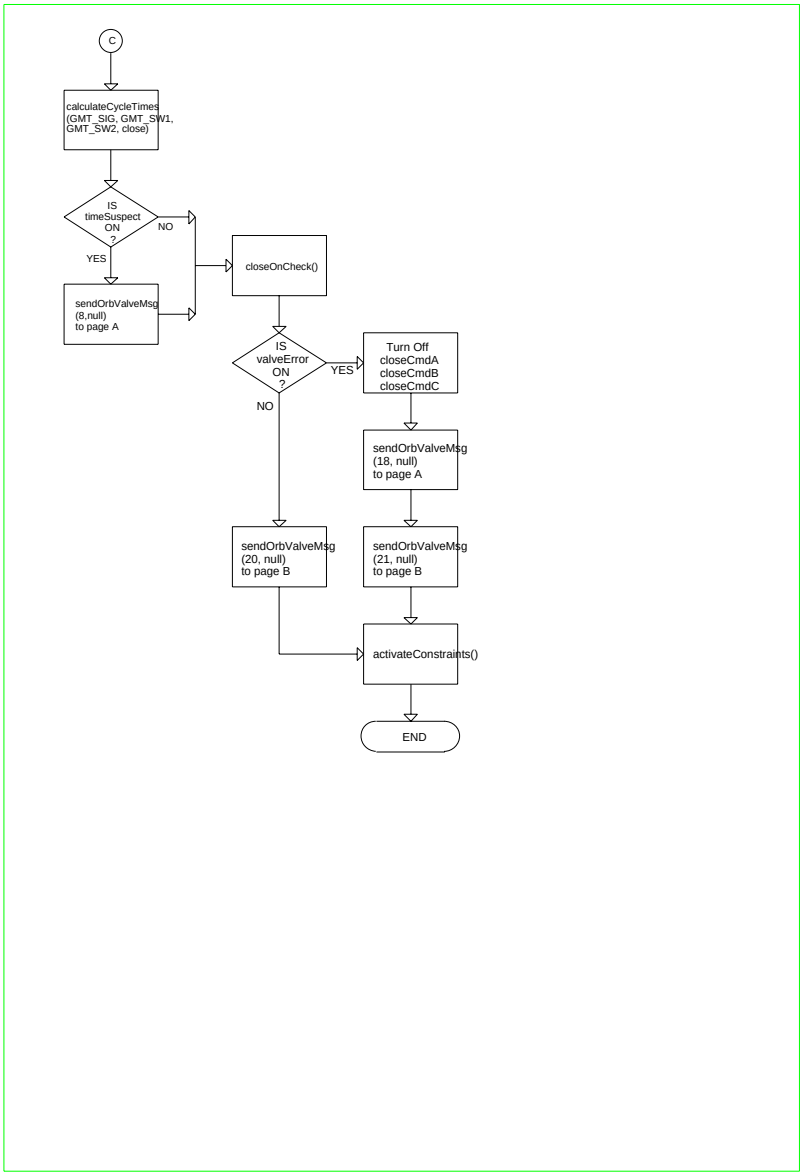
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

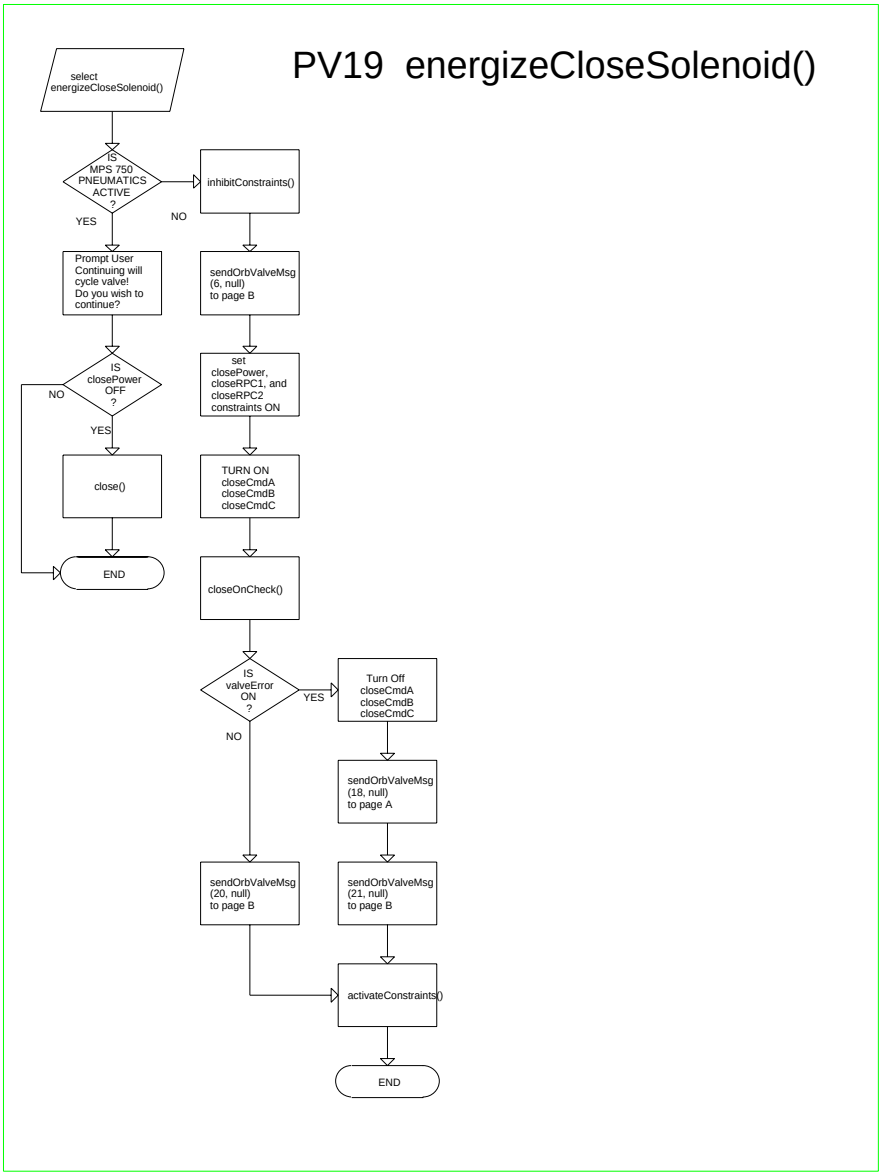


3.1.11.2.3. energizeCloseSolenoid()

3.1.11.2.3.1. Descriptive Text

This method performs applies power to the close solenoid. If MPS 750 pneumatics are active the close() method is called. The flow chart provides the necessary logic for the method, including checks for the proper response of power indicators. The valve cycles in a fail safe mode and 'wimps-out' (all commands removed) at the first indication of hardware failure

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

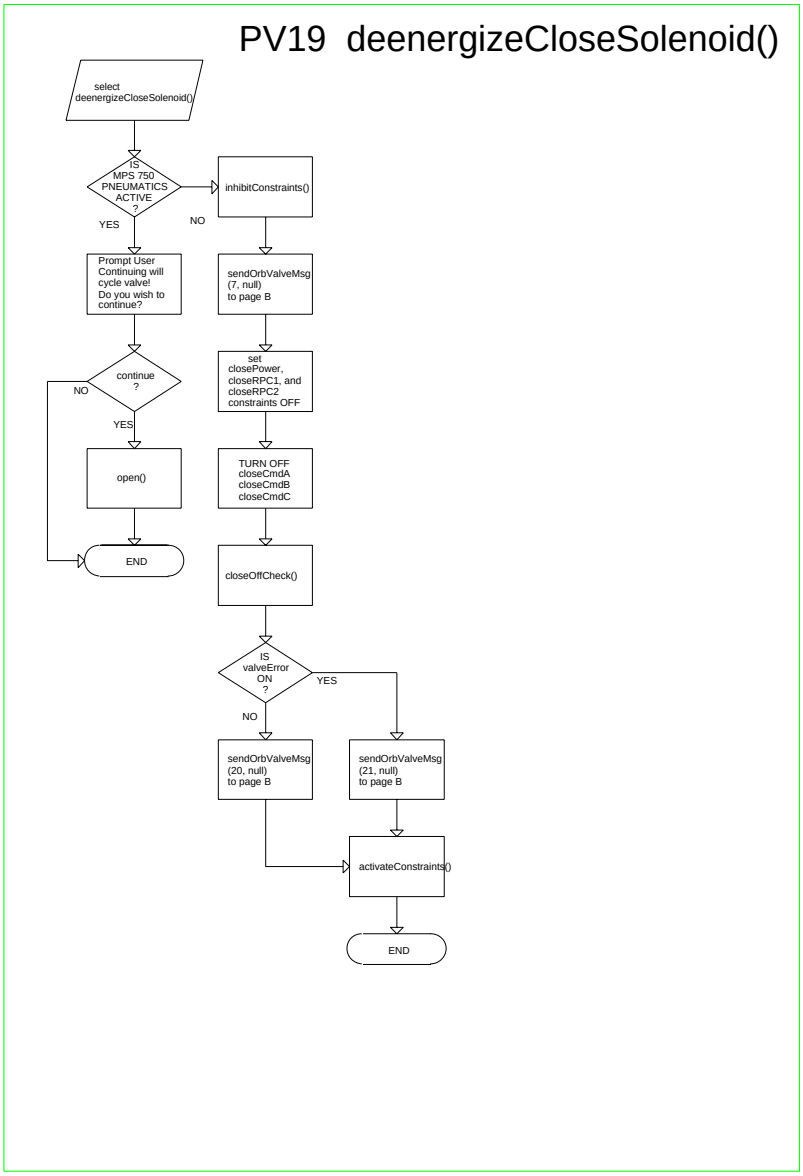


3.1.11.2.4. deenergizeCloseSolenoid()

3.1.11.2.4.1. Descriptive Text

This method performs removes power from the close solenoid. If MPS 750 pneumatics are active the open() method is called. The flow chart provides the necessary logic for the method, including checks for the proper response of power indicators. The valve cycles in a fail safe mode and 'wimps-out' (all commands removed) at the first indication of hardware failure

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



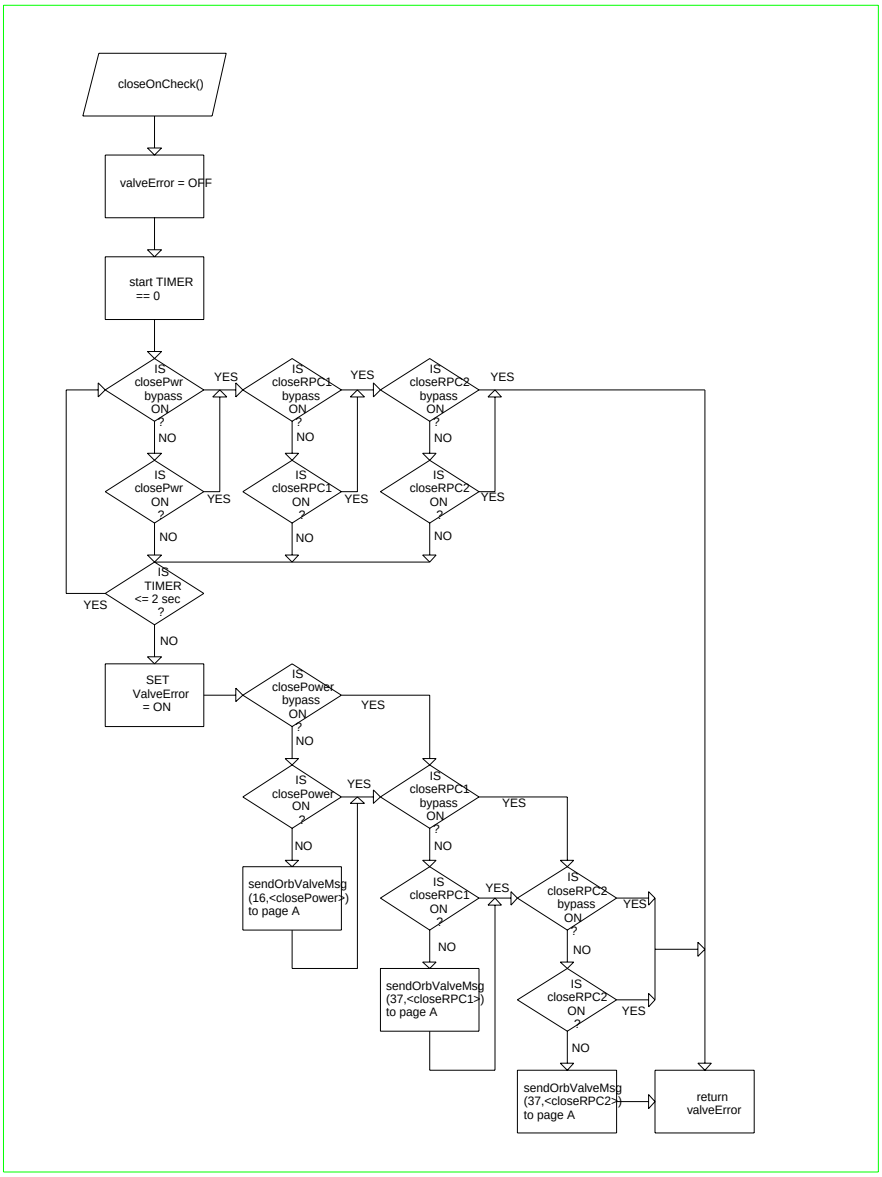
3.1.11.2.5. closeOnCheck()

3.1.11.2.5.1. Descriptive Text

This method checks the close RPCS, and sends a message for each unbypassed RPC that is OFF.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.11.2.5.2. Sequence



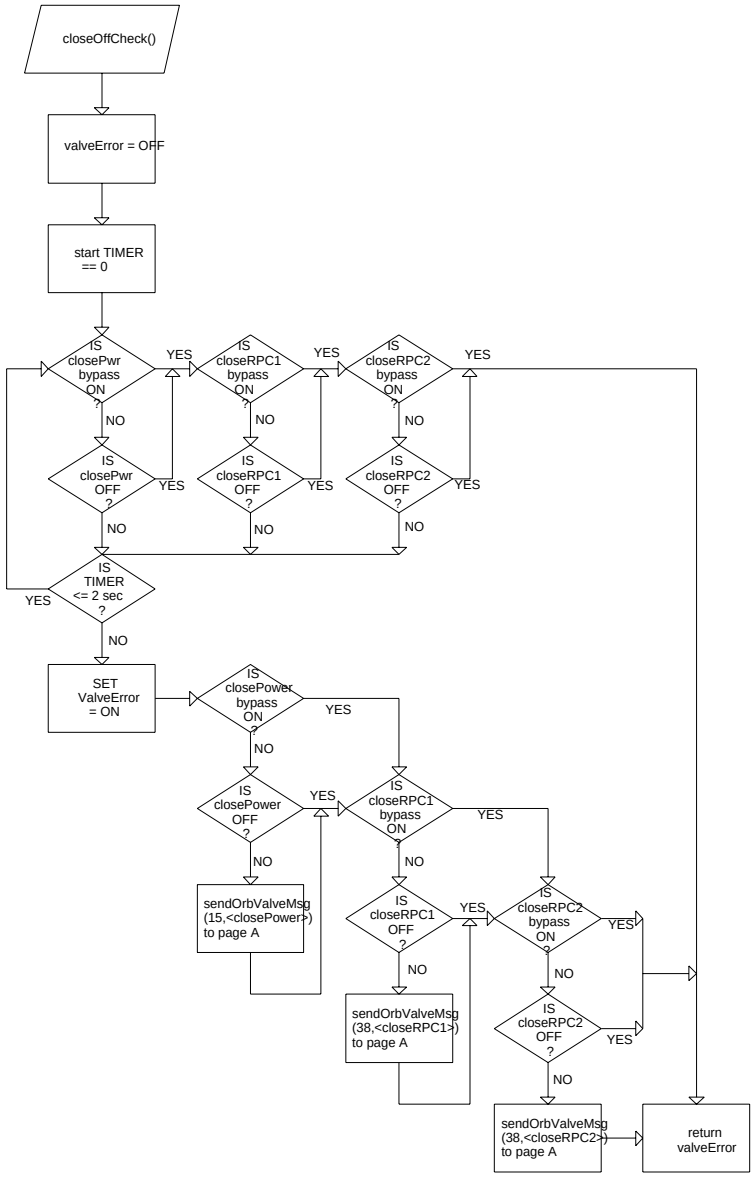
3.1.11.2.6. closeOffCheck()

3.1.11.2.6.1. Descriptive Text

This method checks the close RPCS, and sends a message for each unbypassed RPC that is ON.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.11.2.6.2. Sequence



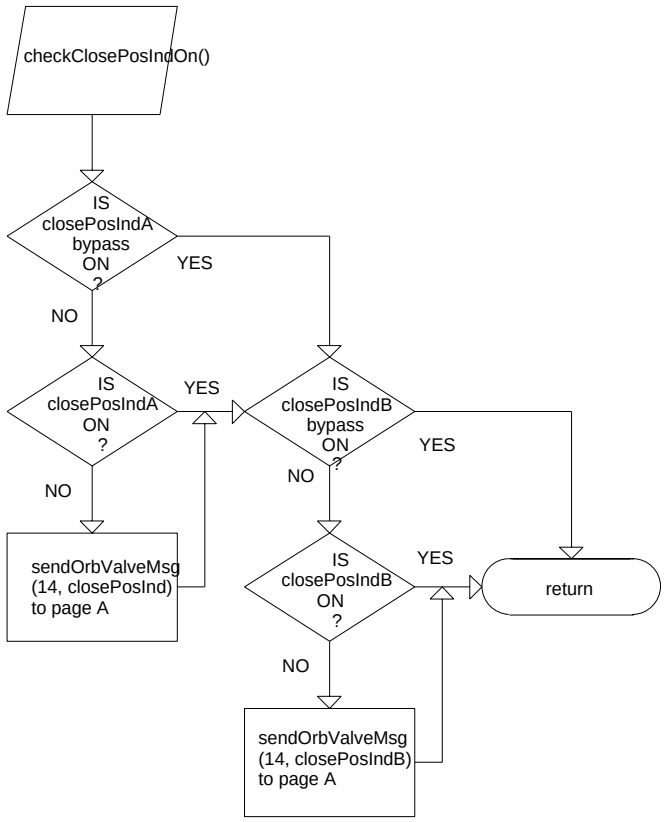
3.1.11.2.7. checkClosePosIndOn()

3.1.11.2.7.1. Descriptive Text

This method checks the closed position indicators, and sends a message for each unbypassed closed position indicator that is Off.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.11.2.7.2. Sequence



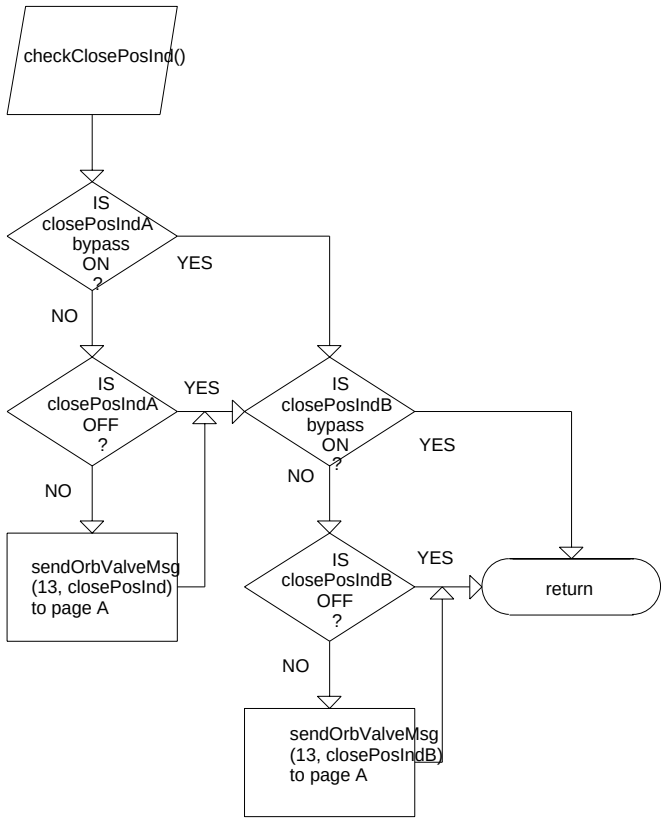
3.1.11.2.8. checkClosePosIndOff()

3.1.11.2.8.1. Descriptive Text

This method checks the closed position indicators, and sends a message for each unbypassed closed position indicator that is ON.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.11.2.8.2. Sequence



3.1.11.2.9. determineIndState()

3.1.11.2.9.1. Descriptive Text

This method will determine the Indicator State of the valve based on an "All of all unbypassed indicator" logic stated below:

3.1.11.2.9.2. Sequence

determineIndState()

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

Indicator State						
open Indicator	open Indicator Byp	close Indicator A	close Indicator A BYP	close Indicator B	close Indicator B BYP	Indicator State
OFF	OFF	----	ON	----	ON	close
OFF	OFF	----	ON	ON	OFF	close
OFF	OFF	ON	OFF	----	ON	close
OFF	OFF	ON	OFF	ON	OFF	close
----	ON	----	ON	ON	OFF	close
----	ON	ON	OFF	----	ON	close
----	ON	ON	OFF	ON	OFF	close
----	ON	OFF	OFF	OFF	OFF	open
----	ON	OFF	OFF	----	ON	open
----	ON	----	ON	OFF	OFF	open
ON	OFF	OFF	OFF	OFF	OFF	open
ON	OFF	OFF	OFF	----	ON	open
ON	OFF	----	ON	OFF	OFF	open
ON	OFF	----	ON	----	ON	open
OFF	OFF	OFF	OFF	OFF	OFF	transition
OFF	OFF	OFF	OFF	----	ON	transition
OFF	OFF	----	ON	OFF	OFF	transition
OFF	OFF	OFF	OFF	ON	OFF	error
OFF	OFF	ON	OFF	OFF	OFF	error
----	ON	OFF	OFF	ON	OFF	error
----	ON	----	ON	----	ON	error
----	ON	ON	OFF	OFF	OFF	error
ON	OFF	OFF	OFF	ON	OFF	error
ON	OFF	----	ON	ON	OFF	error
ON	OFF	ON	OFF	OFF	OFF	error
ON	OFF	ON	OFF	----	ON	error
ON	OFF	ON	OFF	ON	OFF	error

3.1.11.2.10. determineCmdState()

3.1.11.2.10.1. Descriptive Text

This method will use the "All of all unbypassed indicator" logic to determine the command state using the two close RPCs and the close power indicator.

3.1.11.2.10.2. Sequence

```
determineCmdState()
        {
        CASE:
                <closePwrByp> == ON
                <closeRPCAByp> == ON
                <closeIRPCB> == ON
                <closeRPCBByp> == OFF
                                        <NMPSLV76CMDSTATE> = CMD_CLOSE

        CASE:
                <closePwrByp> == ON
                <closeRPCA> == ON
                <closeRPCAByp> == OFF
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

<closeRPCBByp> == ON
<NMPSLV76CMDSTATE> = CMD_CLOSE

CASE:
<closePwrByp> == ON
<closeRPCA> == ON
<closeRPCAByp> == OFF
<closeRPCBByp> == ON
<NMPSLV76CMDSTATE> = CMD_CLOSE

CASE:
<closePwr> == ON
<closePwrByp> == OFF
<closeRPCAByp> == ON
<closeRPCBByp> == ON
<NMPSLV76CMDSTATE> = CMD_CLOSE

CASE:
<closePwr> == ON
<closePwrByp> == OFF
<closeRPCAByp> == ON
<closeIRPCB> == ON
<closeRPCBByp> == OFF
<NMPSLV76CMDSTATE> = CMD_CLOSE

CASE:
<closePwr> == ON
<closePwrByp> == OFF
<closeRPCA> == ON
<closeRPCAByp> == OFF
<closeRPCBByp> == ON
<NMPSLV76CMDSTATE> = CMD_CLOSE

CASE:
<closePwr> == ON
<closePwrByp> == OFF
<closeRPCA> == ON
<closeRPCAByp> == OFF
<closeRPCB> == ON
<closeRPCBByp> == OFF
<NMPSLV76CMDSTATE> = CMD_CLOSE

CASE:
<closePwr> == OFF
<closePwrByp> == OFF
<closeRPCA> == OFF
<closeRPCAByp> == OFF
<closeRPCB> == OFF
<closeRPCBByp> == OFF
<NMPSLV76CMDSTATE> = CMD_OPEN

CASE:
<closePwr> == OFF
<closePwrByp> == OFF

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
                <closeRPCA> == OFF
                <closeRPCAByp> == OFF
                <closeRPCBByp> == ON
                                        <NMPSLV76CMDSTATE> = CMD_OPEN

        CASE:
                <closePwr> == OFF
                <closePwrByp> == OFF
                <closeRPCAByp> == ON
                <closeRPCB> == OFF
                <closeRPCBByp> == OFF
                                        <NMPSLV76CMDSTATE> = CMD_OPEN

        CASE:
                <closePwr> == OFF
                <closePwrByp> == OFF
                <closeRPCAByp> == ON
                <closeRPCBByp> == ON
                                        <NMPSLV76CMDSTATE> = CMD_OPEN

        CASE:

                <closePwrByp> == ON
                <closeRPCA> == OFF
                <closeRPCAByp> == OFF
                <closeRPCB> == OFF
                <closeRPCBByp> == OFF
                                        <NMPSLV76CMDSTATE> = CMD_OPEN

        CASE:
                <closePwrByp> == ON
                <closeRPCA> == OFF
                <closeRPCAByp> == OFF
                <closeRPCBByp> == ON
                                        <NMPSLV76CMDSTATE> = CMD_OPEN
        CASE:
                <closePwrByp> == ON
                <closeRPCAByp> == ON
                <closeRPCB> == OFF
                <closeRPCBByp> == OFF
                                        <NMPSLV76CMDSTATE> = CMD_OPEN

        CASE ELSE
                                        <NMPSLV76CMDSTATE> = ERROR


determineCmdState()
        {
        CASE:
                <closePwrByp> == ON
                <closeRPCAByp> == ON
                <closeIRPCB> == ON
                <closeRPCBByp> == OFF
                                        <NMPSLV76CMDSTATE> = CMD_CLOSE
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

CASE:
<closePwrByp> == ON
<closeRPCA> == ON
<closeRPCAByp> == OFF
<closeRPCBByp> == ON
<NMPSLV76CMDSTATE> = CMD_CLOSE

CASE:
<closePwrByp> == ON
<closeRPCA> == ON
<closeRPCAByp> == OFF
<closeRPCBByp> == ON
<NMPSLV76CMDSTATE> = CMD_CLOSE

CASE:
<closePwr> == ON
<closePwrByp> == OFF
<closeRPCAByp> == ON
<closeRPCBByp> == ON
<NMPSLV76CMDSTATE> = CMD_CLOSE

CASE:
<closePwr> == ON
<closePwrByp> == OFF
<closeRPCAByp> == ON
<closeIRPCB> == ON
<closeRPCBByp> == OFF
<NMPSLV76CMDSTATE> = CMD_CLOSE

CASE:
<closePwr> == ON
<closePwrByp> == OFF
<closeRPCA> == ON
<closeRPCAByp> == OFF
<closeRPCBByp> == ON
<NMPSLV76CMDSTATE> = CMD_CLOSE

CASE:
<closePwr> == ON
<closePwrByp> == OFF
<closeRPCA> == ON
<closeRPCAByp> == OFF
<closeRPCB> == ON
<closeRPCBByp> == OFF
<NMPSLV76CMDSTATE> = CMD_CLOSE

CASE:
<closePwr> == OFF
<closePwrByp> == OFF
<closeRPCA> == OFF
<closeRPCAByp> == OFF
<closeRPCB> == OFF
<closeRPCBByp> == OFF

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

<NMPSLV76CMDSTATE> = CMD_OPEN

CASE:
<closePwr> == OFF
<closePwrByp> == OFF
<closeRPCA> == OFF
<closeRPCAByp> == OFF
<closeRPCBByp> == ON
<NMPSLV76CMDSTATE> = CMD_OPEN

CASE:
<closePwr> == OFF
<closePwrByp> == OFF
<closeRPCAByp> == ON
<closeRPCB> == OFF
<closeRPCBByp> == OFF
<NMPSLV76CMDSTATE> = CMD_OPEN

CASE:
<closePwr> == OFF
<closePwrByp> == OFF
<closeRPCAByp> == ON
<closeRPCBByp> == ON
<NMPSLV76CMDSTATE> = CMD_OPEN

CASE:

<closePwrByp> == ON
<closeRPCA> == OFF
<closeRPCAByp> == OFF
<closeRPCB> == OFF
<closeRPCBByp> == OFF
<NMPSLV76CMDSTATE> = CMD_OPEN

CASE:
<closePwrByp> == ON
<closeRPCA> == OFF
<closeRPCAByp> == OFF
<closeRPCBByp> == ON
<NMPSLV76CMDSTATE> = CMD_OPEN
CASE:
<closePwrByp> == ON
<closeRPCAByp> == ON
<closeRPCB> == OFF
<closeRPCBByp> == OFF
<NMPSLV76CMDSTATE> = CMD_OPEN

CASE ELSE
<NMPSLV76CMDSTATE> = ERROR

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.1.11.2.11. determineState()

3.1.11.2.11.1. Descriptive Text

This method will determine the overall state of the valve based on the combined Command and Indicator States using the logic shown below.

3.1.11.2.11.2. Sequence

determineState()
{

	State (overall)		
pos\cmd	cmd closed	cmd open	error
closed	closed	close error	close error
open	open error	open	open error
transition	transition	transition	error
error	error	error	error

3.1.11.2.12. getIndState()

3.1.11.2.12.1. Descriptive Text

This method will return the indicator state of the valve.

3.1.11.2.12.2. Sequence

getIndState()
{
return IndState;
}

3.1.11.2.13. getCmdState()

3.1.11.2.13.1. Descriptive Text

This method will return the command state of the valve.

3.1.11.2.13.2. Sequence

getCmdState()
{
return CmdState;
}

3.1.11.2.14. getState()

3.1.11.2.14.1. Descriptive Text

This method will return the overall state of the valve.

3.1.11.2.14.2. Sequence

getState()
{
return state;
}

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.11.2.15. autoCycle()

3.1.11.2.15.1. Descriptive Text

The following will be called by the Panel Class autoValveCheckout method
to calculate and record the valve cycle times.

3.1.11.2.15.2. Sequence

Set autoValveCheckoutError = OFF;

close();

if valveCycleError is ON, Set autoValveCheckoutError = ON;

open();

if valveCycleError is ON, Set autoValveCheckoutError = ON;

Return autoValveCheckoutError;

3.1.11.2.16. activateConstraints()

3.1.11.2.16.1. Descriptive Text

This method activates constraint monitoring of all unbypassed FDs associated with this valve.

3.1.11.2.16.2. Sequence

activateConstraints()
{
If (<closePwrByp> == OFF)
{
activate Constraint Criteria monitoring of <closePwr>
}
If (<closeRPCAByp> == OFF)
{
activate Constraint Criteria monitoring of <closeRPCA>
}
If (<closeRPCBByp> == OFF)
{
activate Constraint Criteria monitoring of <closeRPCB>
}
If (<openIndByp> == OFF)
{
activate Constraint Criteria monitoring of <openInd>
}
If (<closeIndAByp> == OFF)
{
activate Constraint Criteria monitoring of <closeIndA>
}
If (<closeIndBByp> == OFF)
{
activate Constraint Criteria monitoring of <closeIndB>
}
}

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.1.11.2.17. inhibitConstraints()

3.1.11.2.17.1. Descriptive Text

This method will inhibit constraint criteria monitoring of the FDs associated with this valve.

3.1.11.2.17.2. Sequence

inhibitConstraints()
{
inhibit Constraint Criteria monitoring of the following FDs:

<closePwr>
<closeRPCA>
<closeRPCB>
<openInd>
<closeIndA>
<closeIndB>
}

3.1.11.2.18. setOpenConstraints()

3.1.11.2.18.1. Descriptive Text

This method sets the constraint criteria for the FDs associated with this valve to the states expected of an open valve.

3.1.11.2.18.2. Sequence

setOpenConstraints()
{
set up the constraints as shown below:

FD	Constraint Criteria
<closePwr>	OFF
<closeRPCA>	OFF
<closeRPCB>	OFF
<openInd>	ON
<closeIndA>	OFF
<closeIndB>	OFF

}

3.1.11.2.19. setCloseConstraints()

3.1.11.2.19.1. Descriptive Text

This method sets the constraint criteria for the FDs associated with this valve to the states expected of an closed valve.

3.1.11.2.19.2. Sequence

setCloseConstraints()
{
set up the constraints as shown below:

FD	Constraint Criteria
<closePwr>	ON

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
        <closeRPCA>                     ON
        <closeRPCB>                     ON
        <openInd>                       OFF
        <closeIndA>                     ON
        <closeIndB>                     ON

        }
```

3.1.11.2.20. verifyCommandsOff()

3.1.11.2.20.1. Descriptive Text

This function will verify that both the open and close commands are OFF

3.1.11.2.20.2. Sequence

```
verifyCommandsOff()
{
        open command status = byteTest4 <Open Command>
        close command status = byteTest4 <Close Command>
        {
        verify  open command status && close command status == OFF

        return TRUE
        }
        ELSE return FALSE
}
```

3.1.11.2.21. //verifyTminus6BT4()//

```
begin sequence;

set commanderrorcheck OFF
{
perform ByteTest4 of <openCMDA>
        verify ByteTest4 = OFF
        else send message to page A
                ValveID <openCMDA> "is not in T-6 hour configuration."
                and set commanderrorcheck ON

perform ByteTest4 of <openCMDB>
        verify ByteTest4 = OFF
        else send message to page A
                ValveID <openCMDB> "is not in T-6 hour configuration."
                and set commanderrorcheck ON

perform ByteTest4 of <openCMDC>
        verify ByteTest4 = OFF
        else send message to page A
                ValveID <openCMDC> "is not in T-6 hour configuration."
                and set commanderrorcheck ON
}

return commanderrorcheck

end sequence;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.1.11.2.22. //verifyTminus6BT4()//

```
begin sequence;

set commanderrorcheck OFF
{
perform ByteTest4 of <openCMDA>
        verify ByteTest4 = OFF
        else send message to page A
                ValveID <openCMDA> "is not in T-3 hour configuration."
                and set commanderrorcheck ON

perform ByteTest4 of <openCMDB>
        verify ByteTest4 = OFF
        else send message to page A
                ValveID <openCMDB> "is not in T-3 hour configuration."
                and set commanderrorcheck ON

perform ByteTest4 of <openCMDC>
        verify ByteTest4 = OFF
        else send message to page A
                ValveID <openCMDC> "is not in T-3 hour configuration."
                and set commanderrorcheck ON
}

return commanderrorcheck

end sequence;
```

3.1.12. MPS_Valve_ORB_Pogos

3.1.12.1. Description

3.1.12.1.1. Instance Index

Instance	Display User	Other Users
MPS_PV20	MPL	
MPS_PV21	MPL	

3.1.12.1.2. Class Diagram

<insert class diagram here>

3.1.12.1.3. Attributes and Aggregations

Attribute	Data Type	Initial Value	Publish / Persistence	Client Access	Description
closeCmdA	DS_ON_OFF				

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

closeCmdB	DS_ON_OFF				
openPositionInd	DM_ON_OFF				
closePositionInd	DM_ON_OFF				
closePower	DM_ON_OFF				

3.1.12.1.4. Associations

Association	Type	Class	Description

3.1.12.1.5. Operations Index

Operation	Description
open()	This method performs the operations necessary to open the valve.
close()	This method performs the operations necessary to close the valve.
energizeCloseSolenoid()	This method performs the operations necessary to energize the valve closing solenoid.
deenergizeCloseSolenoid()	This method performs the operations necessary to deenergize the valve closing solenoid.
determineIndState()	This method looks at the position indicators of the valve and returns an enumerated state based on a logic table.
determineCmdState()	This method returns an enumerated state based on valve power and RPC indicators.
determineState()	This method looks at the cmdState and indState pseudo-FDs of the valve and returns an enumerated state based on a logic table.
getIndState()	returns the enumerated indicator state.
getCmdState()	returns the enumerated command state.
getState()	returns the enumerated overall state.
inhibitConstraints()	This method inhibits constraint monitoring of all values associated with this valve.
activateConstraints()	This method activates constraint monitoring of all unbypassed values associated with this valve.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

autoCycle()	This method runs the open and close methods of the valve and returns either failure or success.
setOpenConstraints()	This method establishes the constraint criteria expected for the valve in the open position.
setCloseConstraints()	This method establishes the constraint criteria expected for the valve in the closed position.
verifyCmdsOff()	This method checks to verify all commands are off.
//verifyTminus6BT4//	This method verifies T-6 hour configuration of commands (S/B OFF).
//verifyTminus3BT4//	This method verifies T-3 hour configuration of commands (S/B OFF).

3.1.12.2. Operations

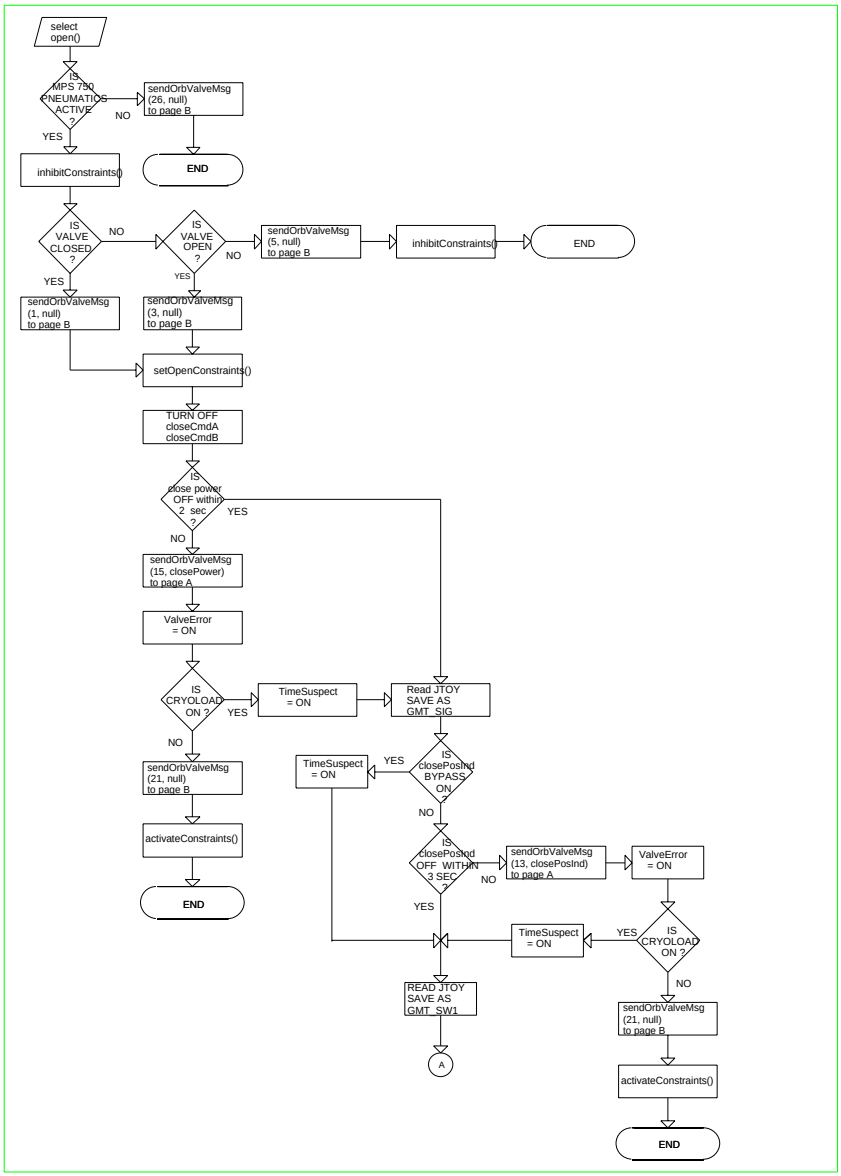
3.1.12.2.1. open()

3.1.12.2.1.1. Descriptive Text

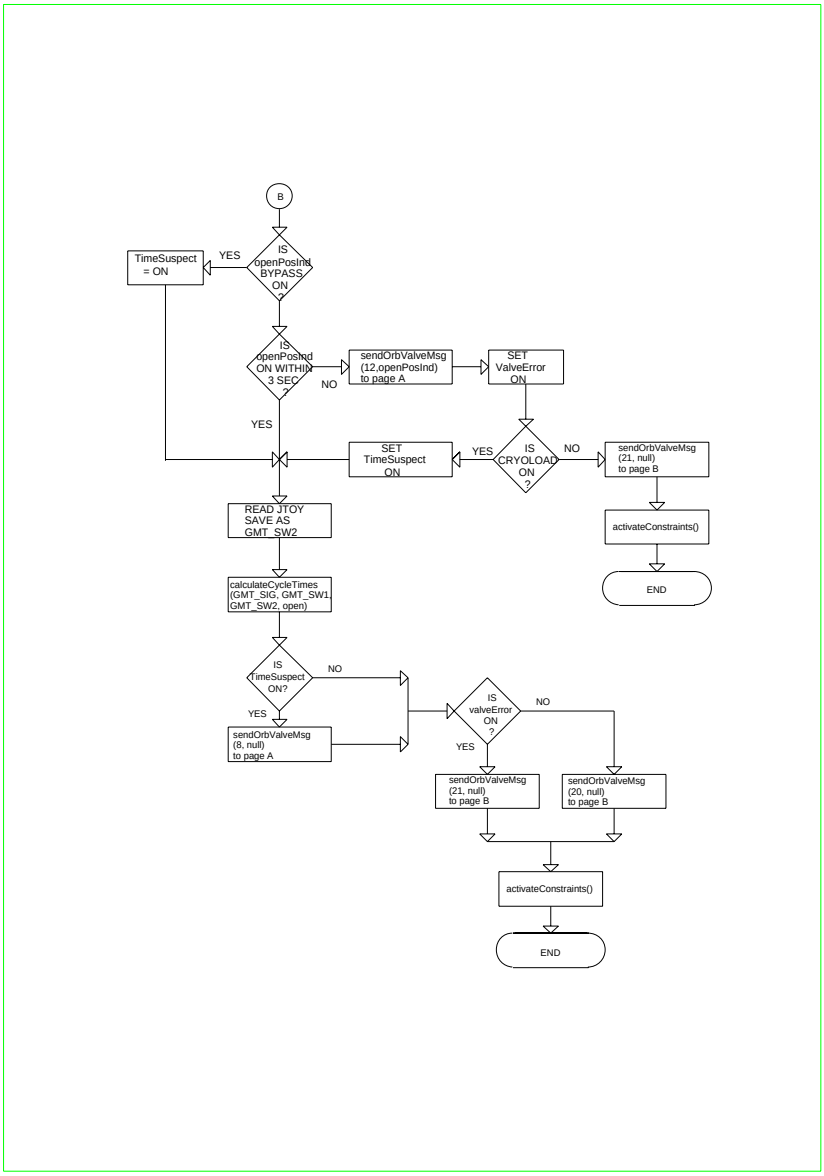
This method opens the valve by removing the close commands, and verifying that the valves indicators update to the appropriate conditions. The flow charts contained below describe the logic to be used to code the method. The implementation of the details is left up to the coders and is acceptable as long as the overall logic is maintained. It is expected that cycle timing will be accomplished using the most accurate source. All messages will include a time stamp, and the appropriate FD description.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.12.2.1.2. Sequence



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



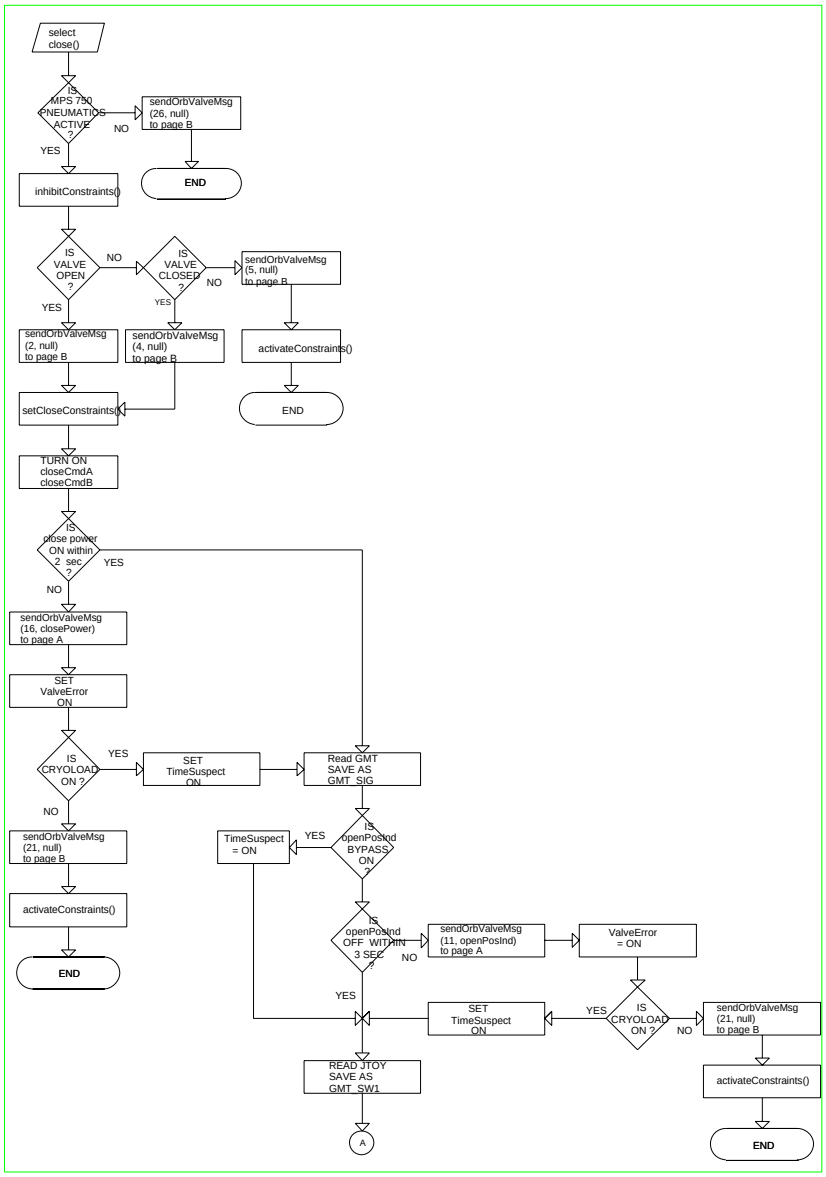
3.1.12.2.2. close()

3.1.12.2.2.1. Descriptive Text

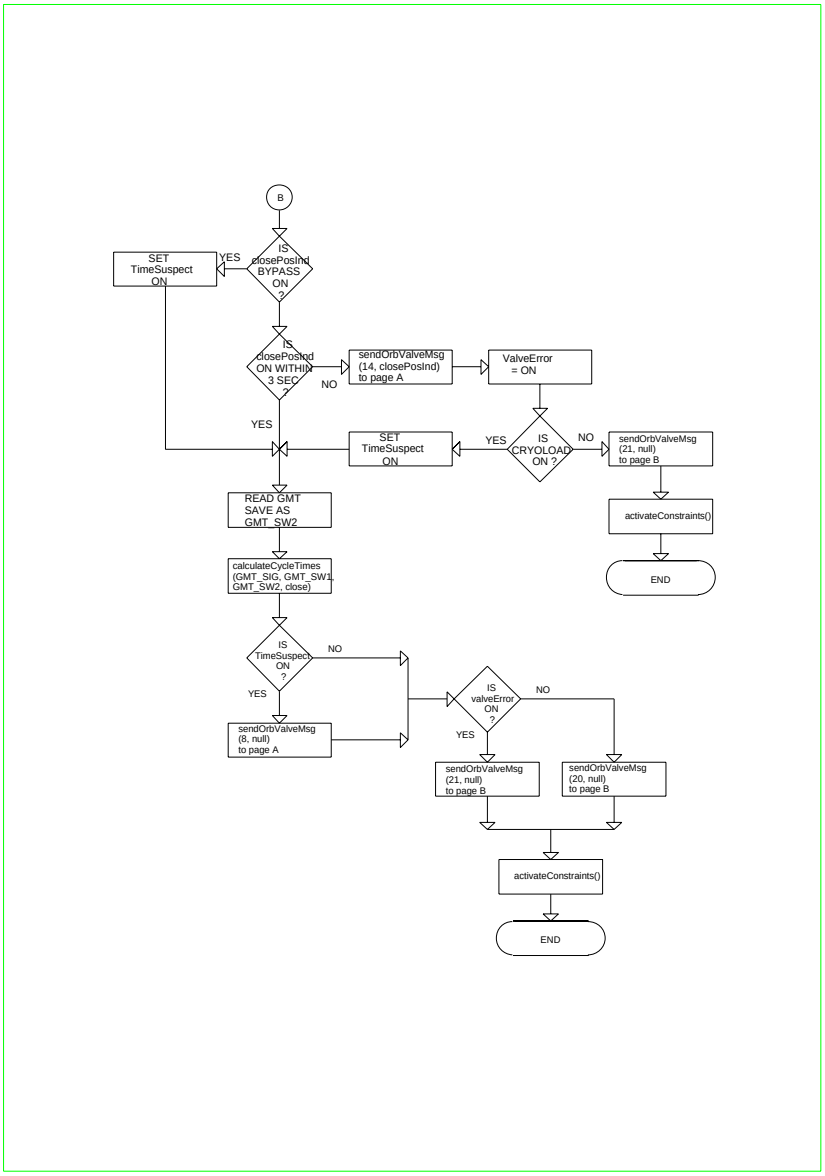
The flow charts contained below describe the logic to be used to code the method. The implementation of the details is left up to the coders and is acceptable as long as the overall logic is maintained. It is expected that cycle timing will be accomplished using the most accurate source. All messages will include a time stamp, and the appropriate FD description.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.12.2.2.2. Sequence



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



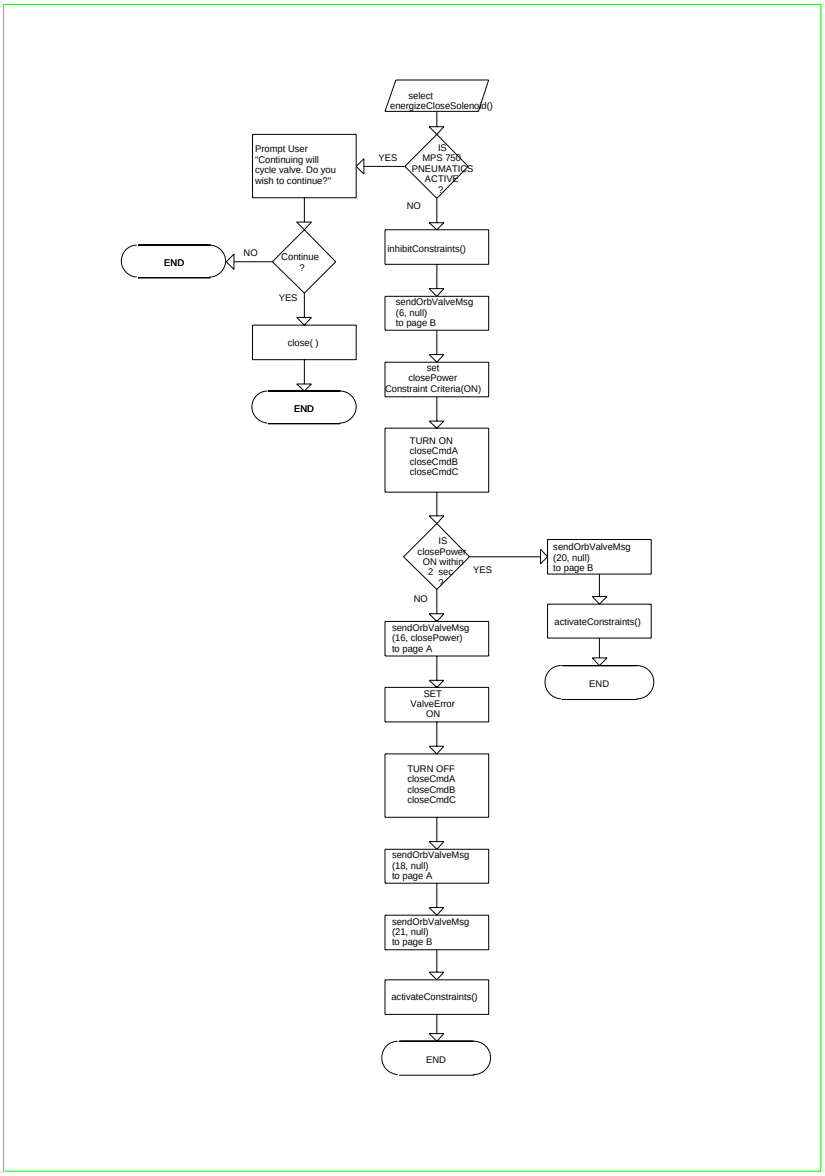
3.1.12.2.3. energizeCloseSolenoid()

3.1.12.2.3.1. Descriptive Text

This method performs applies power to the close solenoid. If MPS 750 pneumatics is active the close() method is called. The flow chart provides the necessary logic for the method, including checks for the proper response of power indicators. The valve cycles in a fail safe mode and 'wimps-out' (all commands removed) at the first indication of hardware failure

3.1.12.2.3.2. Sequence

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



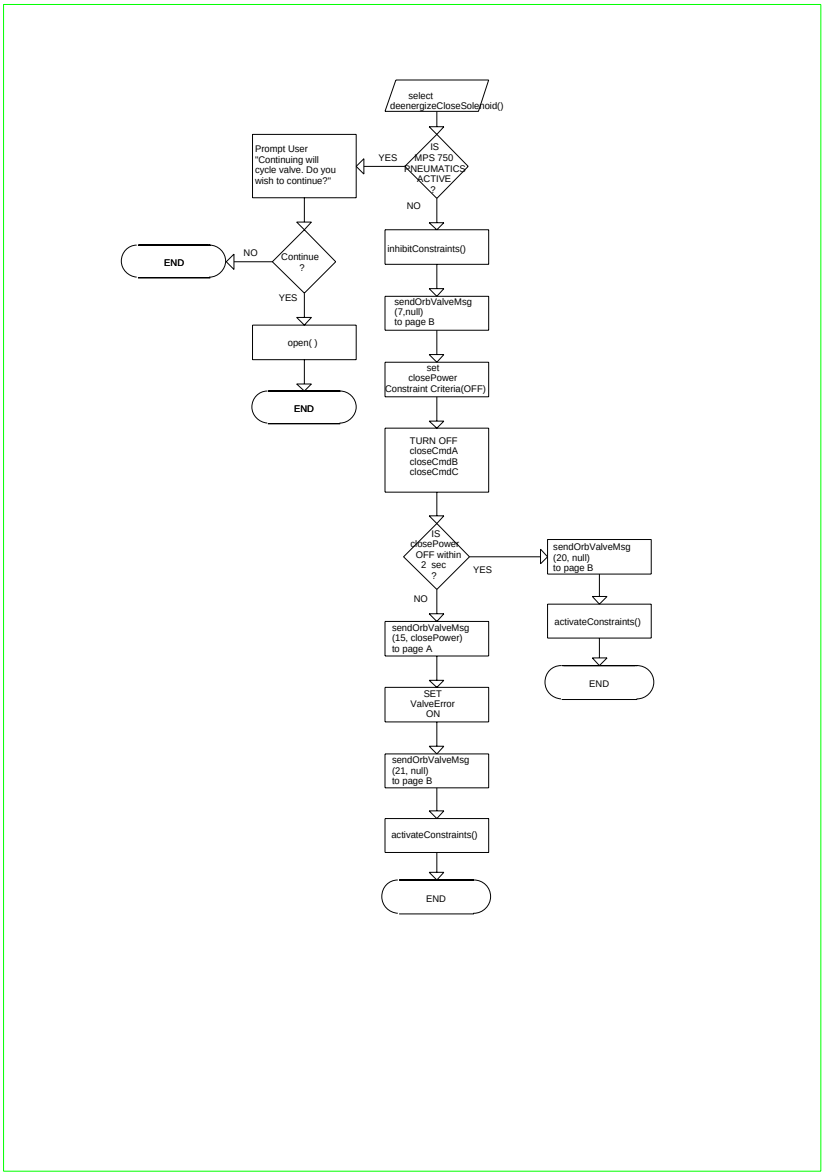
3.1.12.2.4. deenergizeCloseSolenoid()

3.1.12.2.4.1. Descriptive Text

This method performs removes power from the close solenoid. If MPS 750 pneumatics is active the open() method is called. The flow chart provides the necessary logic for the method, including checks for the proper response of power indicators. The valve cycles in a fail safe mode and 'wimps-out' (all commands removed) at the first indication of hardware failure

3.1.12.2.4.2. Sequence

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



3.1.12.2.5. determineCmdState()

3.1.12.2.5.1. Descriptive Text

This method uses the state of the close power indicator to determine the command state of the valve.

3.1.12.2.5.2. Sequence

Command states	
close power	**state**
OFF	cmd open
ON	cmd closed

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.12.2.6. determineIndState()

3.1.12.2.6.1. Descriptive Text

This method will apply "all of all unbypassed indicator" logic to determine the enumerated indicator state of the valve.

3.1.12.2.6.2. Sequence

determineIndState();

Indicator State

open pos ind	open pos byp	close pos ind	close pos byp	state
OFF	OFF	ON	OFF	closed
OFF	OFF	----	ON	closed
----	ON	ON	OFF	closed
----	ON	----	ON	error
ON	OFF	ON	OFF	error
ON	OFF	OFF	OFF	open
----	ON	OFF	OFF	open
ON	OFF	----	ON	open
OFF	OFF	OFF	OFF	transition

3.1.12.2.7. determineState()

3.1.12.2.7.1. Descriptive Text

This method uses the indicator state and the command state to determine the overall state of the valve.

3.1.12.2.7.2. Sequence

determineState();

State (overall)

pos \ cmd	cmd closed	cmd open
closed	closed	close error
open	open error	open
transition	transition	transition
error	error	error

3.1.12.2.8. getIndState()

3.1.12.2.8.1. Descriptive Text

This method will return the Enumerated Indicator State of the valve

3.1.12.2.8.2. Sequence

```
getIndState()
{
        return indState;
}
```

3.1.12.2.9. getCmdState()

3.1.12.2.9.1. Descriptive Text

This method will return the Enumerated Command State of the valve.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.12.2.9.2. Sequence

```
get cmdState()
        {
                return cmdState;
        }
```

3.1.12.2.10. getState()

3.1.12.2.10.1. Descriptive Text

This method will return the overall Enumerated State of the valve.

3.1.12.2.10.2. Sequence

```
getState()
        {
                return state;
        }
```

3.1.12.2.11. autoCycle()

3.1.12.2.11.1. Descriptive Text

The following will be called by the Panel Class autoValveCheckout method
to calculate and record the valve cycle times.

3.1.12.2.11.2. Sequence

Set autoValveCheckoutError = OFF;

valveCycleError = close();

if valveCycleError is ON, Set autoValveCheckoutError = ON;

valveCycleError = open();

if valveCycleError is ON, Set autoValveCheckoutError = ON;

Return autoValveCheckoutError;

3.1.12.2.12. activateConstraints()

3.1.12.2.12.1. Descriptive Text

This method will activate constraint monitoring of all unbypassed measurement of this valve.

3.1.12.2.12.2. Sequence

```
activateConstraints()
        {
        if(<openPosInd>.BYP == OFF)
                {
                activate constraint monitoring for <openPosInd>
                }
        if(<closePwr>.BYP == OFF)
                {
                activate constraint monitoring for <openPwr>
                }
        if(<closePosInd>.BYP == OFF)
                {
                activate constraint monitoring for <closePosInd>
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
        }
    }
```

3.1.12.2.13. inhibitConstraints()

3.1.12.2.13.1. Descriptive Text

This method will inhibit constraint monitoring of all measurements related to this valve.

3.1.12.2.13.2. Sequence

```
inhibitConstraints()
    {
    inhibit constraint monitoring of the following:
    <openPosInd>
    <closePosInd>
    <closePwr>
    }
```

3.1.12.2.14. setOpenConstraints()

3.1.12.2.14.1. Descriptive Text

This method changes the constraint criteria of the measurements associated with this valve to those expected with the valve in the open position.

3.1.12.2.14.2. Sequence

```
    void setOpenConstraints()
    {
        CHANGE <openPosInd> CONSTRAINT CRITERIA TO ON;

        CHANGE <closePwr>  CONSTRAINT CRITERIA TO OFF;

        CHANGE <closePosInd> CONSTRAINT CRITERIA TO OFF;

    }
```

3.1.12.2.15. setClosedConstraints()

3.1.12.2.15.1. Descriptive Text

This method changes the constraint criteria of the measurements associated with this valve to those expected with the valve in the closed position.

3.1.12.2.15.2. Sequence

```
    void setCloseConstraints()
    {
        CHANGE <openPosInd> CONSTRAINT CRITERIA TO OFF;

        CHANGE<closePwr>  CONSTRAINT CRITERIA TO ON;

        CHANGE (CL IND) CONSTRAINT CRITERIA TO ON;

    }
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.1.13. MPS_Valve_ORB_Recirc_Isolation

3.1.13.1. Description

3.1.13.1.1. Instance Index

Instance	Display User	Other Users
MPS_PV14	MPS	
MPS_PV15	MPS	
MPS_PV16	MPS	

3.1.13.1.2. Class Diagram

<insert class diagram here>

3.1.13.1.3. Attributes and Aggregations

Attribute	Data Type	Initial Value	Publish / Persistence	Client Access	Description
openInd	DM_ON_OFF				recirc valve open indicator
closeInd	DM_ON_OFF				recirc valve close indicator
indState	MPS_VALVE_IND_STATE		PUB		recirc valve enumerated indicator state based on both position indicators and associated bypasses
state	MPS_ORB_VALVE_STATE		PUB		recirc valve enumerated state based on cmdState and indState

3.1.13.1.4. Operations Index

Operation	Description
monitorValveOpening(GMT_SIG)	This method is initiated by MPS_Recirc::open() method, and monitors the position indicators. The method 'wimps out' at the first error unless the 'CRYOFLAG' is ON.
monitorValveClosing(GMT_SIG)	This method is initiated by MPS_Recirc::close() method, and monitors the position indicators. The method 'wimps out' at the first error unless the 'CRYOFLAG' is ON.
setOpenConstraints()	This method sets the constraint criteria to that associated with the open position.
setCloseConstraints()	This method sets the constraint criteria to that associated with the closed position.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

determineIndState()	This method determines the enumerated indicator state of the recirc valves.
determineState	This method determines the enumerated overall state of the recirc valve.
getIndState()	This method returns the enumerated indicator state.
inhibitConstraints()	This method inhibits constraint monitoring of the position indicators associated with the valve.
activateConstraints()	This method activates monitoring of the constraint criteria associated with this valve.

3.1.13.1.5. Associations

Association	Type	Class	Description

3.1.13.2. State Description

3.1.13.2.1. State Diagram

<insert state diagram here>

3.1.13.2.2. State Table

3.1.13.3. Operations

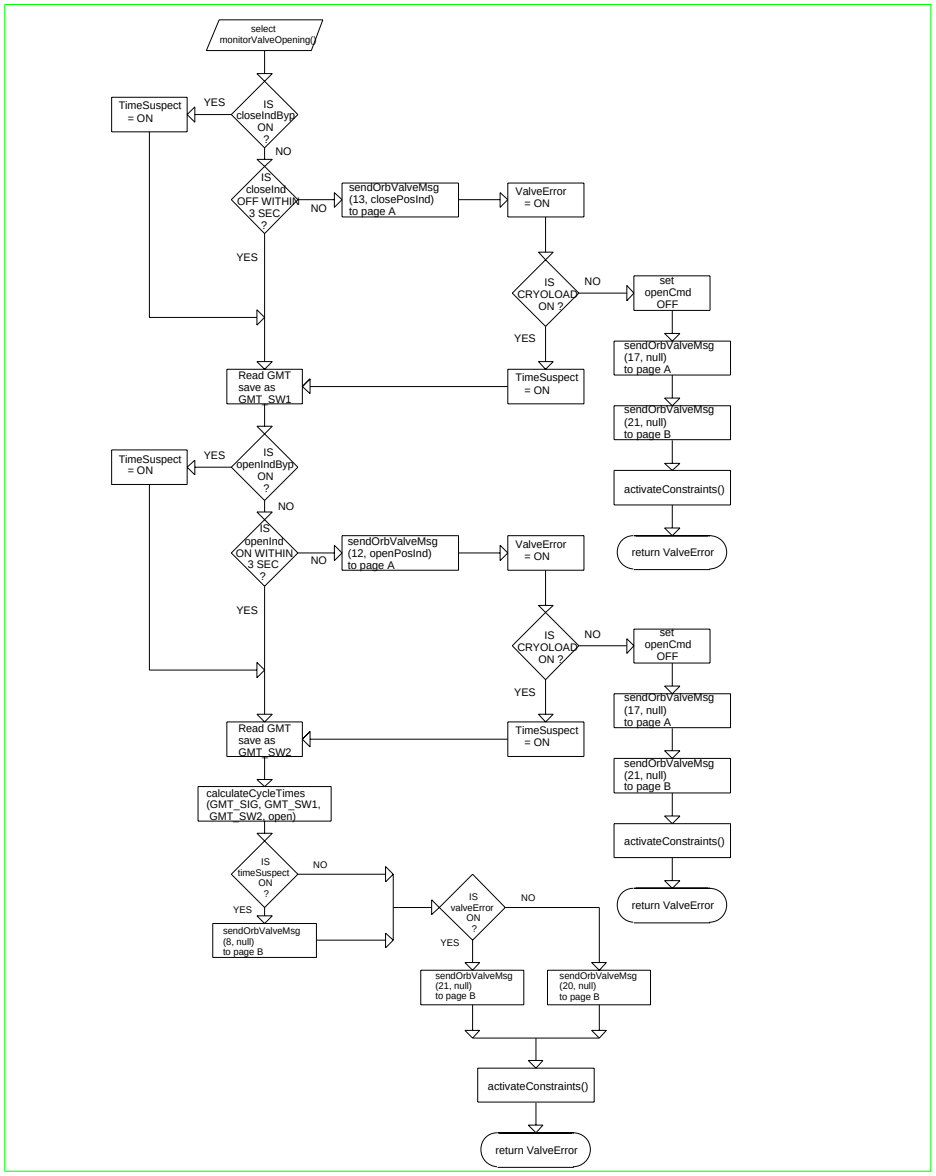
3.1.13.3.1. monitorValveOpening(GMT_SIG)

3.1.13.3.1.1. Descriptive Text

This method verifies the cycling of the position indicators associated with Recirc Isolation valve opening. This method is called by the MPS_Recirc::open() method and is run concurrently for each of the three MPS_RecircIsolation instances. The flow charts contained below describe the logic to be used to code the method. The implementation of the details is left up to the coders and is acceptable as long as the overall logic is maintained. It is expected that cycle timing will be accomplished using the most accurate source. All messages will include a time stamp, and the appropriate FD description.

3.1.13.3.1.2. Sequence

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



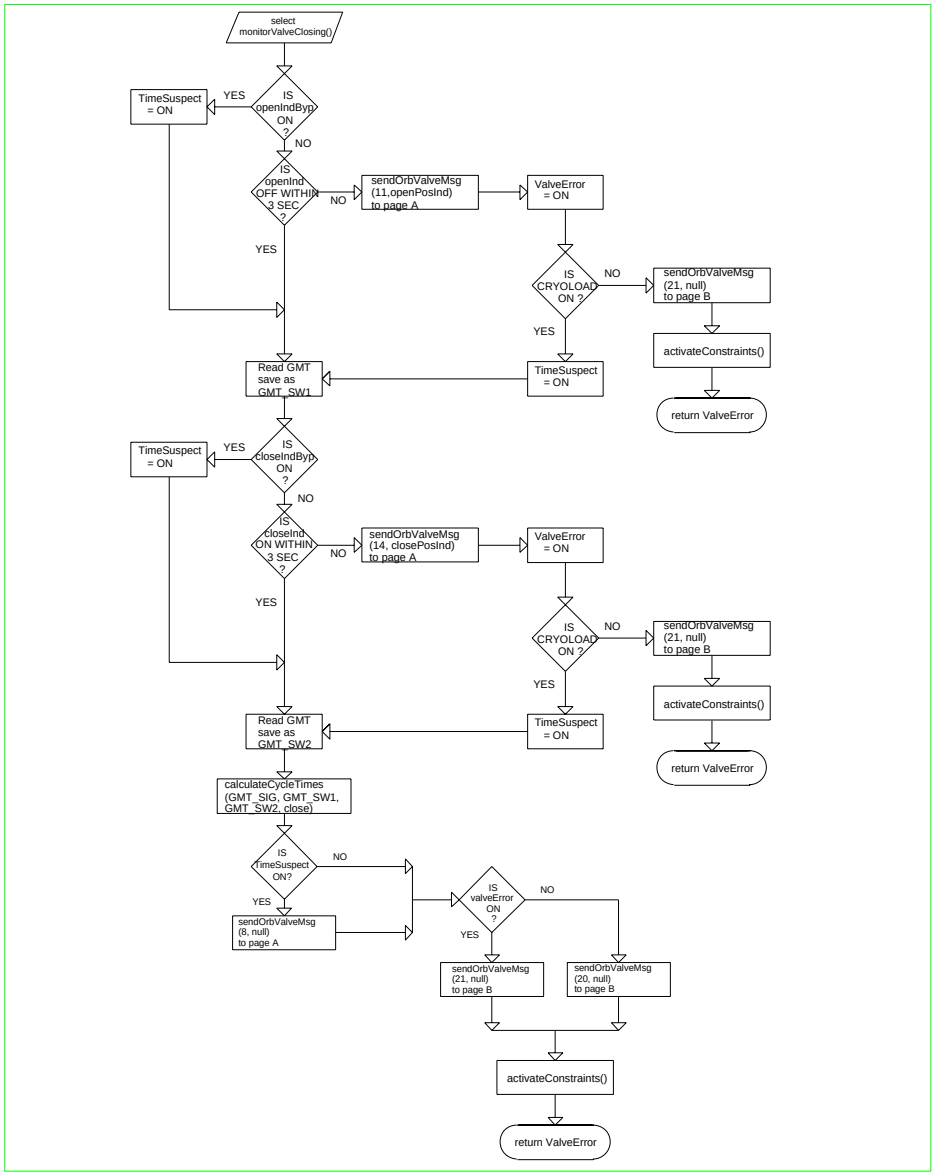
3.1.13.3.2. monitorValveClosing(GMT_SIG)

3.1.13.3.2.1. Descriptive Text

This method verifies the cycling of the position indicators associated with Recirc Isolation valve opening. This method is called by the MPS_Recirc::open() method and is run concurrently for each of the three MPS_RecircIsolation instances. The flow charts contained below describe the logic to be used to code the method. The implementation of the details is left up to the coders and is acceptable as long as the overall logic is maintained. It is expected that cycle timing will be accomplished using the most accurate source. All messages will include a time stamp, and the appropriate FD description.

3.1.13.3.2.2. Sequence

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



3.1.13.3.3. determineIndState()

3.1.13.3.3.1. Descriptive Text

This method will determine the valve's commanded state based on the valve open and close position indicators. The return value will be an enumerated type with possible values of OPEN, CLOSED, TRANSITION and ERROR.

3.1.13.3.3.2. Effectivity

Operates on change data.

3.1.13.3.3.3. Sequence

Case:

<OP POS IND> OFF AND
<OP POS IND>.BYP OFF AND

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
  <CL POS IND>          OFF AND
  <CL POS IND>.BYP    OFF :     indState = TRANSITION;

  <OP POS IND>          OFF AND
  <OP POS IND>.BYP    OFF AND
  <CL POS IND>          ON  AND
  <CL POS IND>.BYP    OFF :     indState = CLOSED;

  <OP POS IND>.BYP    ON  AND
  <CL POS IND>          ON  AND
  <CL POS IND>.BYP    OFF :     indState = CLOSED;

  <OP POS IND>          OFF AND
  <OP POS IND>.BYP    OFF AND
  <CL POS IND>.BYP    ON  :     indState = CLOSED;

  <OP POS IND>          ON  AND
  <OP POS IND>.BYP   OFF AND
  <CL POS IND>          OFF AND
  <CL POS IND>.BYP    OFF :     indState = OPEN;

  <OP POS IND>.BYP    ON  AND
  <CL POS IND>          OFF AND
  <CL POS IND>.BYP    OFF :     indState = OPEN;

   <OP POS IND>         ON  AND
  <OP POS IND>.BYP    OFF AND
  <CL POS IND>.BYP    ON  :     indState = OPEN;

  <OP POS IND>          ON  AND
  <OP POS IND>.BYP   OFF AND
  <CL POS IND>          ON  AND
  <CL POS IND>.BYP    OFF :     indState = ERROR;

  <OP POS IND>.BYP    ON AND
  <CL POS IND>.BYP    ON :      indState = ERROR;

  Default: indState = ERROR;

End Case:
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

Indicator State				
open pos ind	**open pos byp**	**close pos ind**	**close pos byp**	**state**
OFF	OFF	ON	OFF	closed
OFF	OFF	----	ON	closed
----	ON	ON	OFF	closed
ON	OFF	OFF	OFF	open
----	ON	OFF	OFF	open
ON	OFF	----	ON	open
OFF	OFF	OFF	OFF	transition
----	ON	----	ON	error
ON	OFF	ON	OFF	error

3.1.13.3.4. getIndState()

3.1.13.3.4.1. Descriptive Text

This method will return the Enumerated Command State of the valve.

3.1.13.3.4.2. Sequence

```
getIndState()
{
        return IndState;
}
```

3.1.13.3.5. determineState()

3.1.13.3.5.1. Descriptive Text

This method compares the commanded state and indicated state of the valve and assigns an overall state to the valve. All three instances of this valve are controlled by a single pneumatic actuator. The commanded state for each of the valve instances is inherited from the MPS_Recirc class and is the same for each instance. The indicated state of each instance is unique for each instance.

3.1.13.3.5.2. Sequence

State (overall)		
pos \ cmd	**cmd closed**	**cmd open**
closed	closed	close error
open	open error	open
transition	transition	transition
error	error	error

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.13.4. User Interfaces

3.1.13.5. Safety Requirements

3.1.14. MPS_Valve_ORB_Relief_Isolation

3.1.14.1. Description

3.1.14.1.1. Instance Index

Instance	Display User	Other Users
MPS_PV7	MPL	
MPS_PV8	MPS	

3.1.14.1.2. Class Diagram

<insert class diagram here>

3.1.14.1.3. Attributes and Aggregations

Attribute	Data Type	Initial Value	Publish / Persistence	Client Access	Description
closeCmdA	DS_ON_OFF				
closeCmdB	DS_ON_OFF				
closeCmdC	DS_ON_OFF				
openPositionInd	DM_ON_OFF				
closePositionInd	DM_ON_OFF				
closePower	DM_ON_OFF				
closeRPC1	DM_ON_OFF				
closeRPC2	DM_ON_OFF				
cmdState	ENUM		PUB		
indState	ENUM		PUB		
state	ENUM		PUB		

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.14.1.4. Associations

Association	Type	Class	Description

3.1.14.1.5. Operations Index

Operation	Description
open()	This method performs the operations necessary to open the valve.
close()	This mthod performs the operations necessary to close the valve.
determineIndState()	This method looks at the position indicators of the valve and returns an enumerated state based on a logic table.
determineCmdState()	This method returns an enumerated state based on valve power and RPC indicators.
determineState()	This method looks at the cmdState and indState pseudo-FDs of the valve and returns an enumerated state based on a logic table.
getIndState()	returns the enumerated indicator state.
getCmdState()	returns the enumerated command state.
getState()	returns the enumerated overall state.
inhibitConstraints()	This method inhibits constraint monitoring of all values associated with this valve.
activateConstraints()	This method activates constraint monitoring of all unbypassed values associated with this valve.
autoCycle()	This method runs the open and close methods of the valve and returns either failure or success.
setOpenConstraints()	This method establishes the constraint criteria expected for the valve in the open position.
setCloseConstraints()	This method establishes the constraint criteria expected for the valve in the closed position.
verifyCmdsOff()	This method checks to verify all commands are off.
//verifyTminus6BT4//	This method verifies T-6 hour configuration of commands (S/B OFF).
//verifyTminus3BT4//	This method verifies T-3 hour configuration of commands (S/B OFF).

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

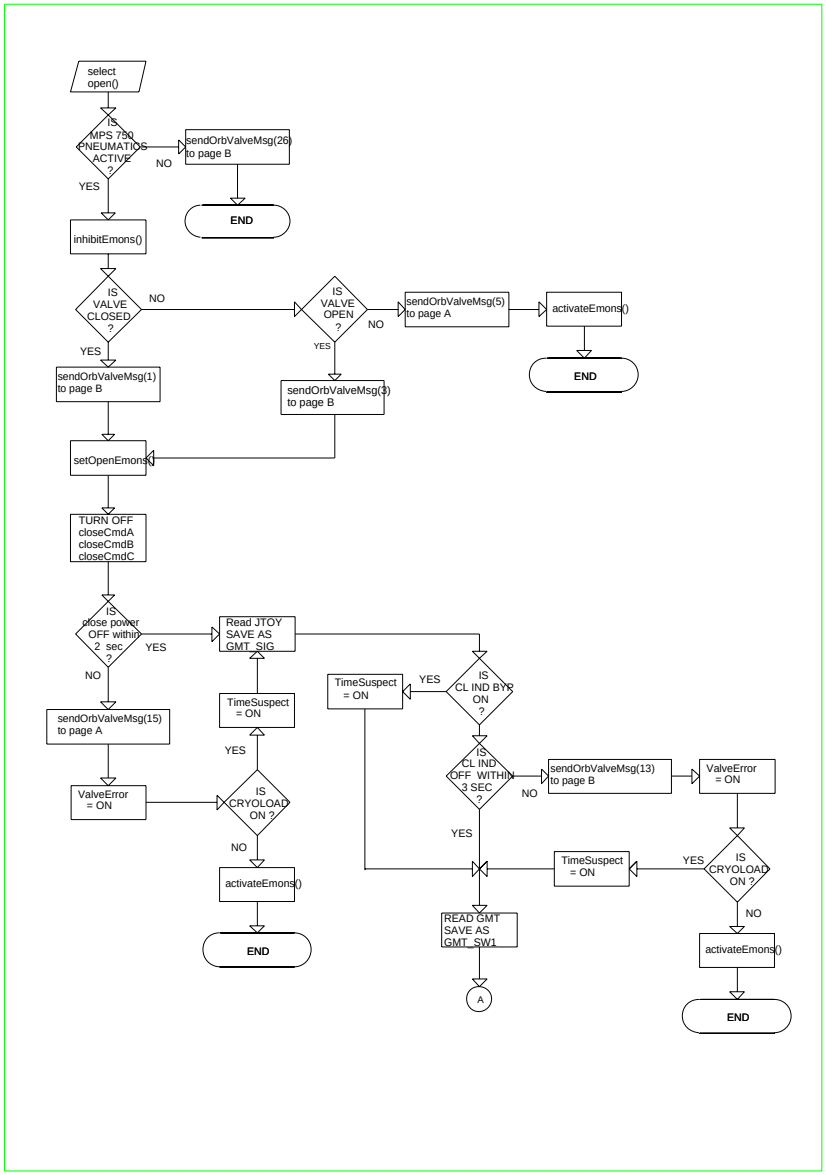
3.1.14.2. Operations

3.1.14.2.1. open()

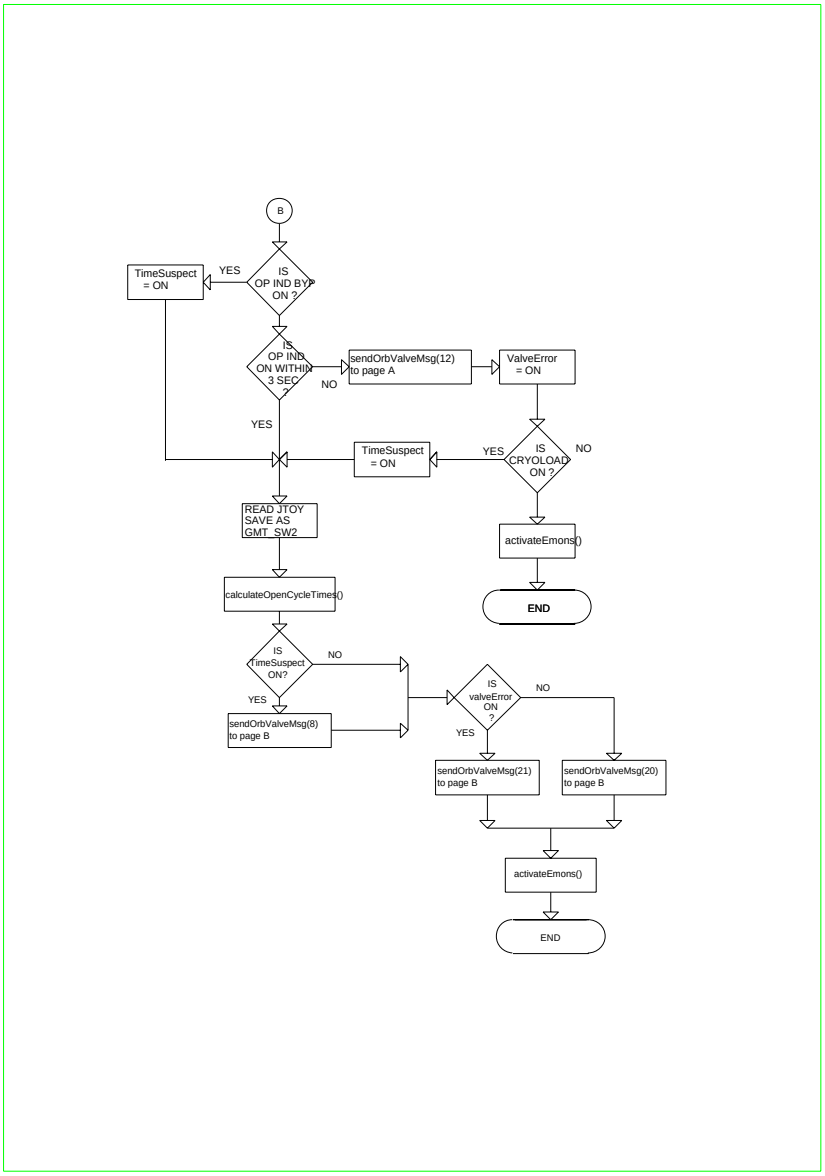
3.1.14.2.1.1. Descriptive Text

The flow charts contained below describe the logic to be used to code the method. The implementation of the details is left up to the coders and is acceptable as long as the overall logic is maintained. It is expected that cycle timing will be accomplished using the most accurate source. All messages will include a time stamp, and the appropriate FD description.

3.1.14.2.1.2. Sequence



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



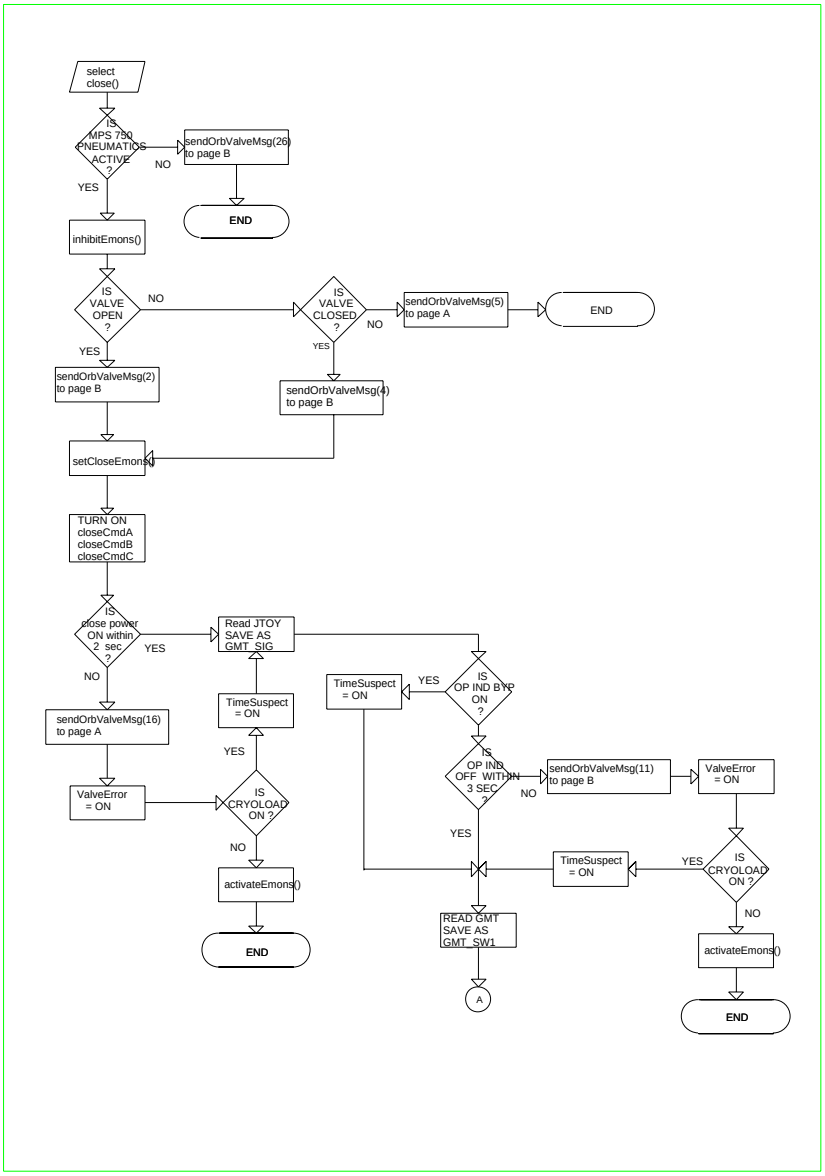
3.1.14.2.2. close()

3.1.14.2.2.1. Descriptive Text

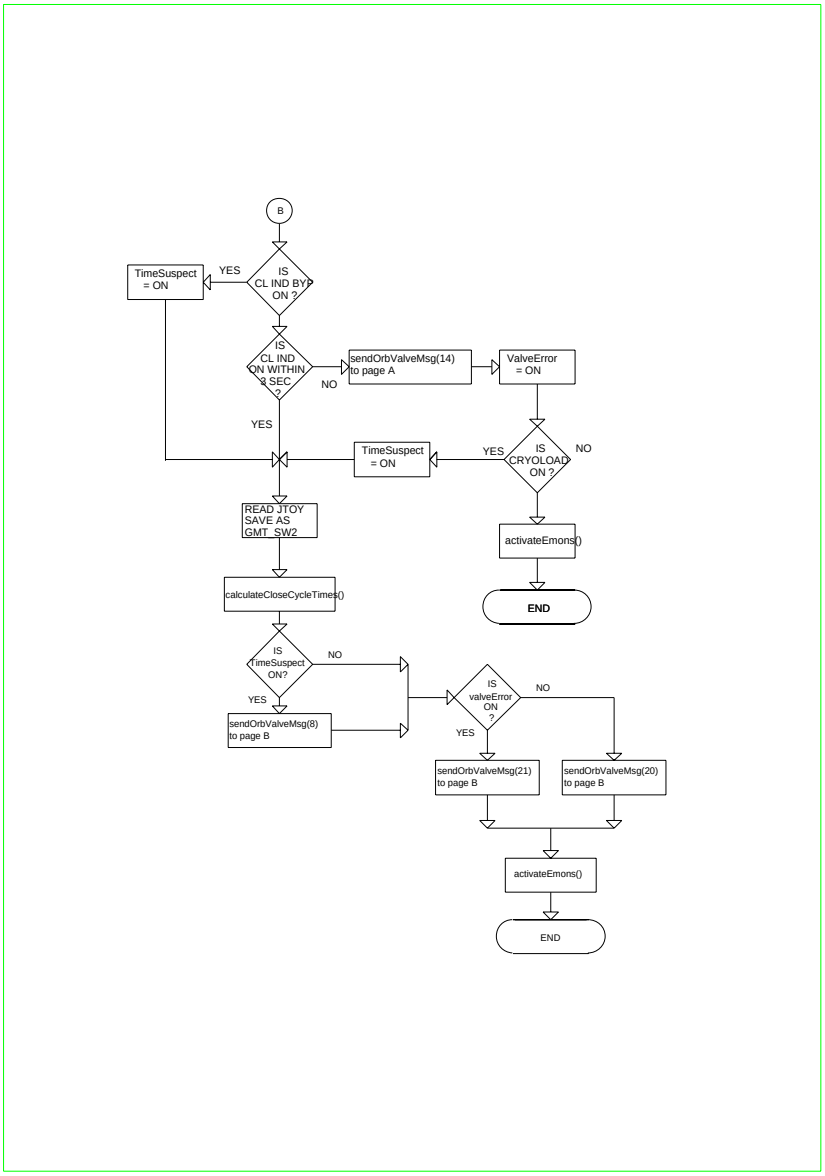
The flow charts contained below describe the logic to be used to code the method. The implementation of the details is left up to the coders and is acceptable as long as the overall logic is maintained. It is expected that cycle timing will be accomplished using the most accurate source. All messages will include a time stamp, and the appropriate FD description.

3.1.14.2.2.2. Sequence

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



3.1.14.2.3. determineCmdState()

3.1.14.2.3.1. Descriptive Text

This method will use the "All of all unbypassed indicator" logic to determine the command state using the two close RPCs and the close power indicator.

3.1.14.2.3.2. Sequence

determineCmdState()

{

CASE:

<closePwrByp> == ON

<closeRPCAByp> == ON

<closeIRPCB> == ON

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

<closeRPCBByp> == OFF
<cmdState> = CMD_CLOSE

CASE:
<closePwrByp> == ON
<closeRPCA> == ON
<closeRPCAByp> == OFF
<closeRPCBByp> == ON
<cmdState> = CMD_CLOSE

CASE:
<closePwrByp> == ON
<closeRPCA> == ON
<closeRPCAByp> == OFF
<closeRPCBByp> == ON
<cmdState>= CMD_CLOSE

CASE:
<closePwr> == ON
<closePwrByp> == OFF
<closeRPCAByp> == ON
<closeRPCBByp> == ON
<cmdState> = CMD_CLOSE

CASE:
<closePwr> == ON
<closePwrByp> == OFF
<closeRPCAByp> == ON
<closeIRPCB> == ON
<closeRPCBByp> == OFF
<cmdState> = CMD_CLOSE

CASE:
<closePwr> == ON
<closePwrByp> == OFF
<closeRPCA> == ON
<closeRPCAByp> == OFF
<closeRPCBByp> == ON
<cmdState> = CMD_CLOSE

CASE:
<closePwr> == ON
<closePwrByp> == OFF
<closeRPCA> == ON
<closeRPCAByp> == OFF
<closeRPCB> == ON
<closeRPCBByp> == OFF
<cmdState> = CMD_CLOSE

CASE:
<closePwr> == OFF
<closePwrByp> == OFF
<closeRPCA> == OFF
<closeRPCAByp> == OFF

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

<closeRPCB> == OFF
<closeRPCBByp> == OFF
<cmdState> = CMD_OPEN

CASE:
<closePwr> == OFF
<closePwrByp> == OFF
<closeRPCA> == OFF
<closeRPCAByp> == OFF
<closeRPCBByp> == ON
<cmdState> = CMD_OPEN

CASE:
<closePwr> == OFF
<closePwrByp> == OFF
<closeRPCAByp> == ON
<closeRPCB> == OFF
<closeRPCBByp> == OFF
<cmdState> = CMD_OPEN

CASE:
<closePwr> == OFF
<closePwrByp> == OFF
<closeRPCAByp> == ON
<closeRPCBByp> == ON
<cmdState> = CMD_OPEN

CASE:

<closePwrByp> == ON
<closeRPCA> == OFF
<closeRPCAByp> == OFF
<closeRPCB> == OFF
<closeRPCBByp> == OFF
<cmdState> = CMD_OPEN

CASE:
<closePwrByp> == ON
<closeRPCA> == OFF
<closeRPCAByp> == OFF
<closeRPCBByp> == ON
<cmdState> = CMD_OPEN
CASE:
<closePwrByp> == ON
<closeRPCAByp> == ON
<closeRPCB> == OFF
<closeRPCBByp> == OFF
<cmdState> = CMD_OPEN

CASE ELSE
<cmdState> = ERROR

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

Command State

close power	close power bypass	close RPC_A	close RPC_A Bypass	close RPC_B	close RPC_B Bypass	command State
----	ON	----	ON	ON	OFF	cmd_close
----	ON	ON	OFF	----	ON	cmd_close
----	ON	ON	OFF	ON	OFF	cmd_close
ON	OFF	----	ON	----	ON	cmd_close
ON	OFF	----	ON	ON	OFF	cmd_close
ON	OFF	ON	OFF	----	ON	cmd_close
ON	OFF	ON	OFF	ON	OFF	cmd_close
OFF	OFF	OFF	OFF	OFF	OFF	cmd_open
OFF	OFF	OFF	OFF	----	ON	cmd_open
OFF	OFF	----	ON	OFF	OFF	cmd_open
OFF	OFF	----	ON	----	ON	cmd_open
----	ON	OFF	OFF	OFF	OFF	cmd_open
----	ON	OFF	OFF	----	ON	cmd_open
----	ON	----	ON	OFF	OFF	cmd_open
OFF	OFF	OFF	OFF	ON	OFF	error
OFF	OFF	----	ON	ON	OFF	error
OFF	OFF	ON	OFF	OFF	OFF	error
OFF	OFF	ON	OFF	----	ON	error
OFF	OFF	ON	OFF	ON	OFF	error
----	ON	OFF	OFF	ON	OFF	error
----	ON	----	ON	----	ON	error
----	ON	ON	OFF	OFF	OFF	error
ON	OFF	OFF	OFF	OFF	OFF	error
ON	OFF	OFF	OFF	----	ON	error
ON	OFF	OFF	OFF	ON	OFF	error
ON	OFF	----	ON	OFF	OFF	error
ON	OFF	ON	OFF	OFF	OFF	error

3.1.14.2.4. determineIndState()

3.1.14.2.4.1. Descriptive Text

This method will apply "All of all unbypassed indicator" logic to determine the enumerated indicator state of the valve.

3.1.14.2.4.2. Sequence

```
determineIndState()
        {
        CASE:
                <openPosInd> == OFF
                <openPosIndByp> == OFF
                <closePosInd> == ON
                <openPosIndByp> == OFF
                                        IndState = CLOSED;

        CASE:
                <openPosInd> == OFF
                <openPosIndByp> == OFF
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

<openPosIndByp> == ON
IndState = CLOSED;

CASE:
<openPosIndByp> == ON
<closePosInd> == ON
<openPosIndByp> == OFF
IndState = CLOSED;

CASE:
<openPosInd> == ON
<openPosIndByp> == OFF
<closePosInd> == OFF
<openPosIndByp> == OFF
IndState = OPEN;

CASE:
<openPosIndByp> == ON
<closePosInd> == OFF
<openPosIndByp> == OFF
IndState = OPEN;

CASE:
<openPosInd> == ON
<openPosIndByp> == OFF
<openPosIndByp> == ON
IndState = OPEN;

CASE:
<openPosIndByp> == ON
<openPosIndByp> == ON
IndState = ERROR;

CASE:
<openPosInd> == ON
<openPosIndByp> == OFF
<closePosInd> == ON
<openPosIndByp> == OFF
IndState =ERROR;

CASE:
<openPosInd> == OFF
<openPosIndByp> == OFF
<closePosInd> == OFF
<openPosIndByp> == OFF
IndState = TRANSITION;

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

Indicator State

open pos ind	open pos byp	close pos ind	close pos byp	state
OFF	OFF	ON	OFF	closed
OFF	OFF	----	ON	closed
----	ON	ON	OFF	closed
----	ON	----	ON	error
ON	OFF	ON	OFF	error
ON	OFF	OFF	OFF	open
----	ON	OFF	OFF	open
ON	OFF	----	ON	open
OFF	OFF	OFF	OFF	transition

3.1.14.2.5. determineState()

3.1.14.2.5.1. Descriptive Text

This method uses the indicator state and the command state to determine the overall state of the valve.

3.1.14.2.5.2. Sequence

```
determineState()
        {
        case:

                (<cmdState> ==  CMD_CLOSED  AND    <indState> == closed)  : state=closed;

                (<cmdState> == CMD_CLOSED AND              <indState> == open)    : state=openError;

                (<cmdState> == CMD_CLOSED AND              <indState> == TRANSITION) : state =
TRANSITION;

                (<cmdState> == CMD_CLOSED AND              <indState> == error)       : state = error;

                (<cmdState> ==  CMD_OPEN  AND              <indState> == closed)  : state=closedError;

                (<cmdState> == CMD_OPEN AND                <indState> == open)    : state=open;

                (<cmdState> == CMD_OPEN AND                <indState> == TRANSITION) : state =
TRANSITION;

                (<cmdState> == CMD_OPEN AND                <indState> == error)       : state = error;

        End Case;
        }
```

State (overall)

pos \ cmd	cmd closed	cmd open
closed	closed	close error
open	open error	open
transition	transition	transition
error	error	error

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.1.14.2.6. getIndState()

3.1.14.2.6.1. Descriptive Text

This method will return the Enumerated Indicator State of the valve

3.1.14.2.6.2. Sequence

```
getIndState()
{
        return indState;
}
```

3.1.14.2.7. getCmdState()

3.1.14.2.7.1. Descriptive Text

This method will return the Enumerated Command State of the valve.

3.1.14.2.7.2. Sequence

```
get cmdState()
        {
                return cmdState;
        }
```

3.1.14.2.8. getState()

3.1.14.2.8.1. Descriptive Text

This method will return the overall Enumerated State of the valve.

3.1.14.2.8.2. Sequence

```
getState()
        {
                return state;
        }
```

3.1.14.2.9. autoCycle()

3.1.14.2.9.1. Descriptive Text

The following will be called by the Panel Class autoValveCheckout method to calculate and record the valve cycle times.

3.1.14.2.9.2. Sequence

Set autoValveCheckoutError = OFF;

valveCycleError = close();

if valveCycleError is ON, Set autoValveCheckoutError = ON;

valveCycleError = open();

if valveCycleError is ON, Set autoValveCheckoutError = ON;

Return autoValveCheckoutError;

3.1.14.2.10. activateConstraints()

3.1.14.2.10.1. Descriptive Text

This method will activate constraint monitoring of all unbypassed measurement of this valve.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.14.2.10.2. Sequence

activateConstraints()
{
if(<openPosInd>.BYP == OFF)
{
activate constraint monitoring for <openPosInd>
}
if(<closePwr>.BYP == OFF)
{
activate constraint monitoring for <closePwr>
}
if(<closePosInd>.BYP == OFF)
{
activate constraint monitoring for <closePosInd>
}
if(<closeRPC1>.BYP == OFF)
{
activate constraint monitoring for <closeRPC1>
}
if(<closeRPC2>.BYP == OFF)
{
activate constraint monitoring for <closeRPC2>
}
}

3.1.14.2.11. inhibitConstraints()

3.1.14.2.11.1. Descriptive Text

This method will inhibit constraint monitoring of all measurements related to this valve.

3.1.14.2.11.2. Sequence

inhibitConstraints()
{
inhibit constraint monitoring of the following:
<openPosInd>
<closePosInd>
<closePwr>
<closeRPC1>
<closeRPC2>
}

3.1.14.2.12. setOpenConstraints()

3.1.14.2.12.1. Descriptive Text

This method changes the constraint criteria of the measurements associated with this valve to those expected with the valve in the open position.

3.1.14.2.12.2. Sequence

void setOpenConstraints()
{
CHANGE <openPosInd> CONSTRAINT CRITERIA TO ON;

CHANGE <closePwr> CONSTRAINT CRITERIA TO OFF;

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

CHANGE <closePosInd> CONSTRAINT CRITERIA TO OFF;

CHANGE <closeRPC1> CONSTRAINT CRITERIA TO OFF;

CHANGE <closeRPC2> CONSTRAINT CRITERIA TO OFF;

}

3.1.14.2.13. setClosedConstraints()

3.1.14.2.13.1. Descriptive Text

This method changes the constraint criteria of the measurements associated with this valve to those expected with the valve in the closed position.

3.1.14.2.13.2. Sequence

void setCloseConstraints()
{
CHANGE <openPosInd> CONSTRAINT CRITERIA TO OFF;

CHANGE<closePwr> CONSTRAINT CRITERIA TO ON;

CHANGE (CL IND) CONSTRAINT CRITERIA TO ON;

CHANGE <closeRPC1> CONSTRAINT CRITERIA TO ON;

CHANGE <closeRPC2> CONSTRAINT CRITERIA TO ON;

}

3.1.15. MPS_Valve_ORB_17inch_Disc

3.1.15.1. Description

MPS_Valve_ORB_17inch_Disc is a bi-stable valve with 3 open and 3 close commands, 2 open and 2 close position indications, 1 open and 1 close power indications, 2 open and 2 close RPCs. The valve actuator solenoids (1 each open and close) are each redundantly commanded/powered. There are 2 instances of this valve, one for each orbiter 17" feedline interface to the external tank (MPS_PD1 in the MPS_LO2 Subsystem and MPS_PD2 in the MPS_LH2_Subsystem). Functionally, the valve serves to isolate the orbiter from the external tank, both during post meco/ET separation operations and in contingency leak isolation procedures. The actuation design of the valve requires the ORB and ET to be mated to cycle the component. In an unmated configuration, the orbiter actuator arm will stroke, but no valve cycle will result.

The valve class inherits from the MPS_Valve_ORB Class. The MPS_Valve_ORB_17inch_Disc, along with the MPS_Valve_ORB_Latch comprise the components of the MPS_ORB_17inch_Disconnect_Assembly Class. The disconnect and latch are integrally related in function, and cycling of the disconnect requires the latch to be in the unlocked position. This requirement is checked both in the open/close methods of the disconnect assembly class and by onboard TCS. The valve also has an association with the MPS_Helium_SubSystem Class for status of the MPS 750 pneumatic system. For integrated operations, this valve class has an association with the MPS_SubSystem Class for determination of the cryo load stautus via a Cryo Flag pseudo. During test and check-out mode, valve cycles will break-out and remove all commands/power when a failure is detected. When the Cyro Flag is ON, the valve cycle sequence will continue to completion regardless of failures.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.1.15.1.1. Instance Index

Instance	Display User	Other Users
MPS_PD1	MPL	
MPS_PD2	MPS	

3.1.15.1.2. Class Diagram

<insert class diagram here>

3.1.15.1.3. Attributes and Aggregations

Attribute	Data Type	Initial Value	Publish / Persistence	Client Access	Description
openPosIndA	DM_ON_OFF			R	
openPosIndB	DM_ON_OFF			R	
openCmdA	DS_ON_OFF			R	
openCmdB	DS_ON_OFF			R	
openCmdC	DS_ON_OFF			R	
openRPCB	DM_ON_OFF			R	
openRPCC	DM_ON_OFF			R	
openPwr	DM_ON_OFF			R	
openSolCmdState	MPS_ORB_SOL_STATE ENUM		PUB	R	
closePosIndA	DM_ON_OFF			R	
closePosIndB	DM_ON_OFF			R	
closeCmdA	DS_ON_OFF			R	
closeCmdB	DS_ON_OFF			R	
closeCmdC	DS_ON_OFF			R	
closeRPCB	DM_ON_OFF			R	
closeRPCC	DM_ON_OFF			R	
closePwr	DM_ON_OFF			R	
closeSolCmdState	MPS_ORB_SOL_STATE ENUM		PUB	R	

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

indState	MPS_VALVE_IND_STATE ENUM		PUB	R	
valveCycleError					//do we need this?//
cycleTimeSuspect					//do we need this?//

3.1.15.1.4. Operations Index

Operation	**Description**
open( )	This method performs necessary functions to cycle valve to the open position.
close ( )	This method performs necessary functions to cycle valve to the closed position.
energizeOpenSolenoid( )	This method energizes the open solenoid . Method includes a check of MPS pneumatics status, and cautions operator of snubbing requirements if MPS pneumatics is active.
energizeCloseSolenoid( )	This method energizes closing solenoid. Method includes a check of MPS pneumatics status, and cautions operator of snubbing requirements if MPS pneumatics is active.
deenergizeOpenSolenoid( )	This method deenergizes the open solenoid . Method includes a check of MPS pneumatics status, and cautions operator of snubbing requirements if MPS pneumatics is active.
deenergizeCloseSolenoid( )	This method energizes the closing solenoid . Method includes a check of MPS pneumatics status, and cautions operator of snubbing requirements if MPS pneumatics is active.
openON_check()	This method verifies that the open powers and RPCs are energized, and provides appropriate messages if they are not. The method returns valveError equal to ON if an unbypassed power or RPC does not achieve the correct state within the alloted time.
openOff_check()	This method verifies that the open powers and RPCs are deenergized, and provides appropriate messages if they are not. The method returns valveError equal to ON if an unbypassed power or RPC does not achieve the correct state within the alloted time.
closeON_check()	This method verifies that the close powers and RPCs are energized, and provides appropriate messages if they are not. The method returns valveError equal to ON if an unbypassed power or RPC does not achieve the correct state within the alloted time.
closeOff_check()	This method verifies that the close powers and RPCs are deenergized, and provides appropriate messages if they are not. The method returns valveError equal to ON if an unbypassed power or RPC does not

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

	achieve the correct state within the alloted time.
determineState( )	This method will return valve Enumerated combined overall state and return: COMMANDED_OPEN COMMANDED_CLOSED UNCOMMANDED COMMANDED_SNUBBED ERROR
determineIndState( )	This method will determine the position state of the valve based on open/close indicators and will return either: OPEN CLOSED TRANSITION ERROR
determineOpenSolCmdState( )	This method will determine the command state of the opening solenoid valve based on valve RPC and power indicators and will return either: ENERGIZED DEENERGIZED ERROR
determineCloseSolCmdState( )	This method will determine the command state of the closing solenoid valve based on solenoid RPC and power indicators and will return either: ENERGIZED DEENERGIZED ERROR
getState( )	This method will return valve Enumerated combined overall state.
getIndState( )	This method will return valve indicator state.
getOpenSolCmdState( )	This method will return opening solenoid command state.
getCloseSolCmdState( )	This method will return closing solenoid command state.
verifyCmdsOff( )	This method will verify that no commands associated with this valve are on.
setOpenConstraints( )	This method establishes Constraint configuration for an open valve.
setCloseConstraints( )	This method establishes Constraint configuration for a closed valve.
setCloseIndConstraints()	This method establishes Constraint configuration for a closed bistable valve.
activateConstraints( )	This method will activate constraint criteria monitoring of all measurements associated with this valve and its associated solenoids.
inhibitConstraints( )	This method will inhibit constraint criteria monitoring of all measurements associated with this valve and its associated solenoids.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

sendDisconnectMsg(int)	This method provides disconnect specific messages for the valve cycling methods.

3.1.15.1.5. Associations

Association	Type	Class	Description
MPS_SubSystem			CryoFlag status.
MPS_Helium_SubSystem			MPS 750 pneumatic status.

3.1.15.2. State Description

3.1.15.2.1. State Diagram

<insert state diagram here>

3.1.15.2.2. State Table

3.1.15.3. Operations

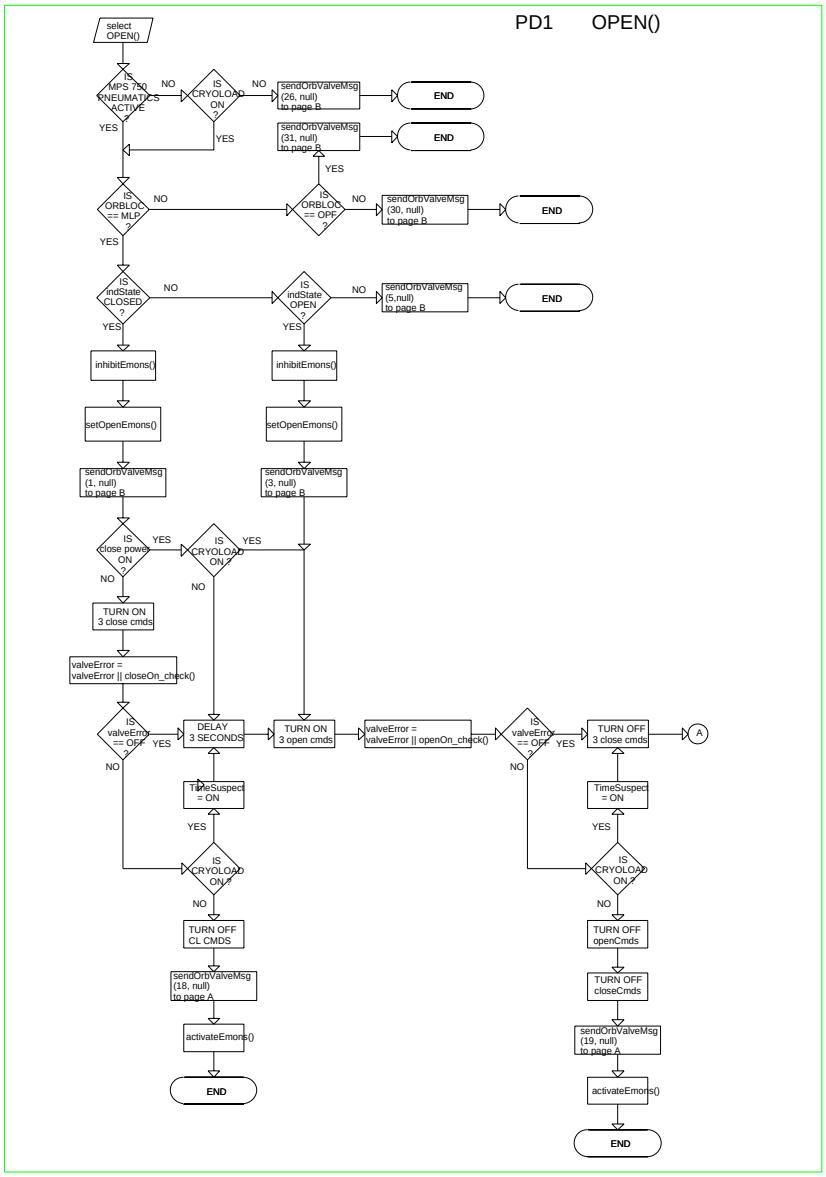
3.1.15.3.1. open()

3.1.15.3.1.1. Descriptive Text

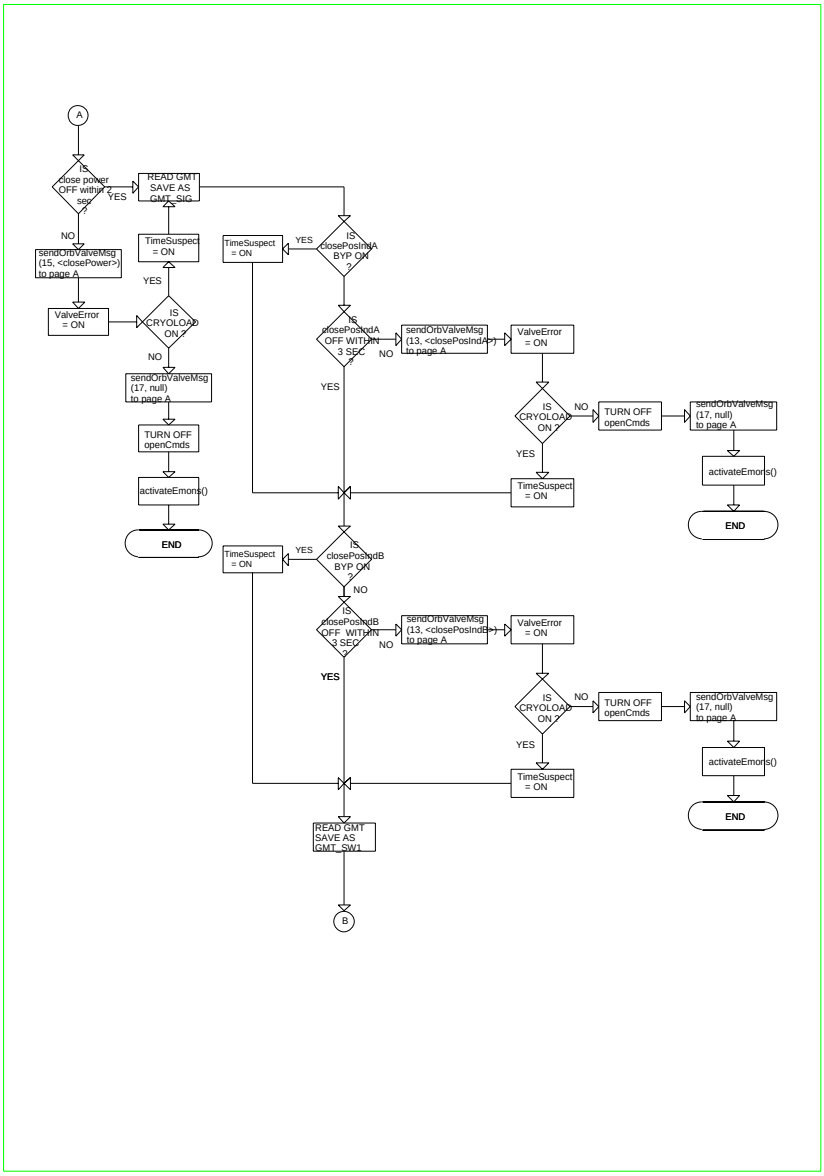
The flow charts contained below describe the logic to be used to code the method. The implementation of the details is left up to the coders and is acceptable as long as the overall logic is maintained. It is expected that cycle timing will be accomplished using the most accurate source. All messages will include a time stamp, and the appropriate FD description. ValveError and TimeSuspect are initialized to OFF and Valve cycle times are set to 0 prior to starting valve cycle methods. If ValveError gets set ON during method run, cycle time FD data health should be set to invalid to aid in identifying suspect cycle data in trend analysis.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

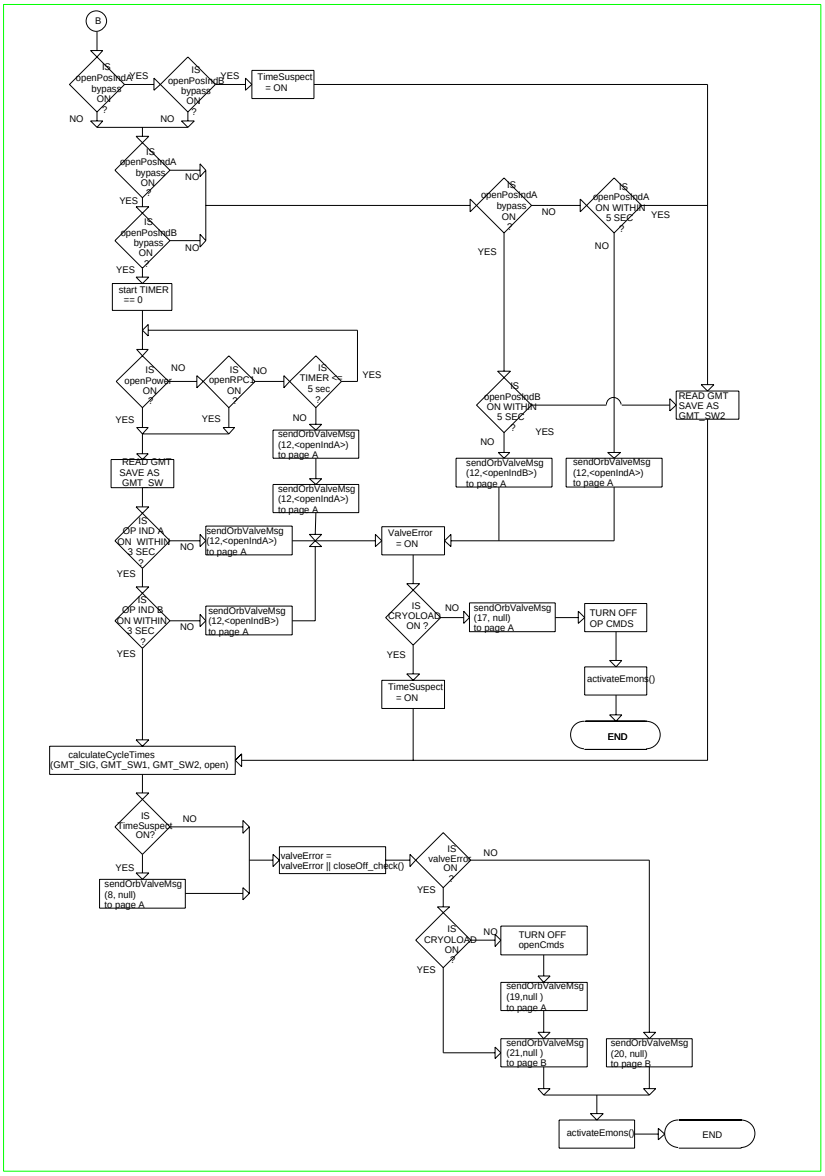
3.1.15.3.1.2. Sequence



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



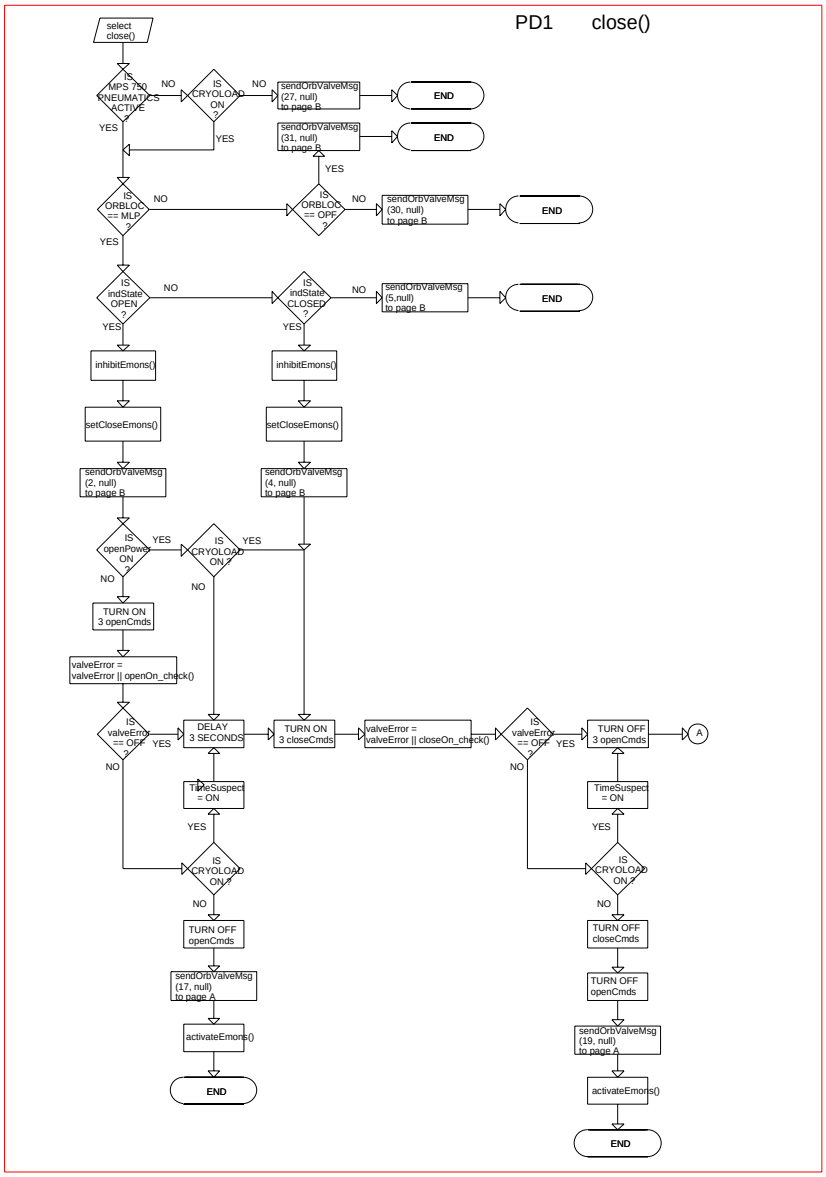
3.1.15.3.2. close()

3.1.15.3.2.1. Descriptive Text

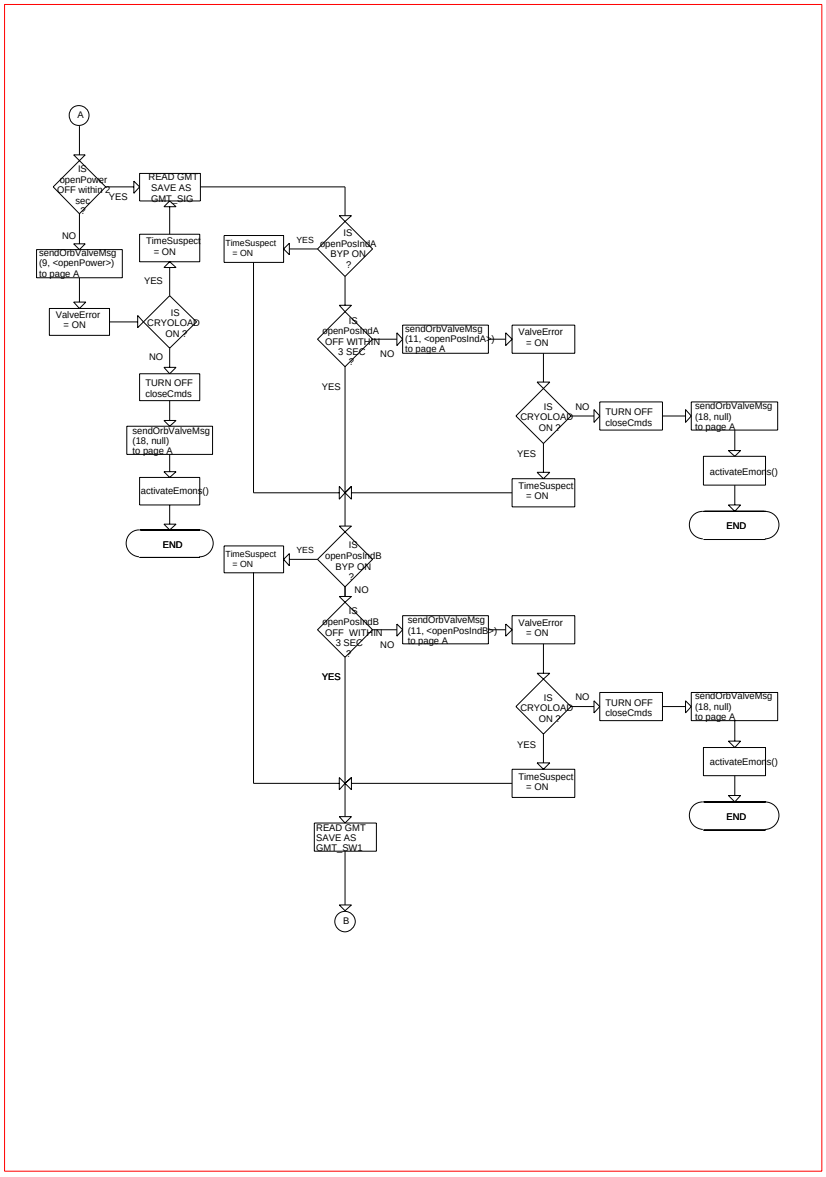
The flow charts contained below describe the logic to be used to code the method. The implementation of the details is left up to the coders and is acceptable as long as the overall logic is maintained. It is expected that cycle timing will be accomplished using the most accurate source. All messages will include a time stamp, and the appropriate FD description. ValveError and TimeSuspect are initialized to OFF and Valve cycle times are set to 0 prior to initiating valve cycle methods.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

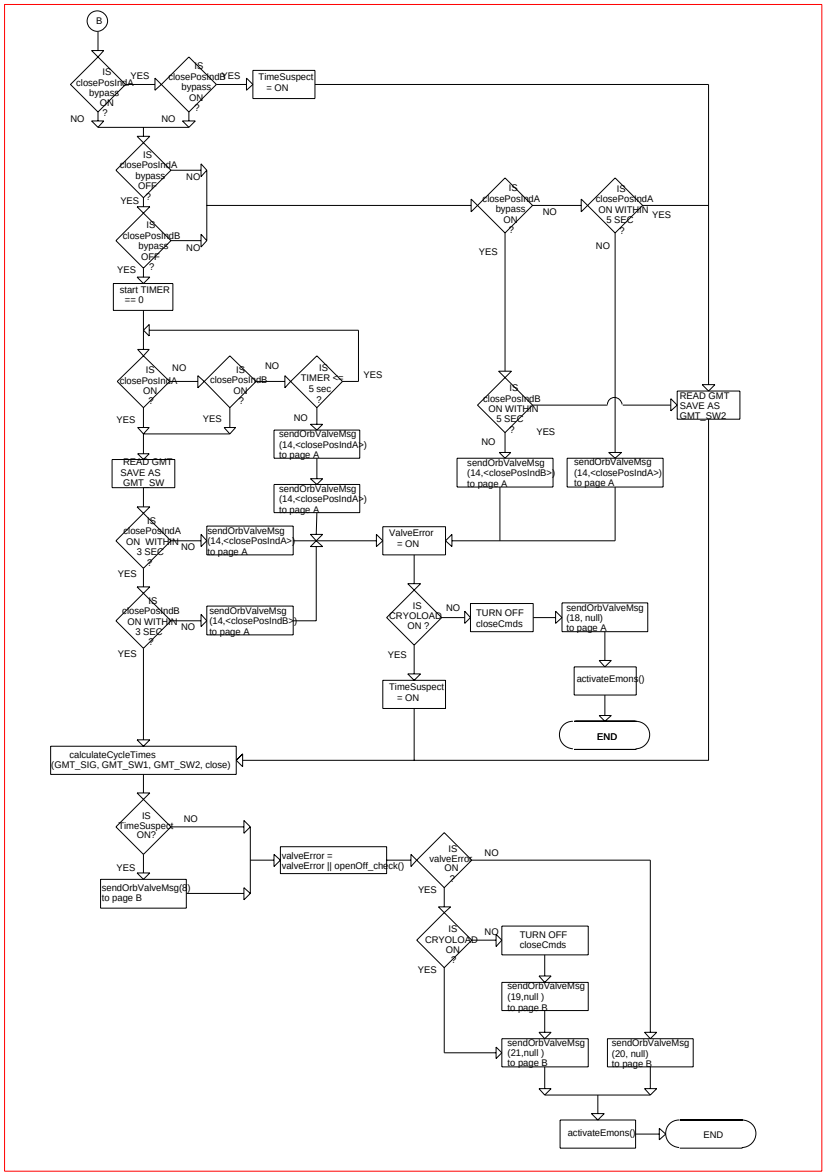
3.1.15.3.2.2. Sequence



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



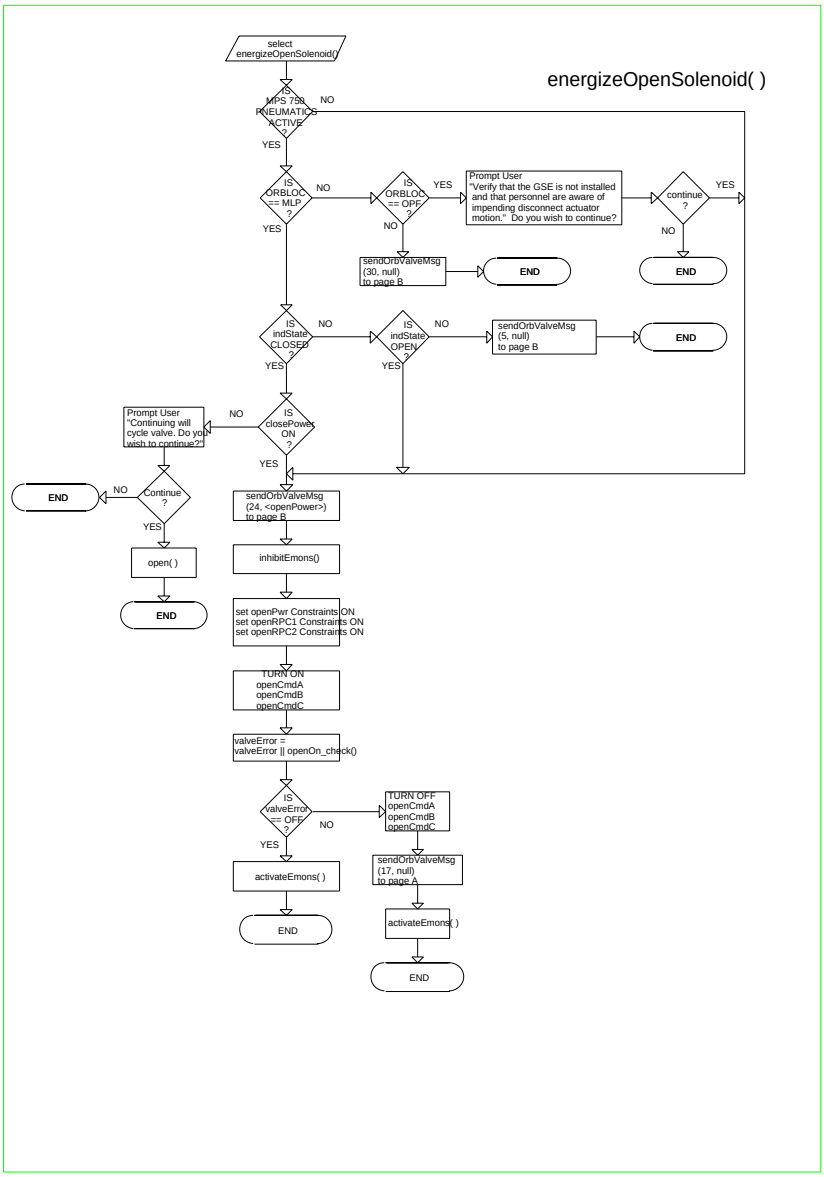
3.1.15.3.3. energizeOpenSolenoid()

3.1.15.3.3.1. Descriptive Text

The flow charts contained below describe the logic to be used to code the method. The implementation of the details is left up to the coders and is acceptable as long as the overall logic is maintained. It is expected that cycle timing will be accomplished using the most accurate source. All messages will include a time stamp, and the appropriate FD description. ValveError is to be initialized to OFF prior to initiating solenoid cycle methods.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.15.3.3.2. Sequence



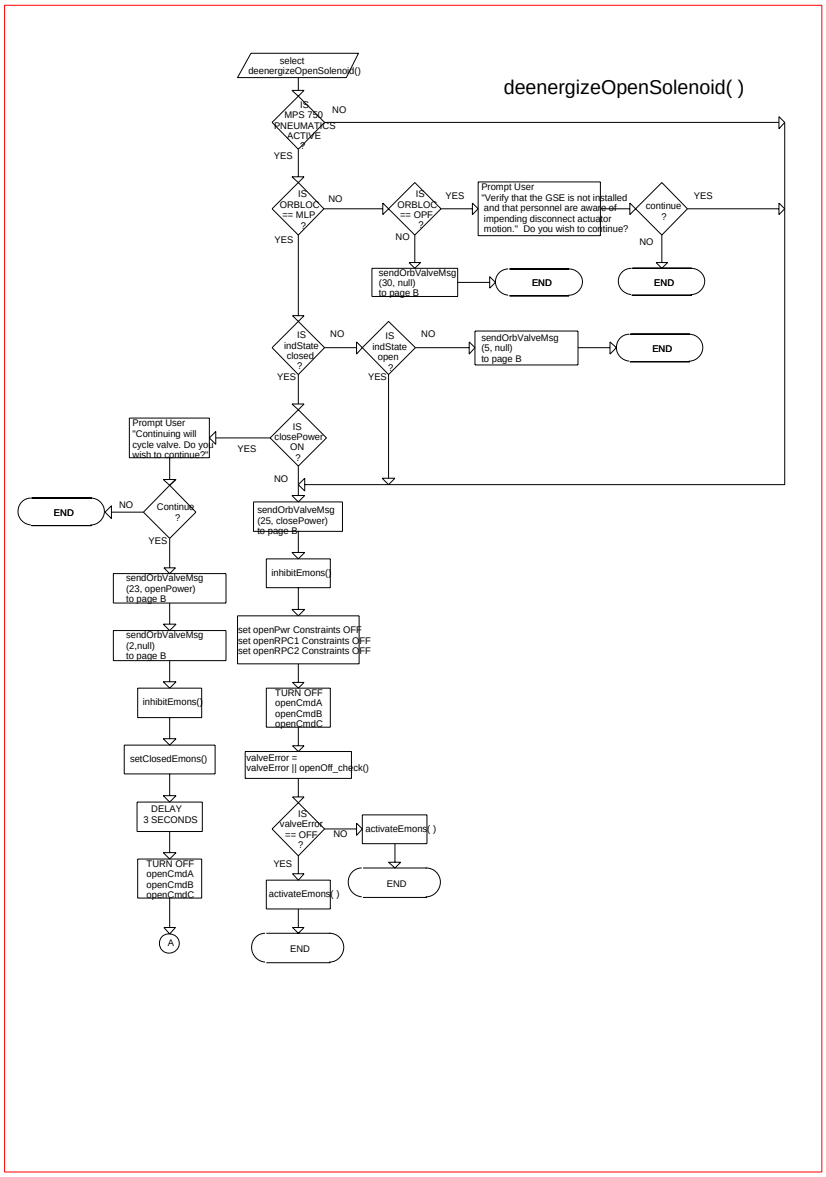
3.1.15.3.4. deenergizeOpenSolenoid()

3.1.15.3.4.1. Descriptive Text

The flow charts contained below describe the logic to be used to code the method. The implementation of the details is left up to the coders and is acceptable as long as the overall logic is maintained. It is expected that cycle timing will be accomplished using the most accurate source. All messages will include a time stamp, and the appropriate FD description. ValveError is to be initialized to OFF prior to initiating solenoid cycle methods.

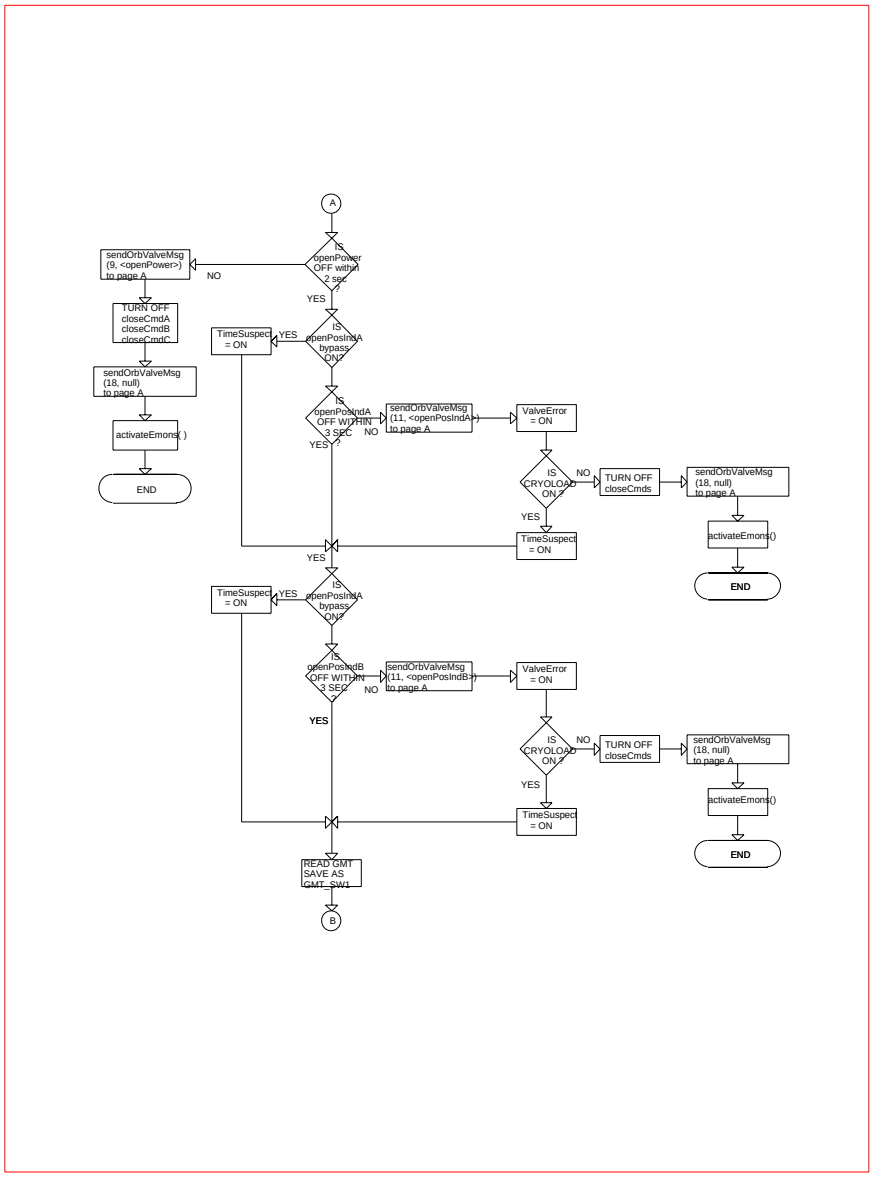
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.15.3.4.2. Sequence



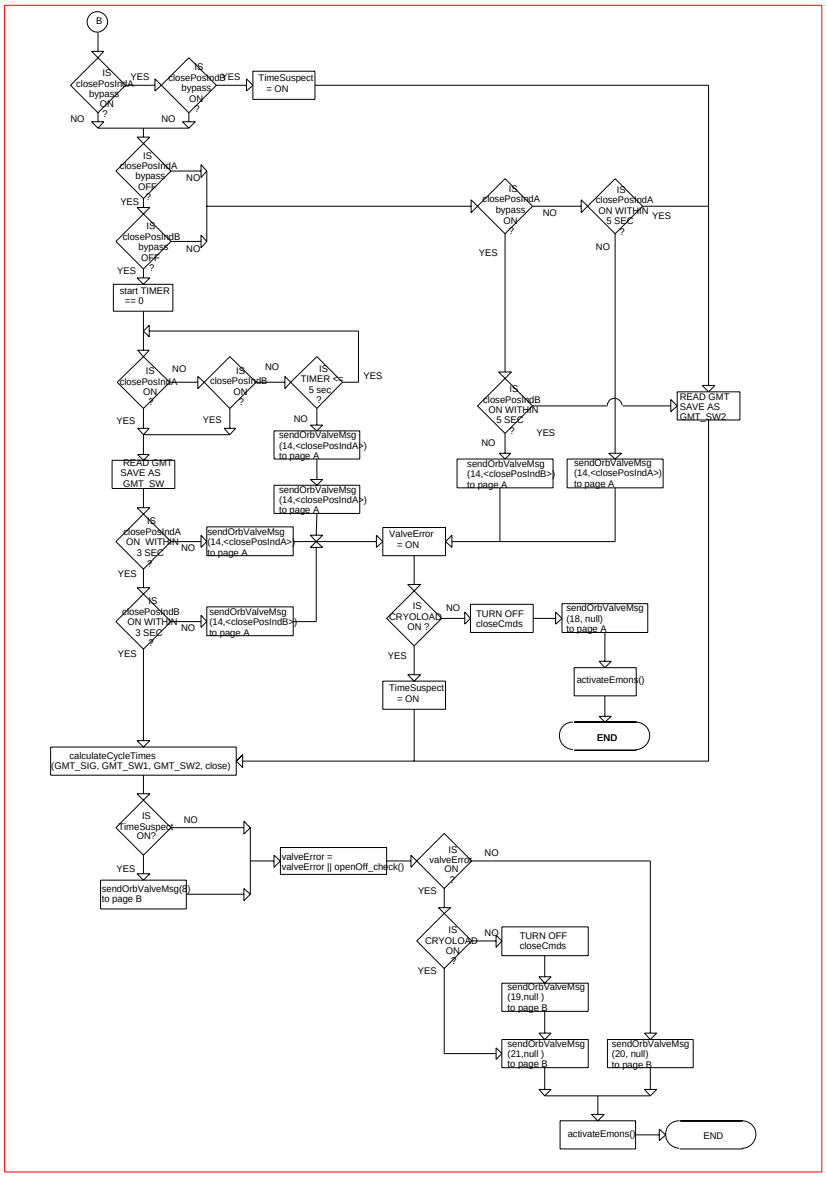
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.15.3.4.3. Sequence



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.15.3.4.4. Sequence



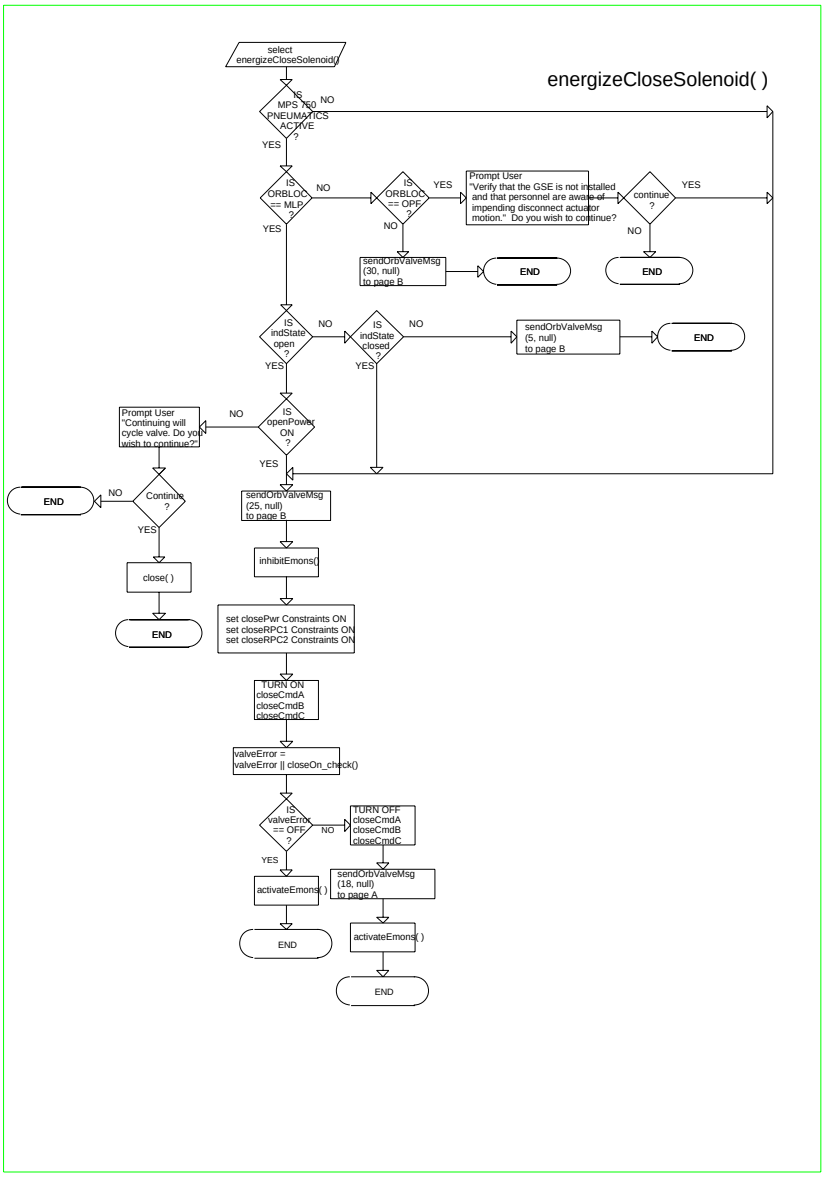
3.1.15.3.5. energizeCloseSolenoid()

3.1.15.3.5.1. Descriptive Text

The flow charts contained below describe the logic to be used to code the method. The implementation of the details is left up to the coders and is acceptable as long as the overall logic is maintained. It is expected that cycle timing will be accomplished using the most accurate source. All messages will include a time stamp, and the appropriate FD description. ValveError is to be initialized to OFF prior to initiating solenoid cycle methods.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.15.3.5.2. Sequence



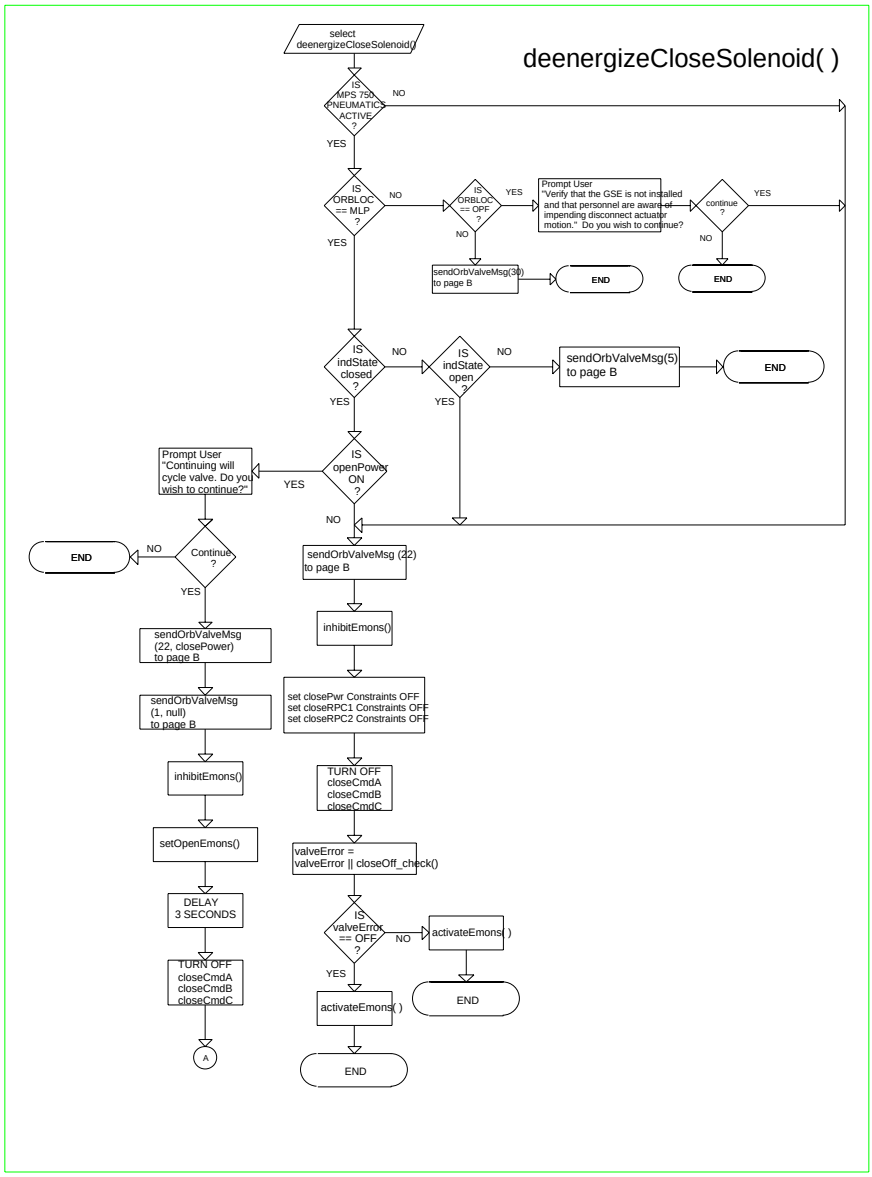
3.1.15.3.6. deenergizeCloseSolenoid()

3.1.15.3.6.1. Descriptive Text

The flow charts contained below describe the logic to be used to code the method. The implementation of the details is left up to the coders and is acceptable as long as the overall logic is maintained. It is expected that cycle timing will be accomplished using the most accurate source. All messages will include a time stamp, and the appropriate FD description. ValveError is to be initialized to OFF prior to initiating solenoid cycle methods.

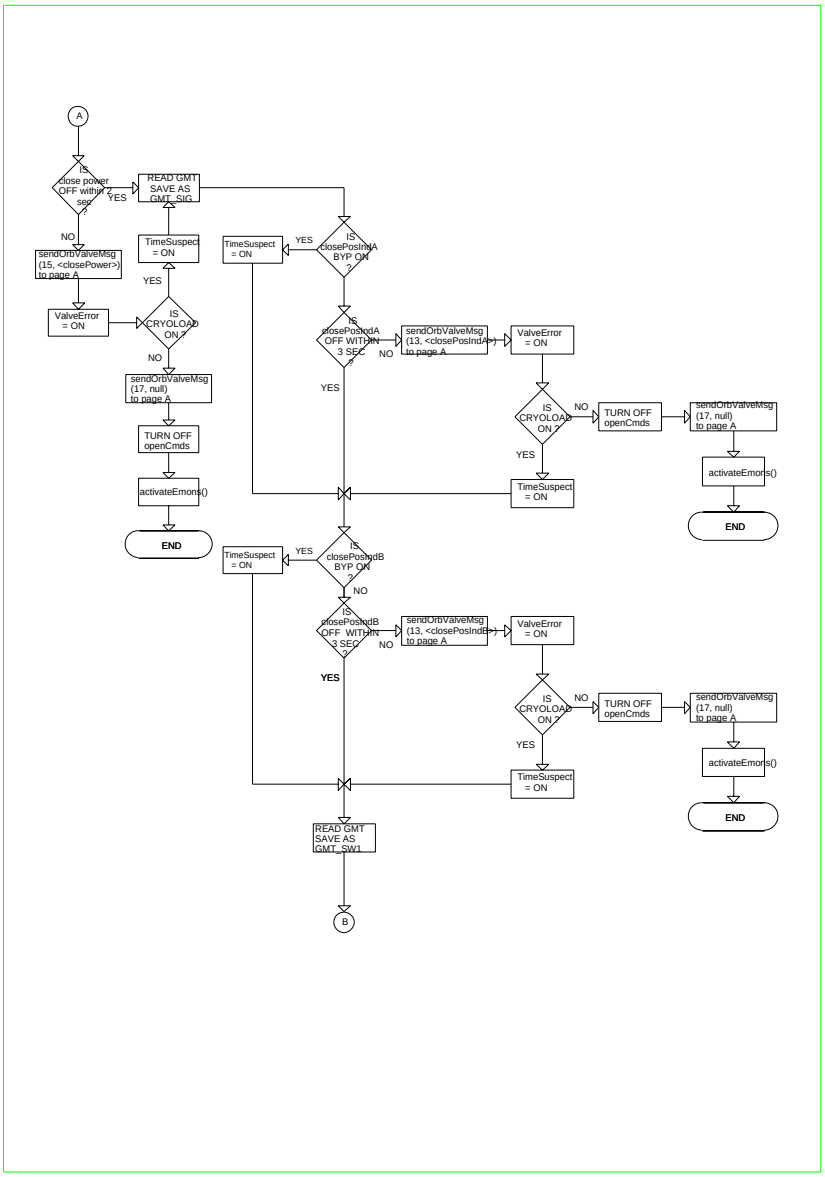
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.15.3.6.2. Sequence



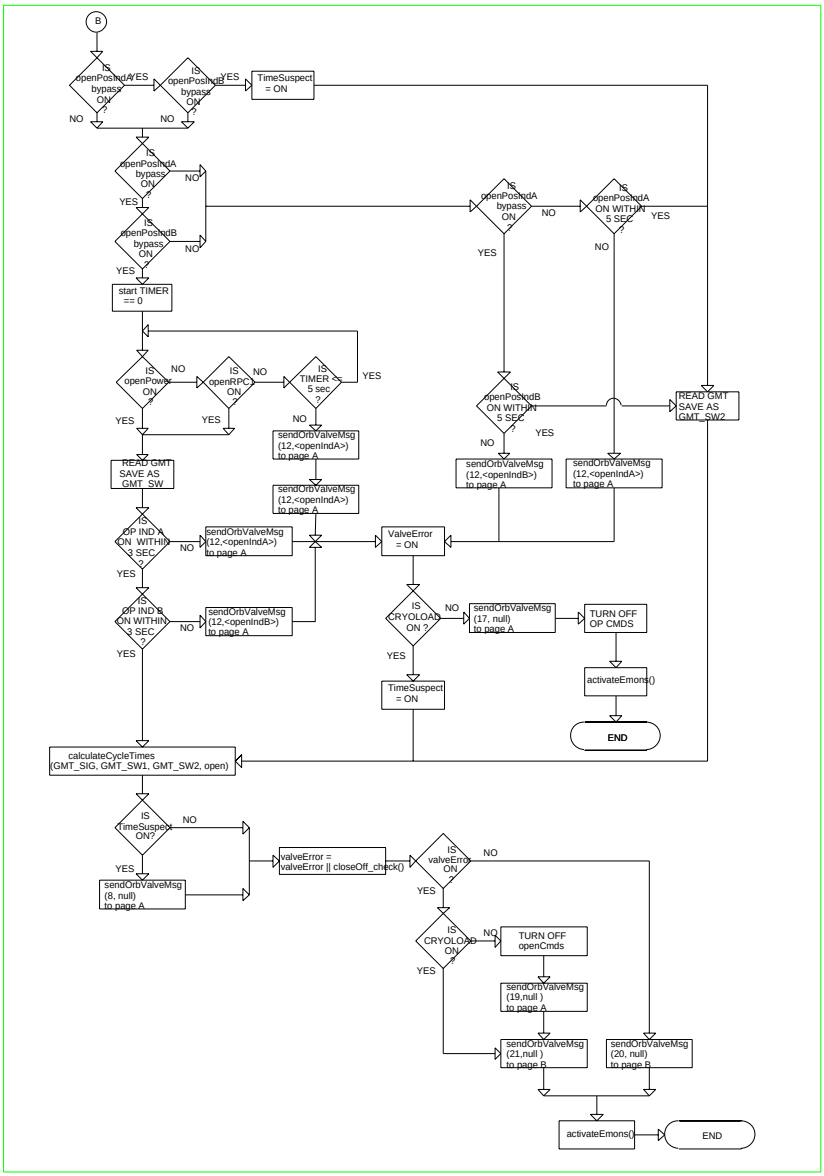
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.15.3.6.3. Sequence



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.15.3.6.4. Sequence



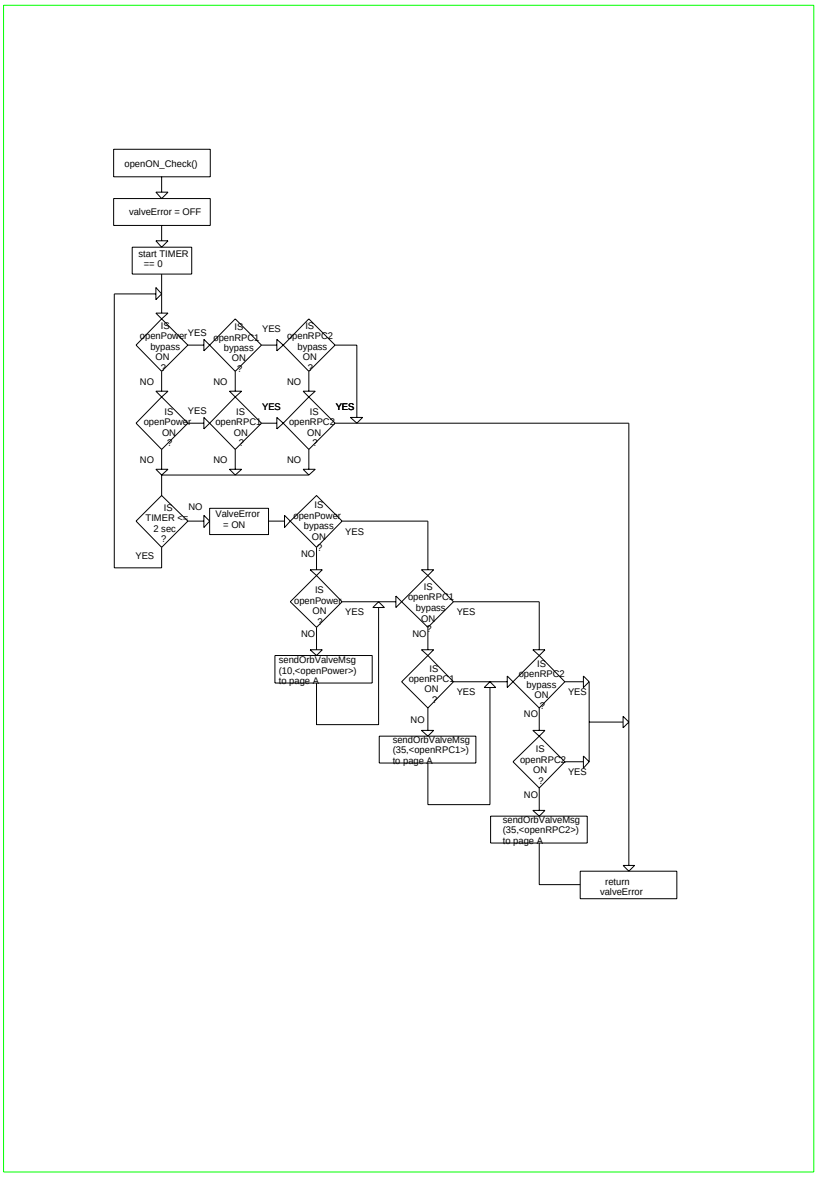
3.1.15.3.7. openON_check()

3.1.15.3.7.1. Descriptive Text

openON_check() is a private method that is called by open, close, and the energize and deenergize methods of the class. The openOn_check method verifies that the open powers and RPCs are on, and provides appropriate messaging if errors are encountered. If all powers and RPCs are either on or are bypassed then valveError equal to OFF is returned to the calling method. If errors are encountered the valveError equal to ON is returned. The flow charts contained below describe the logic to be used to code the method. The implementation of the details is left up to the coders and is acceptable as long as the overall logic is maintained. All messages will include a time stamp, and the appropriate FD description.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.15.3.7.2. Sequence



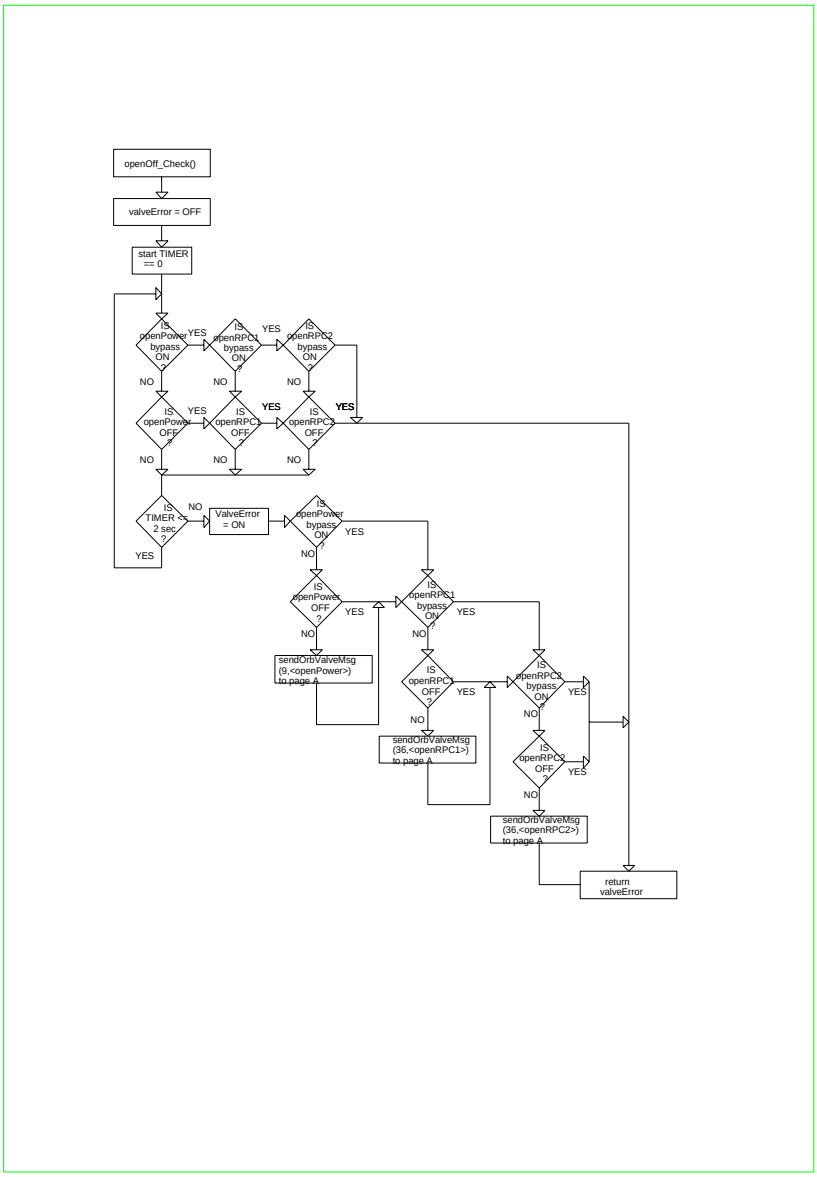
3.1.15.3.8. openOff_check()

3.1.15.3.8.1. Descriptive Text

openOff_check() is a private method that is called by open, close, and the energize and deenergize methods of the class. The openOff_check method verifies that the open powers and RPCs are OFF, and provides appropriate messaging if errors are encountered. If all powers and RPCs are either OFF or are bypassed then valveError equal to OFF is returned to the calling method. If errors are encountered the valveError equal to ON is returned. The flow charts contained below describe the logic to be used to code the method. The implementation of the details is left up to the coders and is acceptable as long as the overall logic is maintained. All messages will include a time stamp, and the appropriate FD description.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.15.3.8.2. Sequence



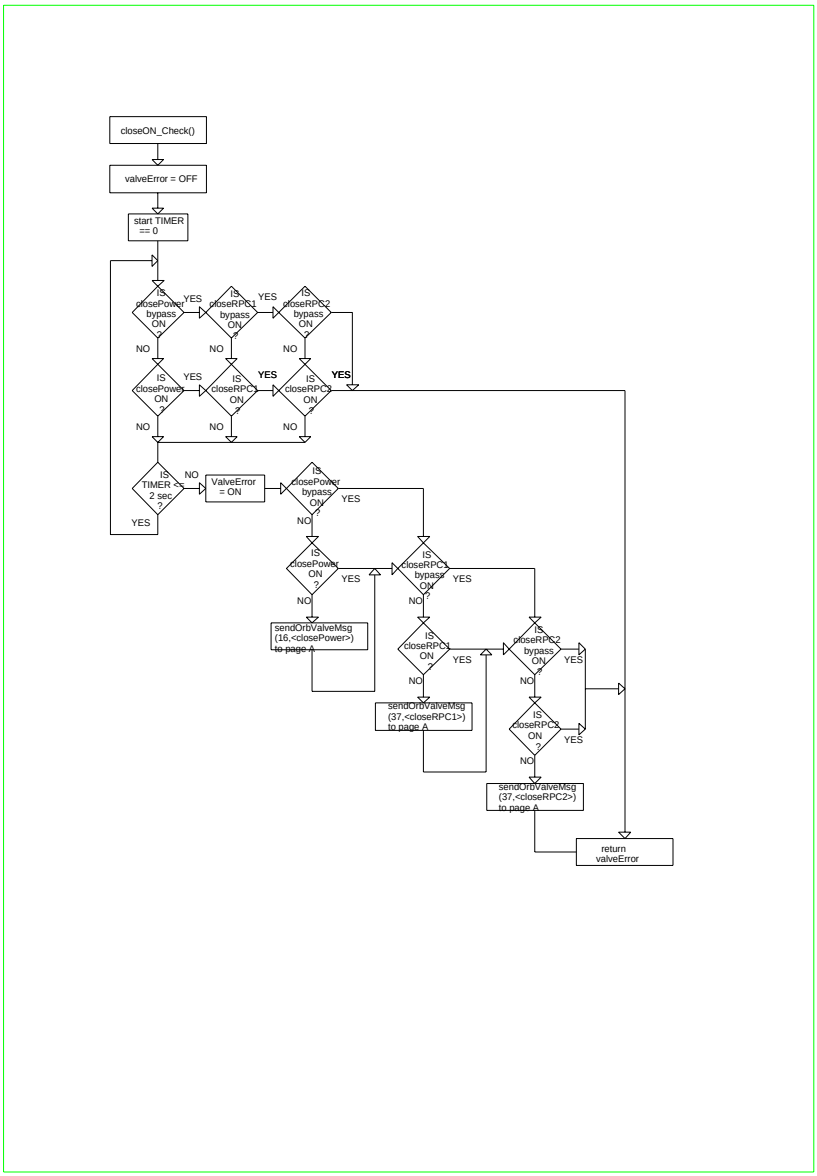
3.1.15.3.9. closeON_check()

3.1.15.3.9.1. Descriptive Text

closeON_check() is a private method that is called by open, close, and the energize and deenergize methods of the class. TheCloseON_check method verifies that the close powers and RPCs are ON, and provides appropriate messaging if errors are encountered. If all powers and RPCs are either ON or are bypassed then valveError equal to OFF is returned to the calling method. If errors are encountered the valveError equal to ON is returned. The flow charts contained below describe the logic to be used to code the method. The implementation of the details is left up to the coders and is acceptable as long as the overall logic is maintained. All messages will include a time stamp, and the appropriate FD description.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.1.15.3.9.2. Sequence



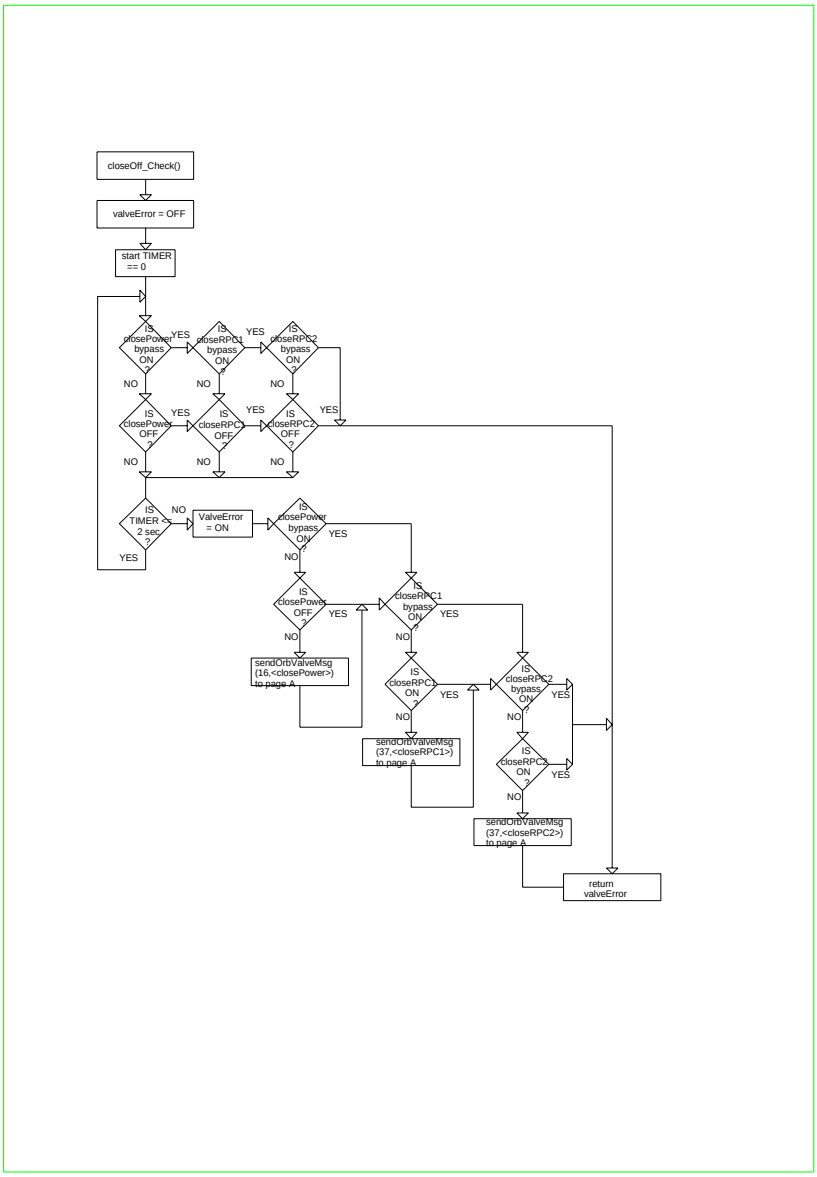
3.1.15.3.10. closeOff_check()

3.1.15.3.10.1. Descriptive Text

closeOff_check() is a private method that is called by open, close, and the energize and deenergize methods of the class. TheCloseOff_check method verifies that the close powers and RPCs are OFF, and provides appropriate messaging if errors are encountered. If all powers and RPCs are either OFF or are bypassed then valveError equal to OFF is returned to the calling method. If errors are encountered the valveError equal to ON is returned. The flow charts contained below describe the logic to be used to code the method. The implementation of the details is left up to the coders and is acceptable as long as the overall logic is maintained. All messages will include a time stamp, and the appropriate FD description.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.15.3.10.2. Sequence



3.1.15.3.11. determineIndState()

3.1.15.3.11.1. Descriptive Text

This method will determine the valve's position based on its position indicators and associated indicator bypasses. The return value will be an enumerated type with possible values of OPEN, CLOSED, TRANSITION, and ERROR.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.15.3.11.2. Logic Table

OPEN Indicator A	OPEN Indicator A Byp	OPEN Indicator B	OPEN Indicator B Byp	close Indicator A	close Indicator A BYP	close Indicator B	close Indicator B BYP	Ind state
----	ON	ON	OFF	OFF	OFF	OFF	OFF	OPEN
----	ON	ON	OFF	OFF	OFF	----	ON	OPEN
----	ON	ON	OFF	----	ON	OFF	OFF	OPEN
ON	OFF	----	ON	OFF	OFF	OFF	OFF	OPEN
ON	OFF	----	ON	OFF	OFF	----	ON	OPEN
ON	OFF	----	ON	----	ON	OFF	OFF	OPEN
ON	OFF	ON	OFF	OFF	OFF	OFF	OFF	OPEN
ON	OFF	ON	OFF	OFF	OFF	----	ON	OPEN
ON	OFF	ON	OFF	----	ON	OFF	OFF	OPEN
----	ON	----	ON	OFF	OFF	OFF	OFF	OPEN
----	ON	----	ON	OFF	OFF	----	ON	OPEN
----	ON	----	ON	----	ON	OFF	OFF	OPEN
----	ON	ON	OFF	----	ON	----	ON	OPEN
ON	OFF	----	ON	----	ON	----	ON	OPEN
ON	OFF	ON	OFF	----	ON	----	ON	OPEN
OFF	OFF	OFF	OFF	----	ON	ON	OFF	CLOSED
OFF	OFF	OFF	OFF	ON	OFF	----	ON	CLOSED
OFF	OFF	OFF	OFF	ON	OFF	ON	OFF	CLOSED
OFF	OFF	----	ON	----	ON	ON	OFF	CLOSED
OFF	OFF	----	ON	ON	OFF	----	ON	CLOSED
OFF	OFF	----	ON	ON	OFF	ON	OFF	CLOSED
----	ON	OFF	OFF	----	ON	ON	OFF	CLOSED
----	ON	OFF	OFF	ON	OFF	----	ON	CLOSED
----	ON	OFF	OFF	ON	OFF	ON	OFF	CLOSED
OFF	OFF	OFF	OFF	----	ON	----	ON	CLOSED
OFF	OFF	----	ON	----	ON	----	ON	CLOSED
----	ON	OFF	OFF	----	ON	----	ON	CLOSED
----	ON	----	ON	----	ON	ON	OFF	CLOSED
----	ON	----	ON	ON	OFF	----	ON	CLOSED
----	ON	----	ON	ON	OFF	ON	OFF	CLOSED
OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF	TRANSITION
OFF	OFF	OFF	OFF	OFF	OFF	----	ON	TRANSITION
OFF	OFF	OFF	OFF	----	ON	OFF	OFF	TRANSITION
OFF	OFF	----	ON	OFF	OFF	OFF	OFF	TRANSITION
OFF	OFF	----	ON	OFF	OFF	----	ON	TRANSITION
OFF	OFF	----	ON	----	ON	OFF	OFF	TRANSITION
----	ON	OFF	OFF	OFF	OFF	OFF	OFF	TRANSITION
----	ON	OFF	OFF	OFF	OFF	----	ON	TRANSITION
----	ON	OFF	OFF	----	ON	OFF	OFF	TRANSITION

3.1.15.3.12. determineOpenSolCmdState()

3.1.15.3.12.1. Descriptive Text

This method will determine the open solenoid's commanded state based on the two open RPCs, the open power, and the associated bypasses. The return value will be an enumerated

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

type with possible values of ENERGIZED, DEENERGIZED and ERROR.

3.1.15.3.12.2. Logic Table

Open Solenoid Command state						
open Power Ind	**open power bypass**	**open RPC_A**	**open RPC_A Bypass**	**open RPC_B**	**open RPC_B Bypass**	**OP CMD State**
OFF	OFF	OFF	OFF	OFF	OFF	DEENERGIZED
OFF	OFF	OFF	OFF	----	ON	DEENERGIZED
OFF	OFF	----	ON	OFF	OFF	DEENERGIZED
OFF	OFF	----	ON	----	ON	DEENERGIZED
----	ON	OFF	OFF	OFF	OFF	DEENERGIZED
----	ON	OFF	OFF	----	ON	DEENERGIZED
----	ON	----	ON	OFF	OFF	DEENERGIZED
----	ON	----	ON	ON	OFF	ENERGIZED
----	ON	ON	OFF	----	ON	ENERGIZED
----	ON	ON	OFF	ON	OFF	ENERGIZED
ON	OFF	----	ON	----	ON	ENERGIZED
ON	OFF	----	ON	ON	OFF	ENERGIZED
ON	OFF	ON	OFF	----	ON	ENERGIZED
ON	OFF	ON	OFF	ON	OFF	ENERGIZED
OFF	OFF	OFF	OFF	ON	OFF	ERROR
OFF	OFF	----	ON	ON	OFF	ERROR
OFF	OFF	ON	OFF	OFF	OFF	ERROR
OFF	OFF	ON	OFF	----	ON	ERROR
OFF	OFF	ON	OFF	ON	OFF	ERROR
----	ON	OFF	OFF	ON	OFF	ERROR
----	ON	----	ON	----	ON	ERROR
----	ON	ON	OFF	OFF	OFF	ERROR
ON	OFF	OFF	OFF	OFF	OFF	ERROR
ON	OFF	OFF	OFF	----	ON	ERROR
ON	OFF	OFF	OFF	ON	OFF	ERROR
ON	OFF	----	ON	OFF	OFF	ERROR
ON	OFF	ON	OFF	OFF	OFF	ERROR

3.1.15.3.13. determineCloseSolCmdState()

3.1.15.3.13.1. Descriptive Text

This method will determine the close solenoid's commanded state based on the two close RPCs, the close power, and the associated bypasses. The return value will be an enumerated
type with possible values of ENERGIZED, DEENERGIZED and ERROR.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.15.3.13.2. Logic Table

Close Solenoid Command State						
close power	close power bypass	close RPC_A	close RPC_A Bypass	close RPC_B	close RPC_B Bypass	CL CMD State
OFF	OFF	OFF	OFF	OFF	OFF	DEENERGIZED
OFF	OFF	OFF	OFF	----	ON	DEENERGIZED
OFF	OFF	----	ON	OFF	OFF	DEENERGIZED
OFF	OFF	----	ON	----	ON	DEENERGIZED
----	ON	OFF	OFF	OFF	OFF	DEENERGIZED
----	ON	OFF	OFF	----	ON	DEENERGIZED
----	ON	----	ON	OFF	OFF	DEENERGIZED
----	ON	----	ON	ON	OFF	ENERGIZED
----	ON	ON	OFF	----	ON	ENERGIZED
----	ON	ON	OFF	ON	OFF	ENERGIZED
ON	OFF	----	ON	----	ON	ENERGIZED
ON	OFF	----	ON	ON	OFF	ENERGIZED
ON	OFF	ON	OFF	----	ON	ENERGIZED
ON	OFF	ON	OFF	ON	OFF	ENERGIZED
OFF	OFF	OFF	OFF	ON	OFF	ERROR
OFF	OFF	----	ON	ON	OFF	ERROR
OFF	OFF	ON	OFF	OFF	OFF	ERROR
OFF	OFF	ON	OFF	----	ON	ERROR
OFF	OFF	ON	OFF	ON	OFF	ERROR
----	ON	OFF	OFF	ON	OFF	ERROR
----	ON	----	ON	----	ON	ERROR
----	ON	ON	OFF	OFF	OFF	ERROR
ON	OFF	OFF	OFF	OFF	OFF	ERROR
ON	OFF	OFF	OFF	----	ON	ERROR
ON	OFF	OFF	OFF	ON	OFF	ERROR
ON	OFF	----	ON	OFF	OFF	ERROR
ON	OFF	ON	OFF	OFF	OFF	ERROR

3.1.15.3.14. determineCmdState()

3.1.15.3.14.1. Descriptive Text

This method is used to determine the valve's overall commanded state based on closeSolCmdState and openSolCmdState. The return value will be an enumerated type with possible values of CMD_CLOSED, CMD_OPEN, UNCOMMANDED, CMD_SNUBBED, and ERROR.

3.1.15.3.14.2. Logic Table

3.1.15.3.15. determineState()

3.1.15.3.15.1. Descriptive Text

This method will determine the over-all state for the valve based on below logic table of the command state and position state.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

Combined Command State			
OP_CMDS \ CL_CMDS	ENERGIZED	DEENERGIZED	ERROR
DEENERGIZED	CMD_CLOSED	UNCOMMANDED	ERROR
ENERGIZED	CMD_SNUBBED	CMD_OPEN	ERROR
ERROR	ERROR	ERROR	ERROR

3.1.15.3.15.2. Logic Table

State (overall)					
pos \ cmd	cmd closed	cmd open	cmd snubbed	uncommanded	error
closed	closed	close error	closed_error	closed_error	closed_error
open	open error	open	open_error	open_error	open_error
transition	transition	transition	error	error	error
error	error	error	error	error	error

3.1.15.3.16. getIndState()

3.1.15.3.16.1. Descriptive Text

This method will return the Enumerated Indicator State of the valve

3.1.15.3.16.2. Sequence

```
getIndState()
{
        return indState;
}
```

3.1.15.3.17. getOpenSolCmdState()

3.1.15.3.17.1. Descriptive Text

This method will return the Enumerated Command State of the valve.

3.1.15.3.17.2. Sequence

```
getOpenSolCmdState()
{
        return openSolCmdState;
}
```

3.1.15.3.18. getCloseSolCmdState()

3.1.15.3.18.1. Descriptive Text

This method will return the Enumerated Command State of the valve.

3.1.15.3.18.2. Sequence

```
getCloseSolCmdState()
{
        return closeSolCmdState;
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.1.15.3.19. getCmdState()

3.1.15.3.19.1. Descriptive Text

This method will return the Enumerated Command State of the valve.

3.1.15.3.19.2. Sequence

```
getCmdState()
{
        return cmdState;
}
```

3.1.15.3.20. getState()

3.1.15.3.20.1. Descriptive Text

This method will return the overall Enumerated State of the valve.

3.1.15.3.20.2. Sequence

```
getState()
{
        return state;
}
```

3.1.15.3.21. activateConstraints()

3.1.15.3.21.1. Descriptive Text

This method will activate constraint monitoring of all unbypassed measurement of this valve.

3.1.15.3.21.2. Sequence

```
activateConstraints()
{
begin sequence;
        {
        if(<openPosIndA>.Byp == OFF)
                {
                activate constraint monitoring for <openPosIndA>
                }
        if(<openPosIndB>.Byp == OFF)
                {
                activate constraint monitoring for <openPosIndB>
                }
        if(<openPwr>.Byp == OFF)
                {
                activate constraint monitoring for <openPwr>
                }
        if(<openRPCB>.Byp == OFF)
                {
                activate constraint monitoring for <openRPCB>
                }
        if(<openRPCC>.Byp == OFF)
                {
                activate constraint monitoring for <openRPCC>
                }
        if(<closePwr>.Byp == OFF)
                {
                activate constraint monitoring for <closePwr>
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
            }
        if(<closePosIndA>.Byp == OFF)
            {
            activate constraint monitoring for <closePosIndA>
            }
        if(<closePosIndB>.Byp == OFF)
            {
            activate constraint monitoring for <closePosIndB>
            }
        if(<closeRPCB>.Byp == OFF)
            {
            activate constraint monitoring for <closeRPCB>
            }
        if(<closeRPCC>.Byp == OFF)
            {
            activate constraint monitoring for <closeRPCC>
            }

end sequence;
}
```

3.1.15.3.22. inhibitConstraints()

3.1.15.3.22.1. Descriptive Text

This method will inhibit constraint monitoring of all measurements related to this valve.

3.1.15.3.22.2. Sequence

```
inhibitConstraints()
        {
        inhibit constraint monitoring of the following:
        <openPosIndA>
        <openPosIndB>
        <openRPCB>
        <openRPCC>
        <openPwr>
        <closePosIndA>
        <closePosIndB>
        <closePwr>
        <closeRPCB>
        <closeRPCC>
        }
```

3.1.15.3.23. setOpenConstraints()

3.1.15.3.23.1. Descriptive Text

This method changes the constraint criteria of the measurements associated with this valve to those expected with the valve in the open position.

3.1.15.3.23.2. Sequence

```
        void setOpenConstraints()
        {
            CHANGE <openPwr> CONSTRAINT CRITERIA TO ON;

          CHANGE <openPosIndA>  CONSTRAINT CRITERIA TO ON;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

CHANGE <openPosIndB> CONSTRAINT CRITERIA TO ON;

CHANGE <openRPCB> CONSTRAINT CRITERIA TO ON;

CHANGE <openRPCC> CONSTRAINT CRITERIA TO ON;

CHANGE <closePwr> CONSTRAINT CRITERIA TO OFF;

CHANGE <closePosIndA> CONSTRAINT CRITERIA TO OFF;

CHANGE <closePosIndB> CONSTRAINT CRITERIA TO OFF;

CHANGE <closeRPCB> CONSTRAINT CRITERIA TO OFF;

CHANGE <closeRPCC> CONSTRAINT CRITERIA TO OFF;

}

3.1.15.3.24. setClosedConstraints()

3.1.15.3.24.1. Descriptive Text

This method changes the constraint criteria of the measurements associated with this valve to those expected with the valve in the closed position.

3.1.15.3.24.2. Sequence

setCloseConstraints()

{

CHANGE <openPwr> CONSTRAINT CRITERIA TO OFF;

CHANGE <openPosIndA> CONSTRAINT CRITERIA TO OFF;

CHANGE <openPosIndB> CONSTRAINT CRITERIA TO OFF;

CHANGE <openRPCB> CONSTRAINT CRITERIA TO OFF;

CHANGE <openRPCC> CONSTRAINT CRITERIA TO OFF;

CHANGE <closePwr> CONSTRAINT CRITERIA TO ON;

CHANGE <closePosIndA> CONSTRAINT CRITERIA TO ON;

CHANGE <closePosIndB> CONSTRAINT CRITERIA TO ON;

CHANGE <closeRPCB> CONSTRAINT CRITERIA TO ON;

CHANGE <closeRPCC> CONSTRAINT CRITERIA TO ON;

}

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.15.3.25. setCloseIndConstraints()

3.1.15.3.25.1. Description

This method changes the constraint criteria of the measurements associated with this valve to those expected with the valve in the closed bistable (uncommanded) position.

3.1.15.3.25.2. Sequence

```
setCloseIndConstraints()
        {
                CHANGE <openPwr> CONSTRAINT CRITERIA TO OFF;

                CHANGE <openPosIndA>  CONSTRAINT CRITERIA TO OFF;

                CHANGE <openPosIndB> CONSTRAINT CRITERIA TO OFF;

                CHANGE <openRPCB> CONSTRAINT CRITERIA TO OFF;

                CHANGE <openRPCC>  CONSTRAINT CRITERIA TO OFF;

                CHANGE <closePwr> CONSTRAINT CRITERIA TO OFF;

                CHANGE <closePosIndA> CONSTRAINT CRITERIA TO ON;

                CHANGE <closePosIndB>  CONSTRAINT CRITERIA TO ON;

                CHANGE <closeRPCB> CONSTRAINT CRITERIA TO OFF;

                CHANGE <closeRPCC> CONSTRAINT CRITERIA TO OFF;
        }
```

3.1.15.3.26. verifyCommandsOff()

3.1.15.3.26.1. Descriptive Text

This function will verify that the open and close solenoid commands are off by performing a Byte Test 4 for each command.

3.1.15.3.26.2. Sequence

```
verifyCommandsOff()
{
open command status 1 = byteTest4 <openCMDA>
        {
        verify  open command status 1 == OFF
        else print message to page A :   "Byte Test 4 shows", valveID, "open command" <openCMDA> "to be ON"

open command status 2 = byteTest4 <openCMDB>
        {
        verify  open command status 2 == OFF
        else print message to page A :   "Byte Test 4 shows", valveID, "open command" <openCMDB> "to be ON"

open command status 3 = byteTest4 <openCMDC>
        {
        verify  open command status 3 == OFF
        else print message to page A :   "Byte Test 4 shows", valveID, "open command" <openCMDC> "to be ON"

close command status 1 = byteTest4 <closeCMDA>
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
        {
        verify  close command status 1 == OFF
        else print message to page A :   "Byte Test 4 shows", valveID, "close command" <closeCMDA> "to be
ON"

close command status 2 = byteTest4 <closeCMDB>
        {
        verify  close command status 2 == OFF
        else print message to page A :   "Byte Test 4 shows", valveID, "close command" <closeCMDB> "to be
ON"

close command status 3 = byteTest4 <closeCMDC>
        {
        verify  close command status 3 == OFF
        else print message to page A :   "Byte Test 4 shows", valveID, "close command" <closeCMDC> "to be
ON"

        }
{
        verify  close command status 1 && close command status 2 && close command status 3 == OFF
        Return TRUE;

        Else return FALSE;

}
end sequence;
```

3.1.15.3.27. sendDisconnectMsg(int)

3.1.15.3.27.1. Descriptive Text

This method returns MPS Prevalve specific messages to the MPS valve software..

3.1.15.4. User Interfaces

3.1.15.5. Safety Requirements

3.1.16. MPS_Valve_ORB_Latch

3.1.16.1. Description

MPS_Valve_ORB_Latch is a bi-stable component with 3 lock and 3 unlock commands, 2 lock and 2 unlock position indications, 1 lock and 1 unlock power indications, 2 lock and 2 unlock RPCs. The latch actuator solenoids (1 each lock and unlock) are each redundantly commanded/powered. There are 2 instances of this latch, one for each 17 " disconnect feedline (MPS_PD1, PD2). Functionally, the latch serves to provide redundancy against inadvertent closure of the 17" disconnects during powered flight. The latch serves as a mechanical stop to prevent disconnect flapper closure under flow conditions by physically preventing the mated ORB/ET 17" disconnect flappers from closing should the 17" pneumatic actuator mechanism fail. PCL and onboard TCS program prevents inadvertent cycling of the 17" disconnect flappers into a locked latch (latch must be unlocked during disconnect cycle operations to prevent hardware damage). The valve class inherits from the MPS_Valve_ORB Class. The latch class has an association with the MPS_Helium_System class for status of the MPS 750 pneumatic system. For integrated operations, this class has an association with the MPS_System for determination of the cryo load stautus via a Cryo Flag pseudo. During test and check-out mode, latch cycles will break-out and remove all commands/power when a failure is detected. When the Cyro Flag is ON, the latch cycle will continue to completion regardless of failures.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.1.16.1.1. Instance Index

Instance	Other Users
MPS_PD1_LATCH	none
MPS_PD2_LATCH	none

3.1.16.1.2. Class Diagram

<insert class diagram here>

3.1.16.1.3. Attributes and Aggregations

Attribute	Data Type	Initial Value	Publish / Persistence	Client Access	Description
lockPosIndA	DM_ON_OFF		PUB	R	
lockPosIndB	DM_ON_OFF			R	
unlockPosIndA	DM_ON_OFF		PUB	R	
unlockPosIndB	DM_ON_OFF			R	
lockCmdA	DS_ON_OFF			R	
lockCmdB	DS_ON_OFF			R	
lockCmdC	DS_ON_OFF			R	
lockRpcB	DM_ON_OFF			R	
lockRpcC	DM_ON_OFF			R	
lockPwr	DM_ON_OFF		PUB	R	
unlockCmdA	DS_ON_OFF				
unlockCmdB	DS_ON_OFF				
unlockCmdC	DS_ON_OFF				
unlockPwr	DM_ON_OFF		PUB	R	
unlockRpcB	DM_ON_OFF			R	
unlockRpcC	DM_ON_OFF			R	
unlockSolCmdState	MPS_ORB_SOL_STATE ENUM		PUB	R	
unlockSolCmdState	MPS_ORB_SOL_STATE ENUM		PUB	R	

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

indState	MPS_ORB_LATCH_IND_ STATE ENUM		PUB	R	
cmdState	MPS_ORB_VALVE_CMD_ STATE ENUM		PUB	R	
state	MPS_ORB_LATCH_STATE ENUM		PUB	R	

3.1.16.1.4. Methods Index

Operation	**Description**
lock()	This method performs necessary functions to cycle latch to the locked position.
unlock()	This method performs necessary functions to cycle latch to the unlocked position.
energizeLockSolenoid()	This method energizes lock solenoid. Method includes a check of MPS pneumatics status.
energizeUnlockSolenoid()	This method energizes unlock solenoid. Method includes a check of MPS pneumatics status.
deenergizeLockSolenoid()	This method de-energizes lock solenoid.
deenergizeUnlockSolenoid()	This method de-energizes unlock solenoid.
lockON_check()	This method verifies that the lock powers and RPCs are energized, and provides appropriate messages if they are not. The method returns valveError equal to ON if an unbypassed power or RPC does not achieve the correct state within the alloted time.
lockOff_check()	This method verifies that the lock powers and RPCs are deenergized, and provides appropriate messages if they are not. The method returns valveError equal to ON if an unbypassed power or RPC does not achieve the correct state within the alloted time.
unlockON_check()	This method verifies that the unlock powers and RPCs are energized, and provides appropriate messages if they are not. The method returns valveError equal to ON if an unbypassed power or RPC does not achieve the correct state within the alloted time.
unlockOff_check()	This method verifies that the unlock powers and RPCs are deenergized, and provides appropriate messages if they are not. The method returns valveError equal to ON if an unbypassed power or RPC does not achieve the correct state within the alloted time.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

determineIndState()	This method will determine the position state of the latch based on all unbypassed lock / unlock indicators and will return either: LOCKED UNLOCKED TRANSITION ERROR
determineLockSolCmdState()	This method will determine the command state of the locking solenoid based on power indicators and RPCs and will return either: ENERGIZED DEENERGIZED ERROR
determineUnlockSolCmdState()	This method will determine the command state of the unlocking solenoid based on power indicators and RPCs and will return either: ENERGIZED DEENERGIZED ERROR
determineCmdState()	This method will determine the overall command state of the latch by voting on lockSolCmdState and unlockSolCmdState. The possible overall command states are: CMD_LOCKED CMD_UNLOCKED CMD_SNUBBED UNCOMMANDED ERROR
determineState()	This method will determine the overall state of the valve by voting on the overall command state and the indicator state. The possible states are: LOCKED UNLOCKED LOCKED_ERROR UNLOCKED_ERROR TRANSITION ERROR
getIndState()	This method will return latch indicator Enumerated state.
getLockSolCmdState()	This method will return latch locking solenoid Enumerated command state.
getUnlockSolCmdState()	This method will return latch unlocking solenoid Enumerated command state.
getCmdState()	This method will return the overall Enumerated latch command state.
getState()	This method will return the overall state of the latch
verifyCmdsOff()	This method will verify no command issued by performing a byte Test 4 of all latch instance commands.
activateConstraints()	This method will activateconstraint criteria monitoring of all FDs

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

	associated with the instance when in standby mode (uncommanded present state).
inhibitConstraints()	This method will inhibit Constraint notification during latch cycle operations.
autoCycle()	This method will perform latch lock / unlock autocycle operations.
setLockConstraints()	This method will set the constraint criteria for all indicators associated with the latch instance to the states expected for the latch in the locked position.
setUnlockConstraints()	This method will set the constraint criteria of all indicators associated with the latch instance to the states expected for the latch in the unlocked position.
setUnlockPosConstraints()	This method will set the constraint criteria of all indicators associated with the latch instance to the states expected for the latch in the unlocked bistable position.

3.1.16.1.5. Associations

Association	Type	Class	Description
MPS_Subsystem			CryoFlag status
MPS_Helium_Subsystem			MPS750 status
MPS_PCL_PD1_LATCH_LK_CMD		Prerequisite Control Logic	PCL for PD1 latch lock issuance
MPS_PCL_PD1_LATCH_UL_CMD		Prerequisite Control Logic	PCL for PD1 latch unlock issuance
MPS_PCL_PD2_LATCH_LK_CMD		Prerequisite Control Logic	PCL for PD2 latch lock issuance
MPS_PCL_PD2_LATCH_UL_CMD		Prerequisite Control Logic	PCL for PD2 latch unlock issuance

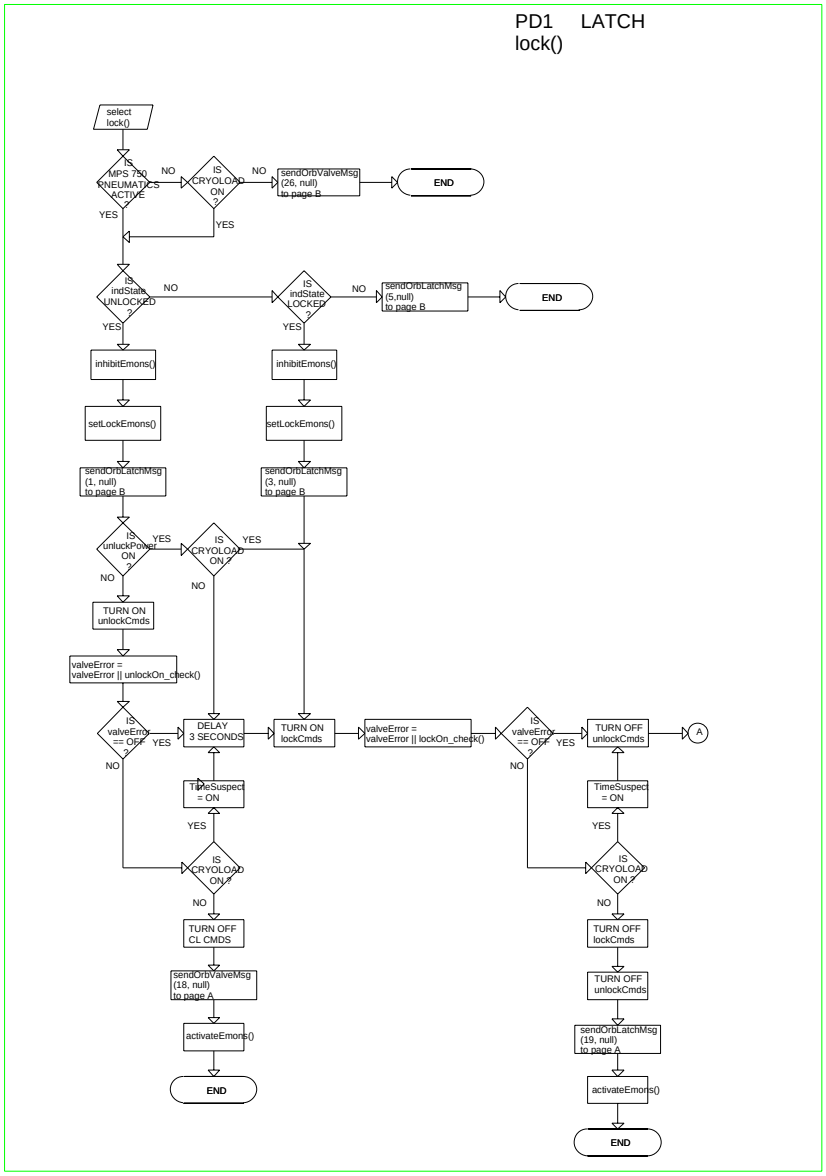
3.1.16.2. Operations

3.1.16.2.1. lock()

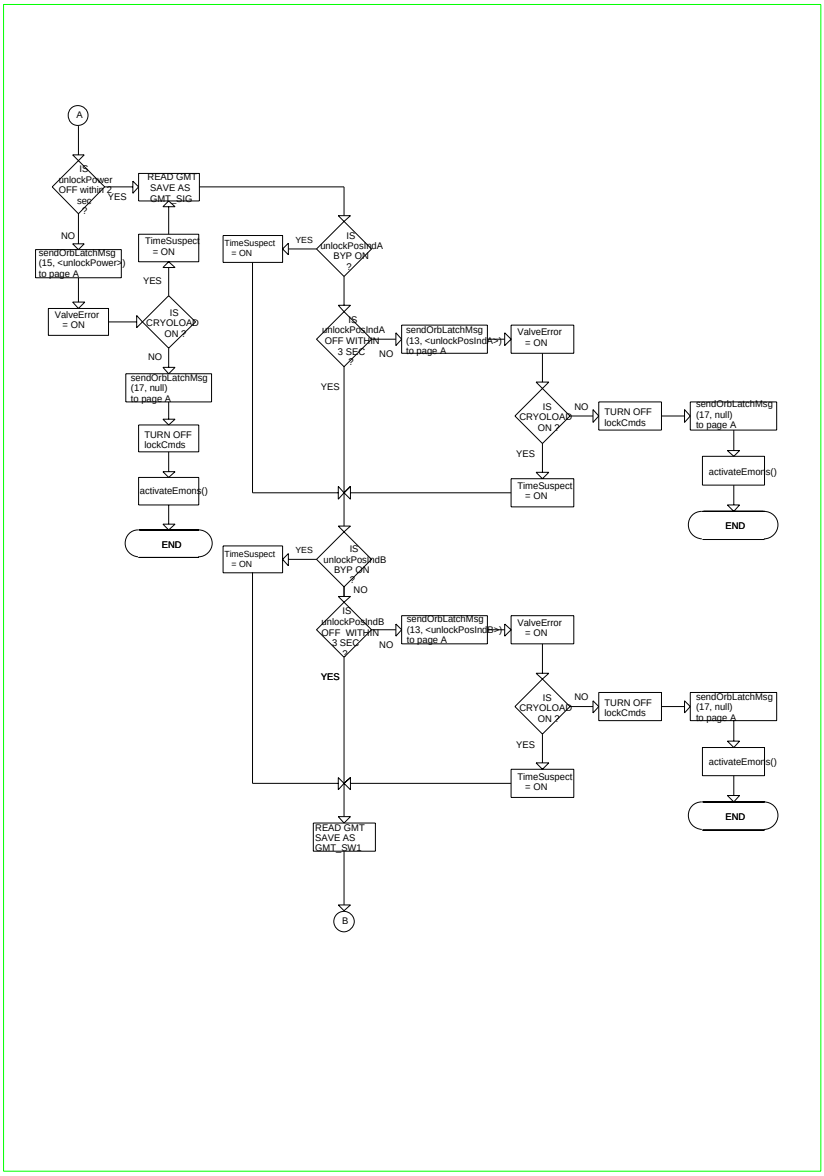
3.1.16.2.1.1. Descriptive Text

The flow charts contained below describe the logic to be used to code the method. The implementation of the details is left up to the coders and is acceptable as long as the overall logic is maintained. It is expected that cycle timing will be accomplished using the most accurate source. All messages will include a time stamp, and the appropriate FD description. ValveError and TimeSuspect are initialized to OFF and Valve cycle times are set to 0 prior to starting valve cycle methods. If ValveError gets set ON during method run, cycle time FD data health should be set to invalid to aid in identifying suspect cycle data in trend analysis.

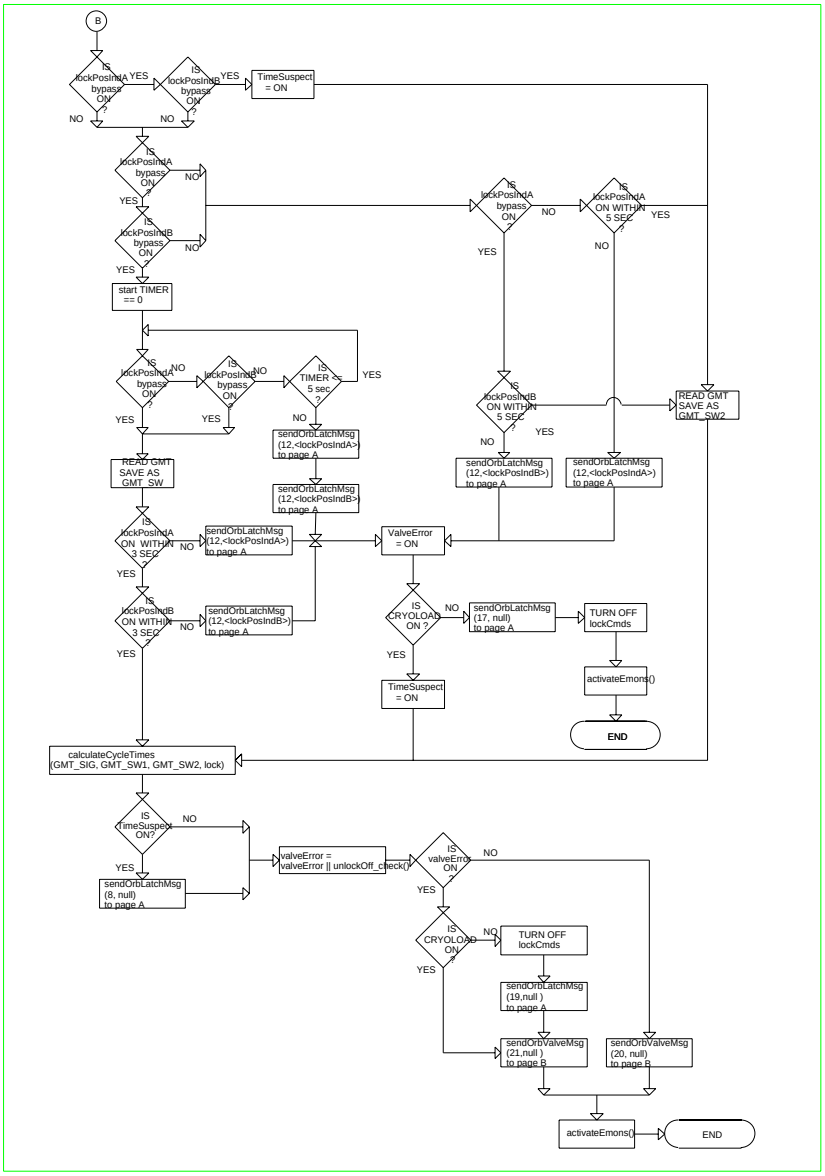
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

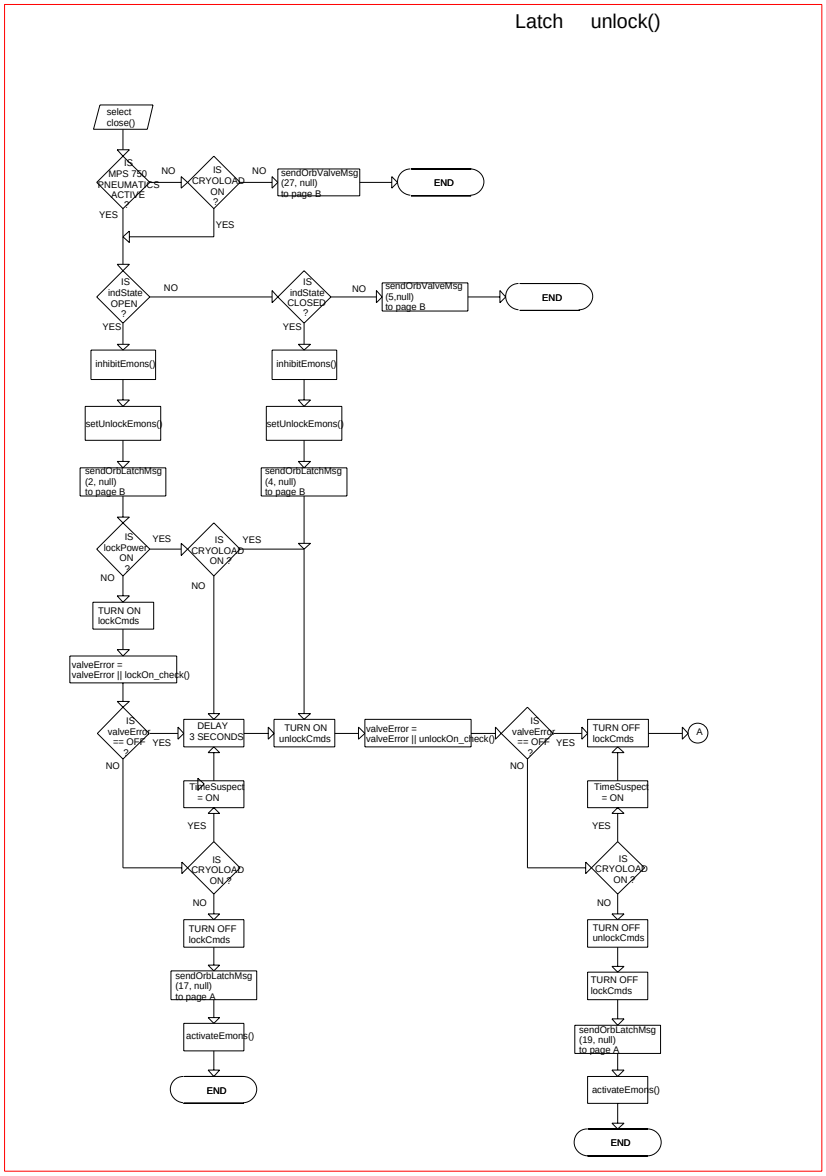


3.1.16.2.2. unlock()

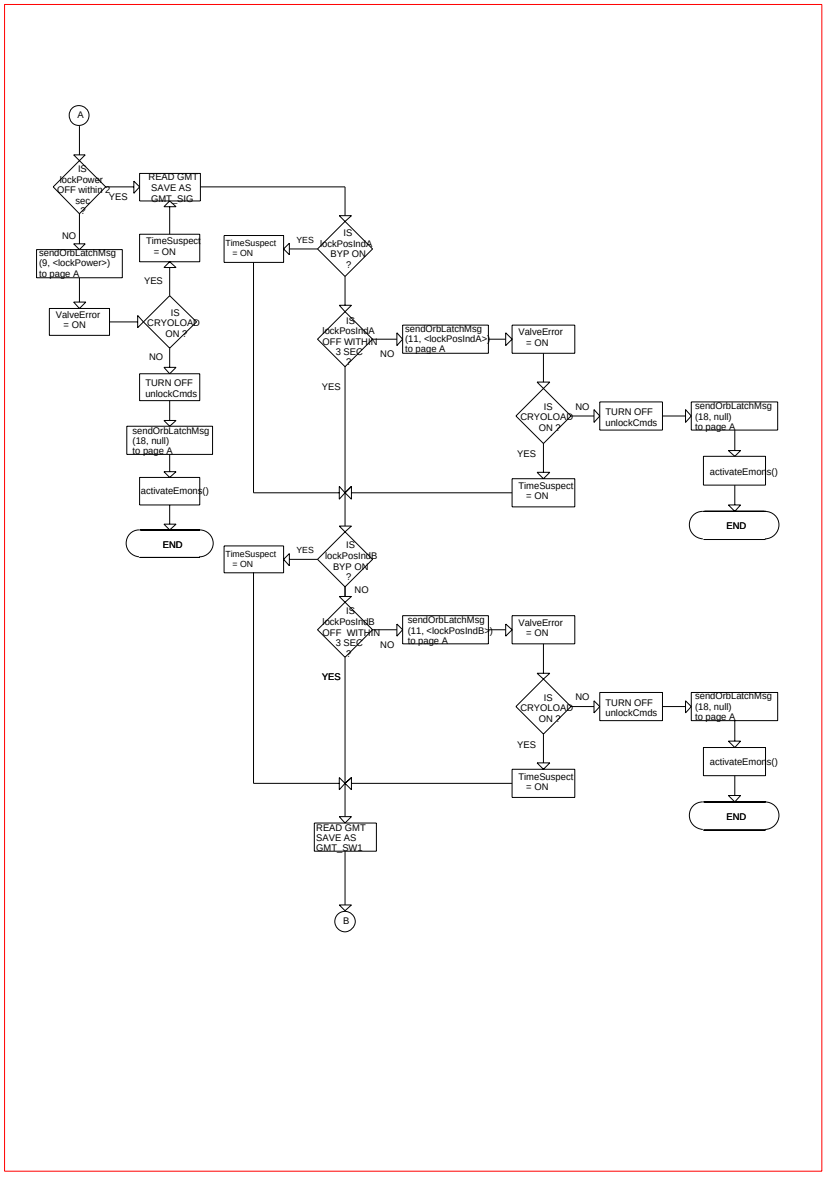
3.1.16.2.2.1. Descriptive Text

The flow charts contained below describe the logic to be used to code the method. The implementation of the details is left up to the coders and is acceptable as long as the overall logic is maintained. It is expected that cycle timing will be accomplished using the most accurate source. All messages will include a time stamp, and the appropriate FD description. ValveError and TimeSuspect are initialized to OFF and Valve cycle times are set to 0 prior to starting valve cycle methods. If ValveError gets set ON during method run, cycle time FD data health should be set to invalid to aid in identifying suspect cycle data in trend analysis.

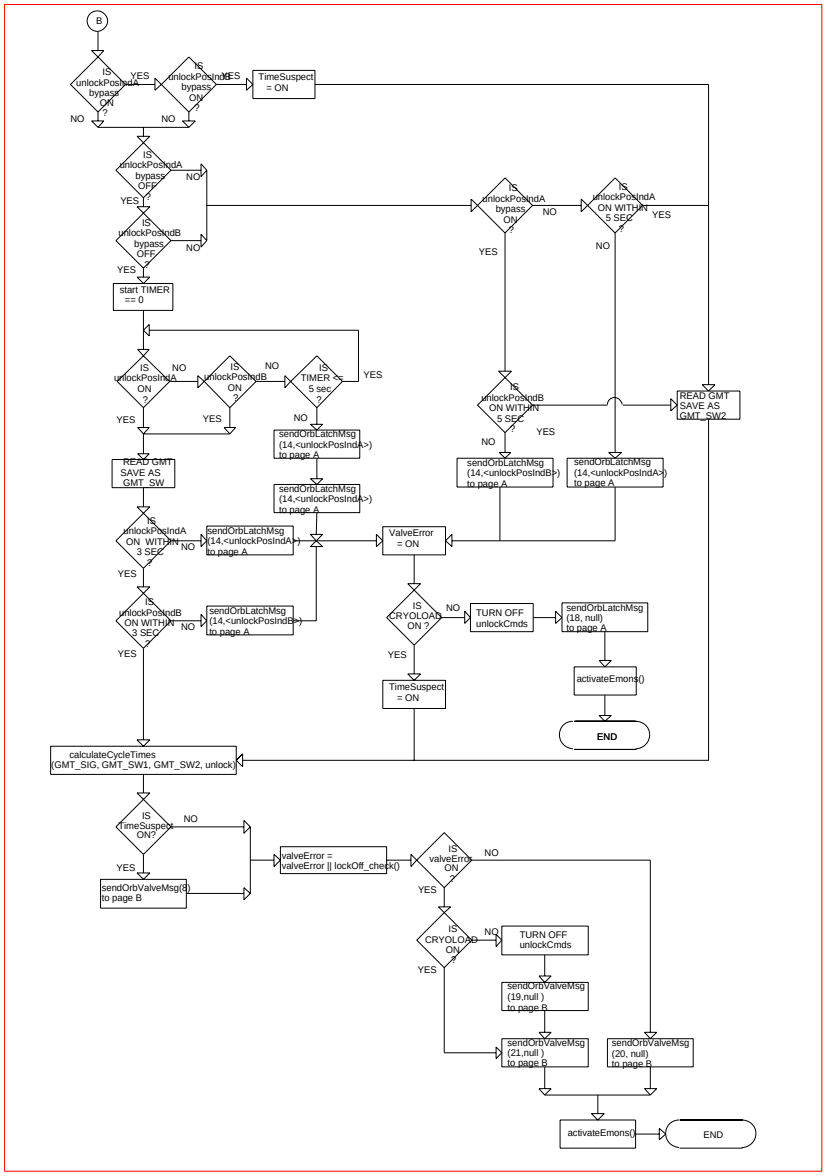
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



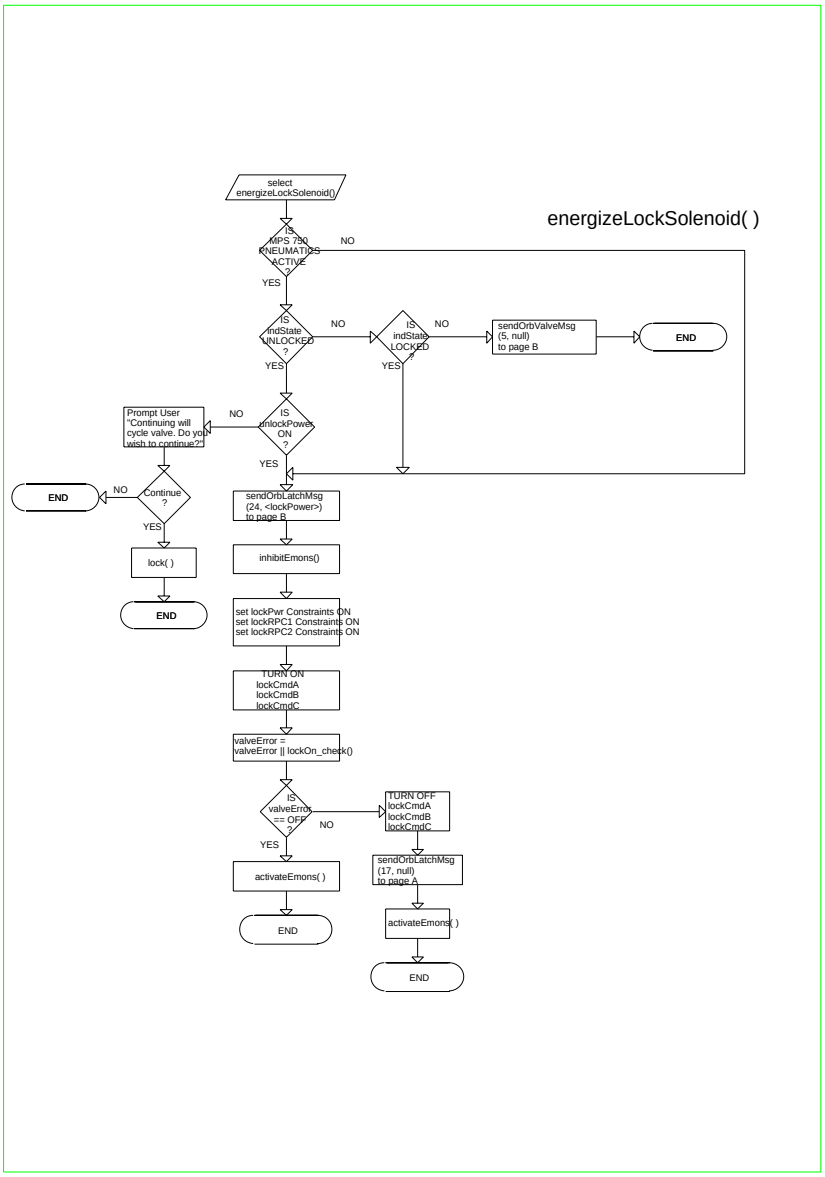
3.1.16.2.3. energizeLockSolenoid()

3.1.16.2.3.1. Descriptive Text

The flow charts contained below describe the logic to be used to code the method. The implementation of the details is left up to the coders and is acceptable as long as the overall logic is maintained. It is expected that cycle timing will be accomplished using the most accurate source. All messages will include a time stamp, and the appropriate FD description.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.16.2.3.2. Sequence



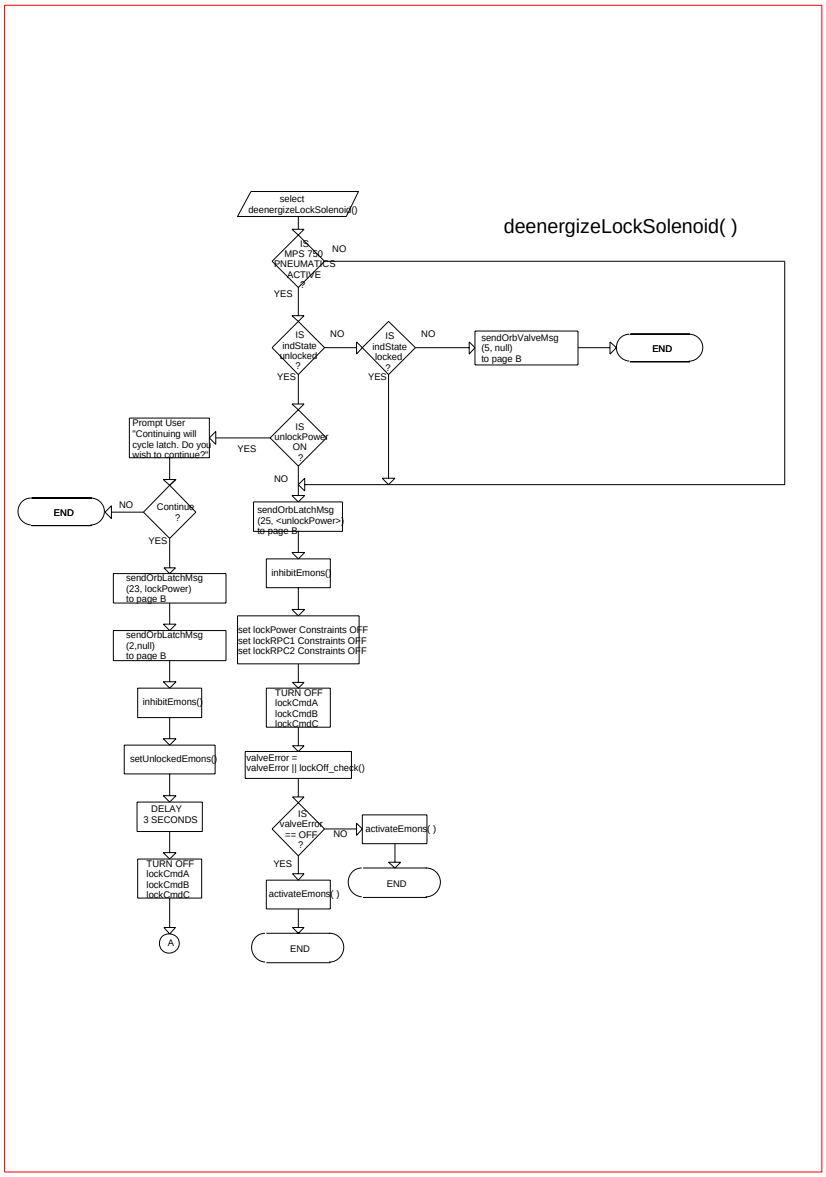
3.1.16.2.4. deenergizeLockSolenoid()

3.1.16.2.4.1. Descriptive Text

The flow charts contained below describe the logic to be used to code the method. The implementation of the details is left up to the coders and is acceptable as long as the overall logic is maintained. It is expected that cycle timing will be accomplished using the most accurate source. All messages will include a time stamp, and the appropriate FD description.

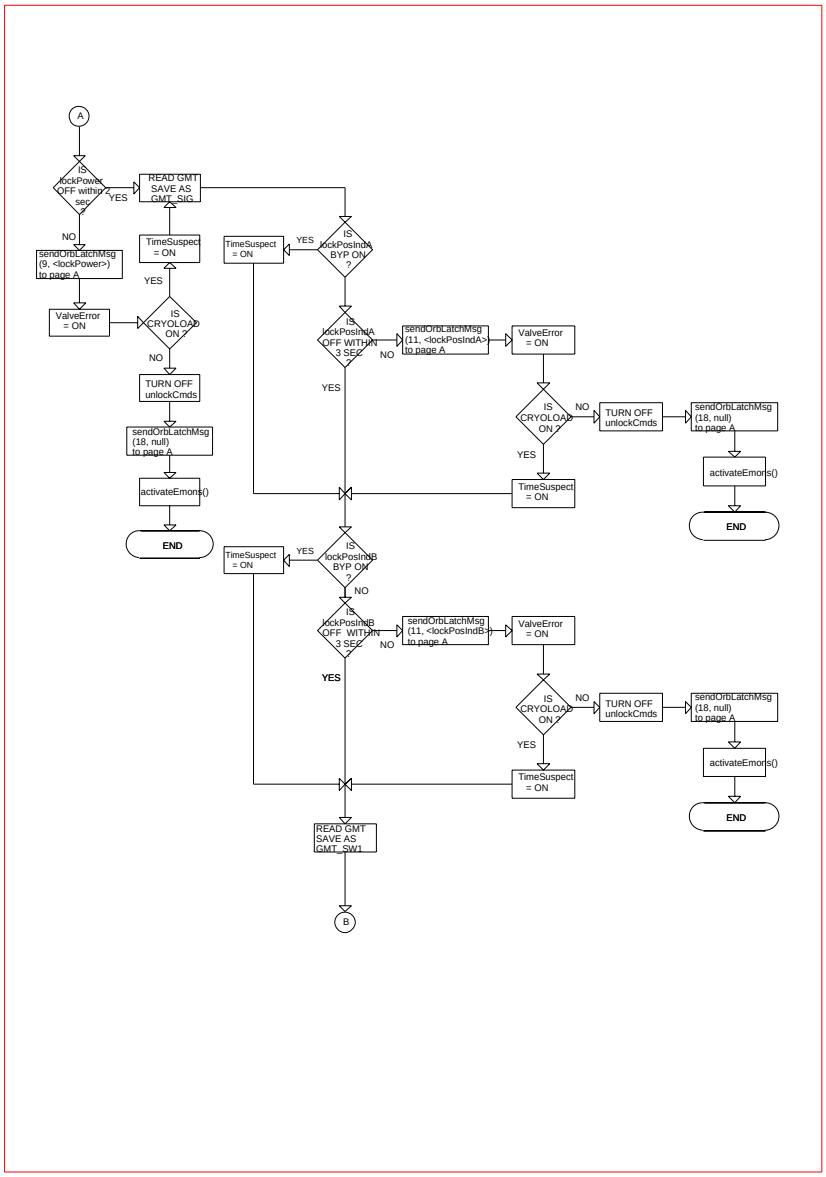
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.16.2.4.2. Sequence (page 1 of 3)



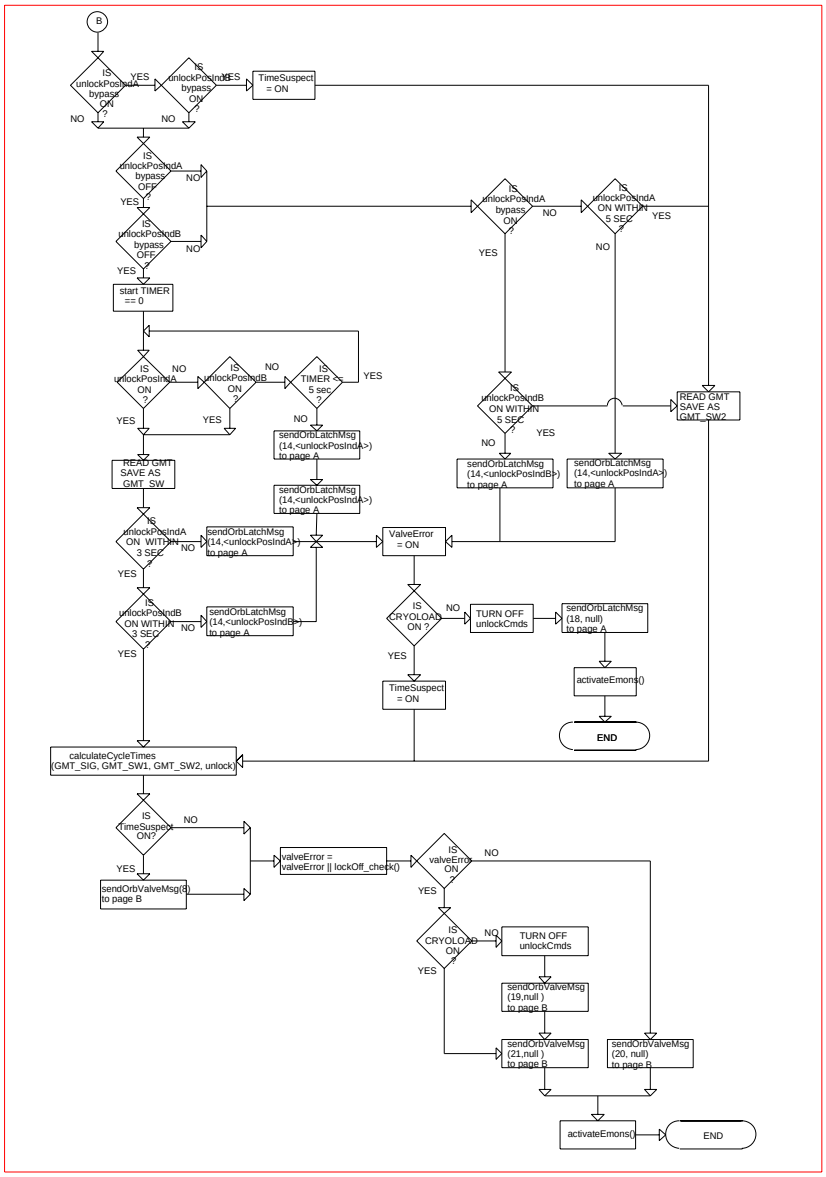
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.16.2.4.3. Sequence (page 2 of 3)



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.16.2.4.4. Sequence (page 3 of 3)



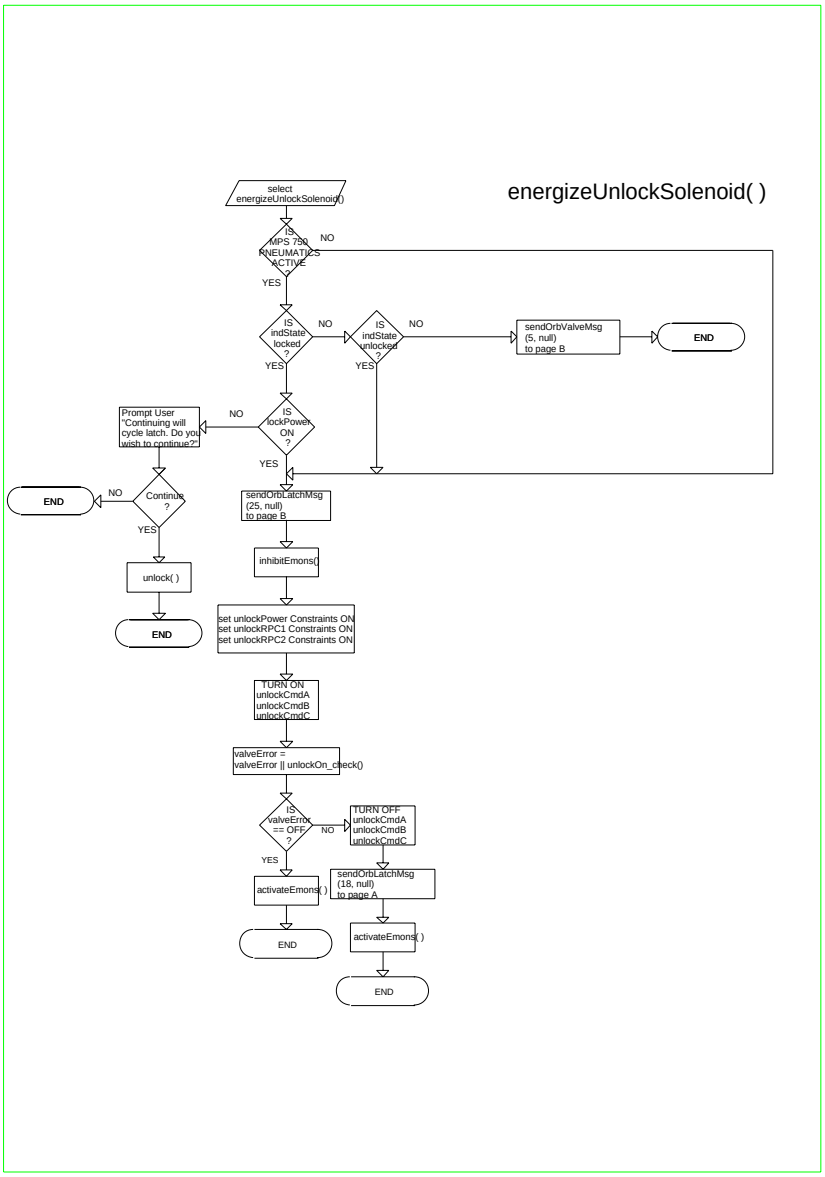
3.1.16.2.5. energizeUnlockSolenoid()

3.1.16.2.5.1. Descriptive Text

The flow charts contained below describe the logic to be used to code the method. The implementation of the details is left up to the coders and is acceptable as long as the overall logic is maintained. It is expected that cycle timing will be accomplished using the most accurate source. All messages will include a time stamp, and the appropriate FD description.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.16.2.5.2. Sequence



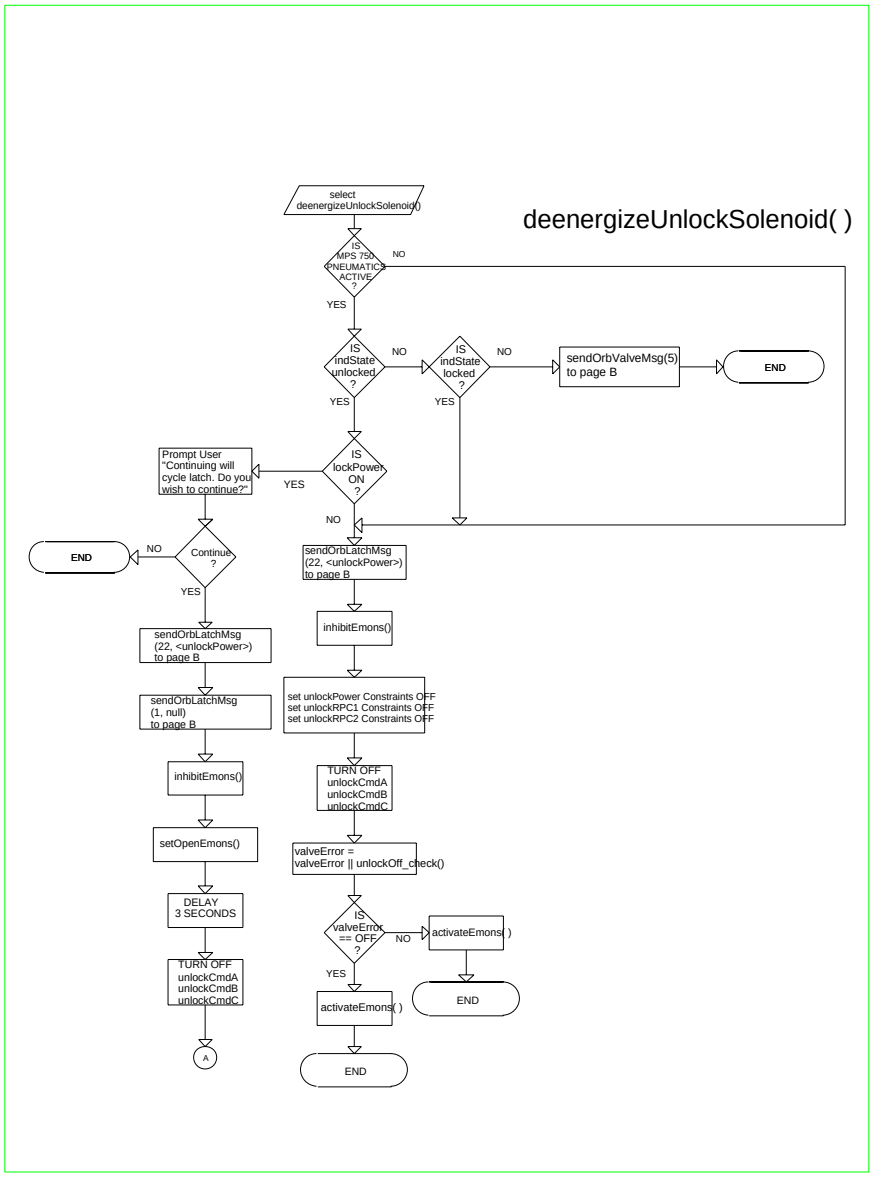
3.1.16.2.6. deenergizeUnlockSolenoid()

3.1.16.2.6.1. Descriptive Text

The flow charts contained below describe the logic to be used to code the method. The implementation of the details is left up to the coders and is acceptable as long as the overall logic is maintained. It is expected that cycle timing will be accomplished using the most accurate source. All messages will include a time stamp, and the appropriate FD description.

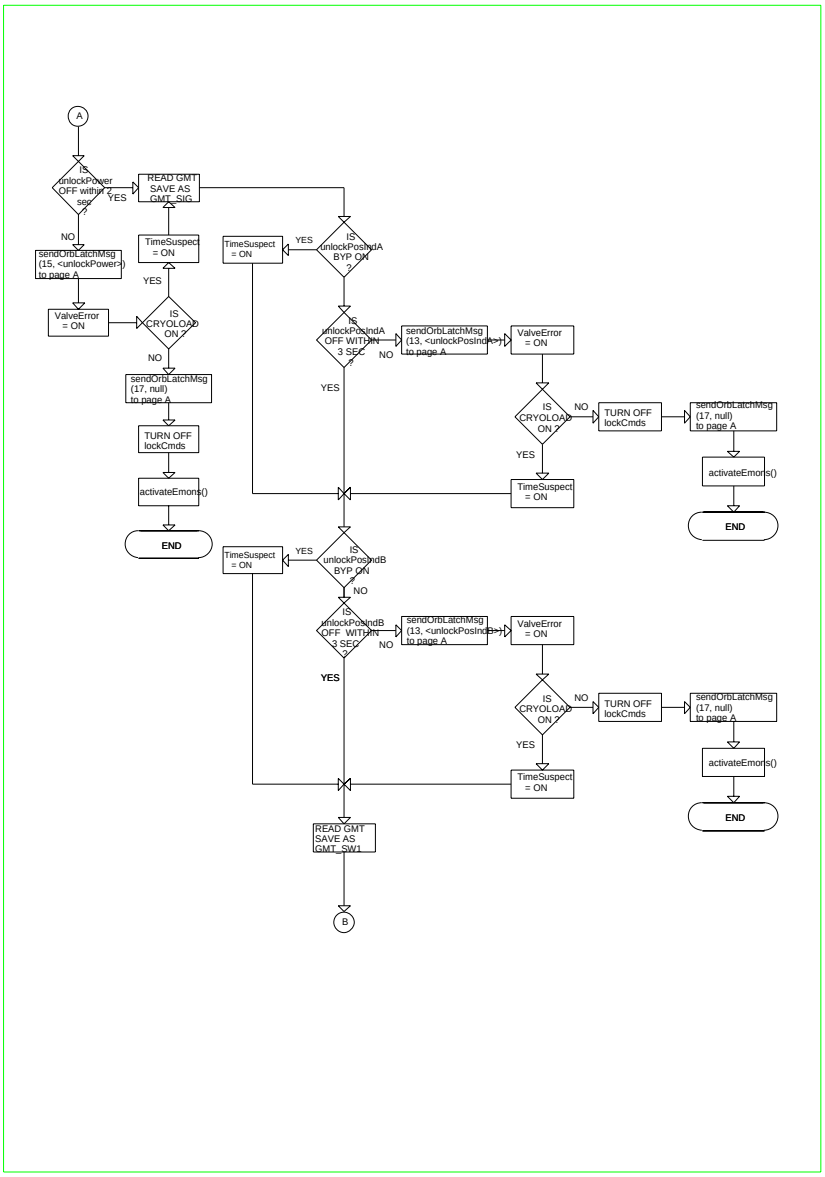
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.16.2.6.2. Sequence (page 1 of 3)



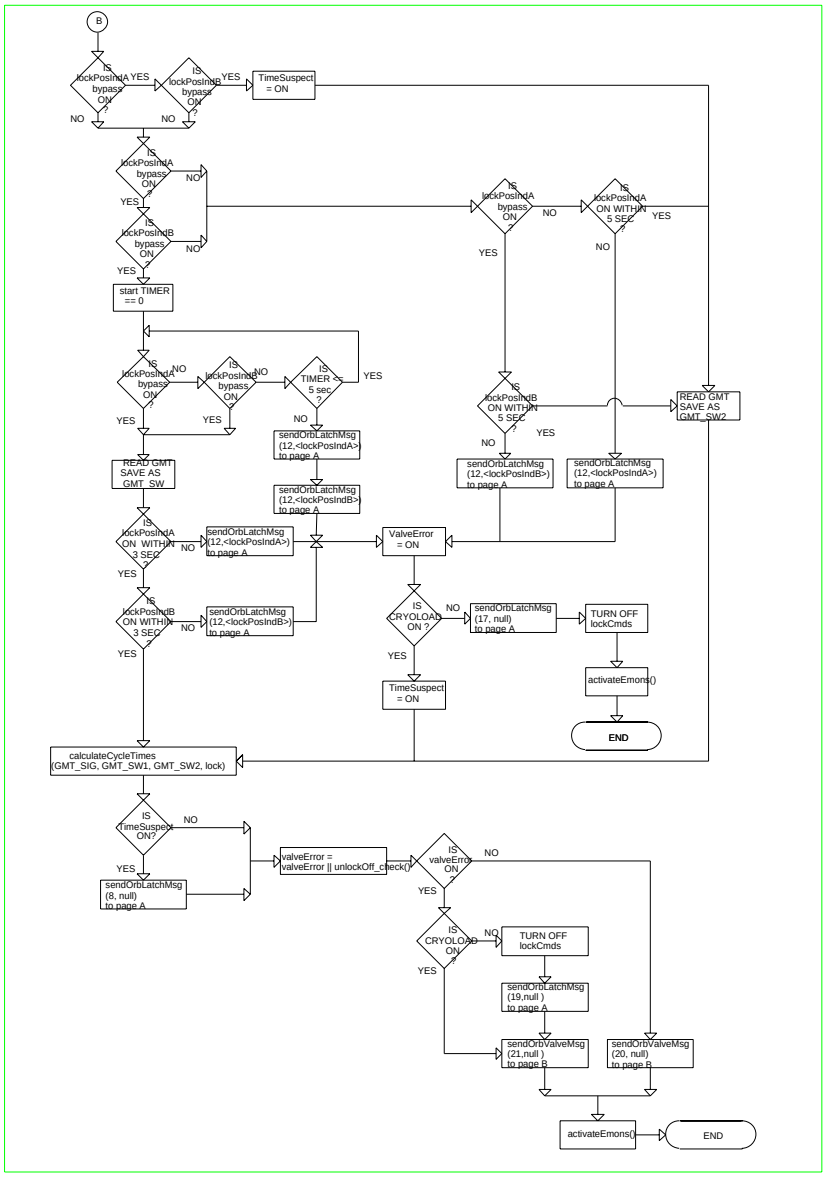
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.16.2.6.3. Sequence (page 2 of 3)



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.16.2.6.4. Sequence (page 3 of 3)

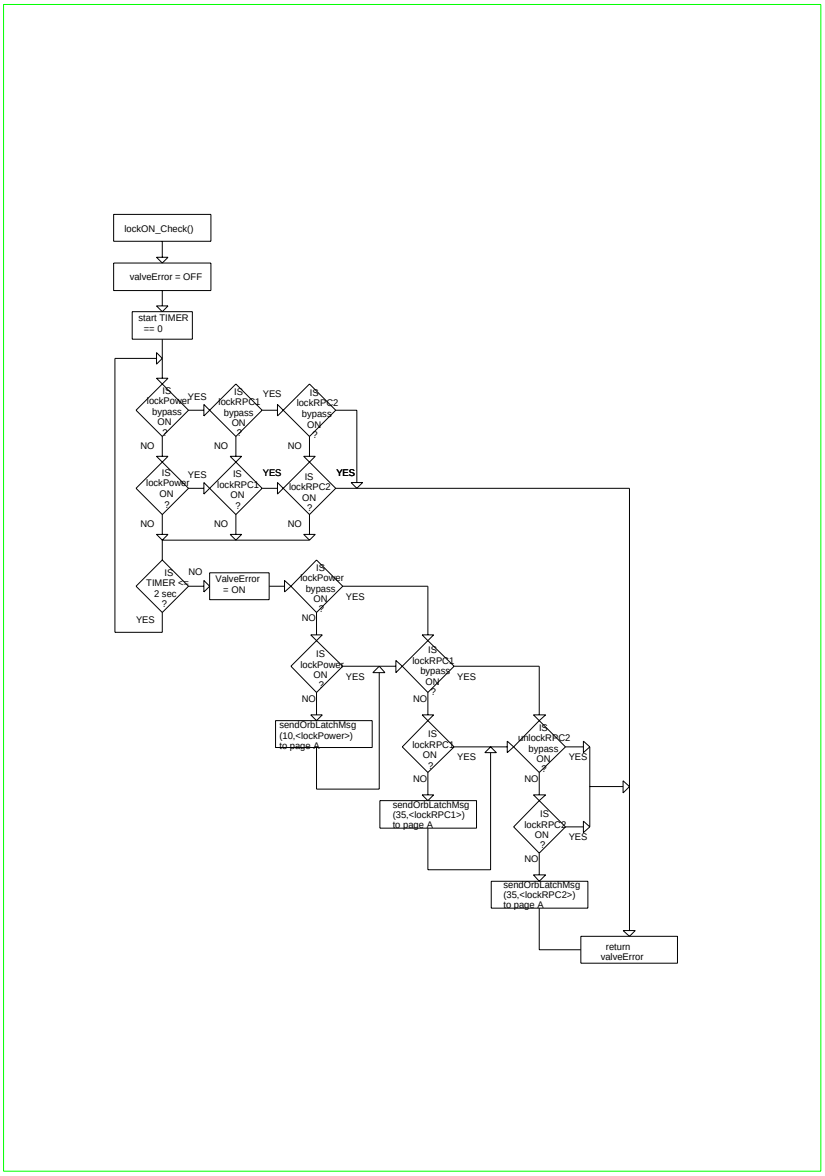


3.1.16.2.7. lockON_check()

3.1.16.2.7.1. Descriptive Text

lockON_check() is a private method that is called by lock, unlock, and the energize and deenergize methods of the class. The lockOn_check method verifies that the lock powers and RPCs are ON, and provides appropriate messaging if errors are encountered. If all powers and RPCs are either ON or are Bypassed then valveError equal to OFF is returned to the calling method. If errors are encountered the valveError equal to ON is returned. The flow charts contained below describe the logic to be used to code the method. The implementation of the details is left up to the coders and is acceptable as long as the overall logic is maintained. All messages will include a time stamp, and the appropriate FD description.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



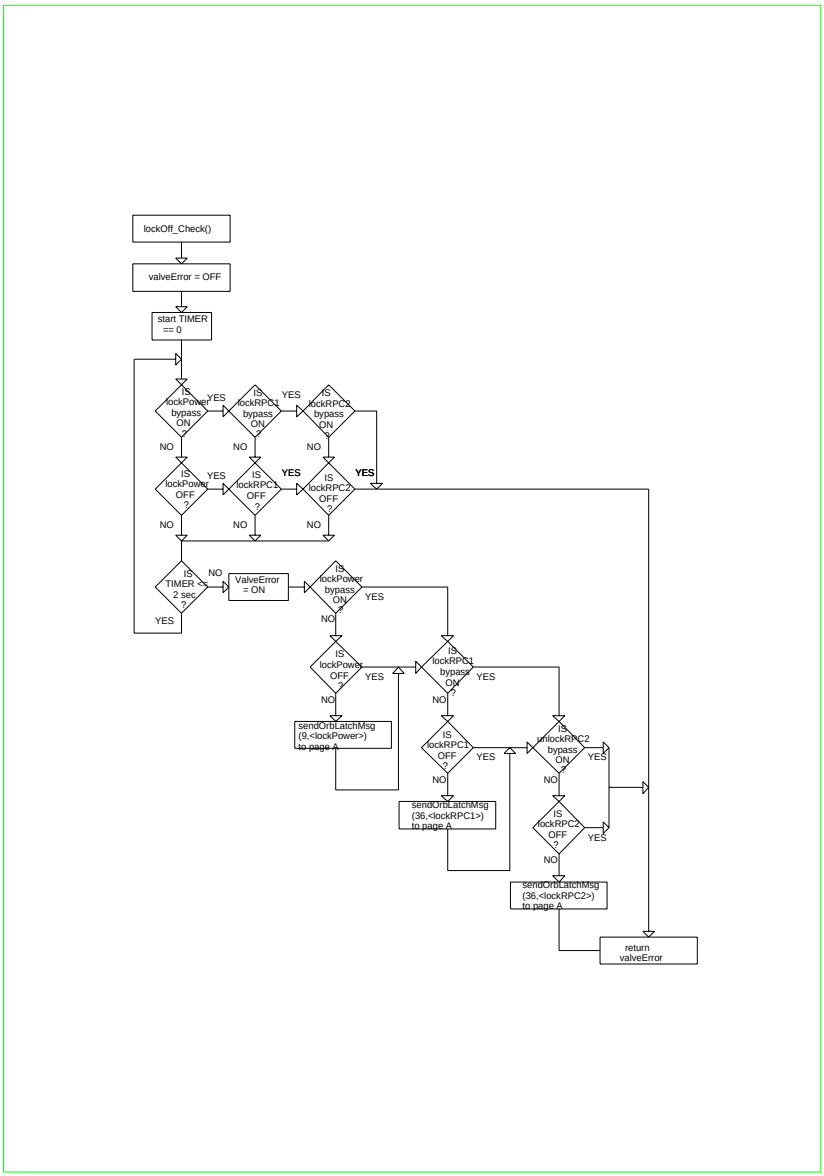
3.1.16.2.8. lockOff_check()

3.1.16.2.8.1. Descriptive Text

lockOff_check() is a private method that is called by lock, unlock, and the energize and deenergize methods of the class. The lockOff_check method verifies that the lock powers and RPCs are OFF, and provides appropriate messaging if errors are encountered. If all powers and RPCs are either OFF or are Bypassed then valveError equal to OFF is returned to the calling method. If errors are encountered the valveError equal to ON is returned. The flow charts contained below describe the logic to be used to code the method. The implementation of the details is left up to the coders and is acceptable as long as the overall logic is maintained. All messages will include a time stamp, and the appropriate FD description.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.1.16.2.8.2. Sequence

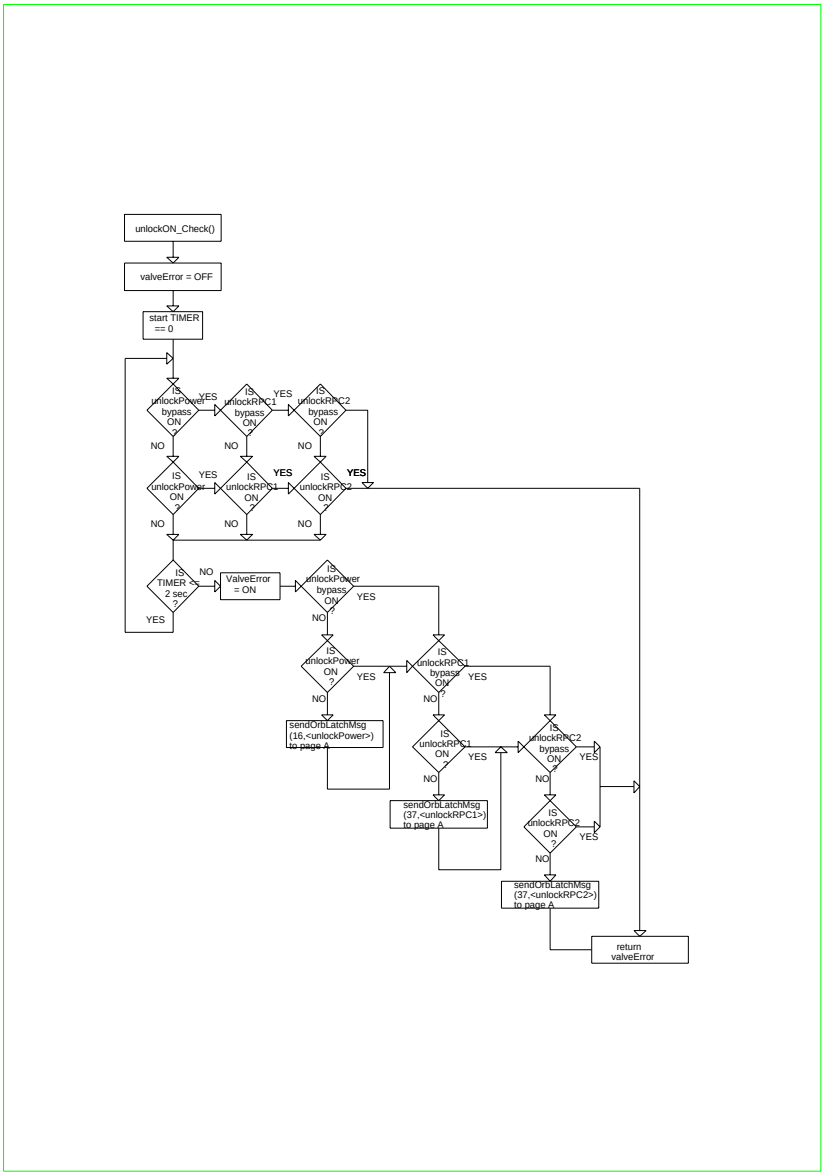


3.1.16.2.9. unlockON_check()

3.1.16.2.9.1. Descriptive Text

unlockON_check() is a private method that is called by lock, unlock, and the energize and deenergize methods of the class. The unlockOn_check method verifies that the unlock powers and RPCs are ON, and provides appropriate messaging if errors are encountered. If all powers and RPCs are either ON or are Bypassed then valveError equal to OFF is returned to the calling method. If errors are encountered the valveError equal to ON is returned. The flow charts contained below describe the logic to be used to code the method. The implementation of the details is left up to the coders and is acceptable as long as the overall logic is maintained. All messages will include a time stamp, and the appropriate FD description.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

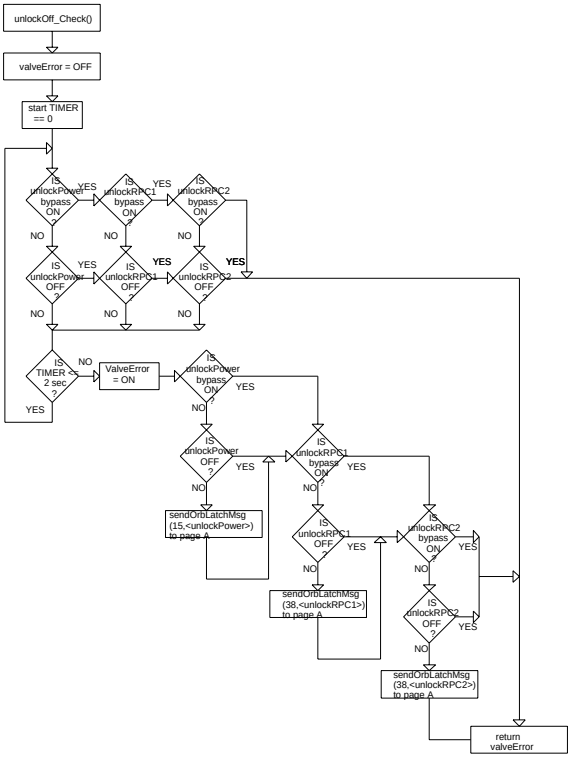


3.1.16.2.10. unlockOff_check()

3.1.16.2.10.1. Descriptive Text

unlockOff_check() is a private method that is called by lock, unlock, and the energize and deenergize methods of the class. The unlockOff_check method verifies that the unlock powers and RPCs are OFF, and provides appropriate messaging if errors are encountered. If all powers and RPCs are either OFF or are Bypassed then valveError equal to OFF is returned to the calling method. If errors are encountered the valveError equal to ON is returned. The flow charts contained below describe the logic to be used to code the method. The implementation of the details is left up to the coders and is acceptable as long as the overall logic is maintained. All messages will include a time stamp, and the appropriate FD description.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



3.1.16.2.11. determineIndState()

3.1.16.2.11.1. Descriptive Text

This method will determine the valve's position based on its position indicators and associated indicator bypasses. The return value will be an enumerated type with possible values of LOCKED, UNLOCKED, TRANSITION, and ERROR.

3.1.16.2.11.2. Effectivity

Operates on change data.

3.1.16.2.11.3. Sequence

All cases not listed in the table are ERROR conditions.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

locked Indicator A	locked Indicator A Byp	locked Indicator B	locked Indicator B Byp	close Indicator A	close Indicator A BYP	close Indicator B	close Indicator B BYP	Ind
----	ON	ON	OFF	OFF	OFF	OFF	OFF	**loc**
----	ON	ON	OFF	OFF	OFF	----	ON	**loc**
----	ON	ON	OFF	----	ON	OFF	OFF	**loc**
ON	OFF	----	ON	OFF	OFF	OFF	OFF	**loc**
ON	OFF	----	ON	OFF	OFF	----	ON	**loc**
ON	OFF	----	ON	----	ON	OFF	OFF	**loc**
ON	OFF	ON	OFF	OFF	OFF	OFF	OFF	**loc**
ON	OFF	ON	OFF	OFF	OFF	----	ON	**loc**
ON	OFF	ON	OFF	----	ON	OFF	OFF	**loc**
----	ON	----	ON	OFF	OFF	OFF	OFF	**loc**
----	ON	----	ON	OFF	OFF	----	ON	**loc**
----	ON	----	ON	----	ON	OFF	OFF	**loc**
----	ON	ON	OFF	----	ON	----	ON	**loc**
ON	OFF	----	ON	----	ON	----	ON	**loc**
ON	OFF	ON	OFF	----	ON	----	ON	**loc**
OFF	OFF	OFF	OFF	----	ON	ON	OFF	**unlo**
OFF	OFF	OFF	OFF	ON	OFF	----	ON	**unlo**
OFF	OFF	OFF	OFF	ON	OFF	ON	OFF	**unlo**
OFF	OFF	----	ON	----	ON	ON	OFF	**unlo**
OFF	OFF	----	ON	ON	OFF	----	ON	**unlo**
OFF	OFF	----	ON	ON	OFF	ON	OFF	**unlo**
----	ON	OFF	OFF	----	ON	ON	OFF	**unlo**
----	ON	OFF	OFF	ON	OFF	----	ON	**unlo**
----	ON	OFF	OFF	ON	OFF	ON	OFF	**unlo**
OFF	OFF	OFF	OFF	----	ON	----	ON	**unlo**
OFF	OFF	----	ON	----	ON	----	ON	**unlo**
----	ON	OFF	OFF	----	ON	----	ON	**unlo**
----	ON	----	ON	----	ON	ON	OFF	**unlo**
----	ON	----	ON	ON	OFF	----	ON	**unlo**
----	ON	----	ON	ON	OFF	ON	OFF	**unlo**
OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF	**trans**
OFF	OFF	OFF	OFF	OFF	OFF	----	ON	**trans**
OFF	OFF	OFF	OFF	----	ON	OFF	OFF	**trans**
OFF	OFF	----	ON	OFF	OFF	OFF	OFF	**trans**
OFF	OFF	----	ON	OFF	OFF	----	ON	**trans**
OFF	OFF	----	ON	----	ON	OFF	OFF	**trans**
----	ON	OFF	OFF	OFF	OFF	OFF	OFF	**trans**
----	ON	OFF	OFF	OFF	OFF	----	ON	**trans**
----	ON	OFF	OFF	----	ON	OFF	OFF	**trans**

3.1.16.2.12. determineUnlockSolCmdState()

3.1.16.2.12.1. Descriptive Text

This method will determine the unlock solenoid's commanded state based on the two unlock RPCs, the unlock power, and the associated bypasses. The return value will be an enumerated type with possible values of ENERGIZED, DEENERGIZED and ERROR.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.16.2.12.2. Effectivity

Operates on change data.

3.1.16.2.12.3. Sequence

Unlock Solenoid Command state						
unlock Power Ind	**unlock power bypass**	**unlock RPC_A**	**unlock RPC_A Bypass**	**unlock RPC_B**	**unlock RPC_B Bypass**	**unlock solenoid CMD State**
OFF	OFF	OFF	OFF	OFF	OFF	deenergized
OFF	OFF	OFF	OFF	----	ON	deenergized
OFF	OFF	----	ON	OFF	OFF	deenergized
OFF	OFF	----	ON	----	ON	deenergized
----	ON	OFF	OFF	OFF	OFF	deenergized
----	ON	OFF	OFF	----	ON	deenergized
----	ON	----	ON	OFF	OFF	deenergized
----	ON	----	ON	ON	OFF	energized
----	ON	ON	OFF	----	ON	energized
----	ON	ON	OFF	ON	OFF	energized
ON	OFF	----	ON	----	ON	energized
ON	OFF	----	ON	ON	OFF	energized
ON	OFF	ON	OFF	----	ON	energized
ON	OFF	ON	OFF	ON	OFF	energized
OFF	OFF	OFF	OFF	ON	OFF	error
OFF	OFF	----	ON	ON	OFF	error
OFF	OFF	ON	OFF	OFF	OFF	error
OFF	OFF	ON	OFF	----	ON	error
OFF	OFF	ON	OFF	ON	OFF	error
----	ON	OFF	OFF	ON	OFF	error
----	ON	----	ON	----	ON	error
----	ON	ON	OFF	OFF	OFF	error
ON	OFF	OFF	OFF	OFF	OFF	error
ON	OFF	OFF	OFF	----	ON	error
ON	OFF	OFF	OFF	ON	OFF	error
ON	OFF	----	ON	OFF	OFF	error
ON	OFF	ON	OFF	OFF	OFF	error

3.1.16.2.13. determineLockSolCmdState()

3.1.16.2.13.1. Descriptive Text

This method will determine the lock solenoid's commanded state based on the two lock RPCs, the lock power, and the associated bypasses. The return value will be an enumerated type with possible values of ENERGIZED, DEENERGIZED and ERROR.

3.1.16.2.13.2. Effectivity

Operates on change data.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.16.2.13.3. Sequence

Locking Solenoid Command State						
lock power	**lock power bypass**	**lock RPC_A**	**lock RPC_A Bypass**	**lock RPC_B**	**lock RPC_B Bypass**	**locking solenoid CMD State**
OFF	OFF	OFF	OFF	OFF	OFF	deenergized
OFF	OFF	OFF	OFF	----	ON	deenergized
OFF	OFF	----	ON	OFF	OFF	deenergized
OFF	OFF	----	ON	----	ON	deenergized
----	ON	OFF	OFF	OFF	OFF	deenergized
----	ON	OFF	OFF	----	ON	deenergized
----	ON	----	ON	OFF	OFF	deenergized
----	ON	----	ON	ON	OFF	energized
----	ON	ON	OFF	----	ON	energized
----	ON	ON	OFF	ON	OFF	energized
ON	OFF	----	ON	----	ON	energized
ON	OFF	----	ON	ON	OFF	energized
ON	OFF	ON	OFF	----	ON	energized
ON	OFF	ON	OFF	ON	OFF	energized
OFF	OFF	OFF	OFF	ON	OFF	error
OFF	OFF	----	ON	ON	OFF	error
OFF	OFF	ON	OFF	OFF	OFF	error
OFF	OFF	ON	OFF	----	ON	error
OFF	OFF	ON	OFF	ON	OFF	error
----	ON	OFF	OFF	ON	OFF	error
----	ON	----	ON	----	ON	error
----	ON	ON	OFF	OFF	OFF	error
ON	OFF	OFF	OFF	OFF	OFF	error
ON	OFF	OFF	OFF	----	ON	error
ON	OFF	OFF	OFF	ON	OFF	error
ON	OFF	----	ON	OFF	OFF	error
ON	OFF	ON	OFF	OFF	OFF	error

3.1.16.2.14. determineCmdState()

3.1.16.2.14.1. Descriptive Text

This method will determine the valve's overall commanded state based on closeSolCmdState and openSolCmdState. The return value will be an enumerated type with possible values of CMD_LOCKED, CMD_UNLOCKED, UNCOMMANDED, CMD_SNUBBED, and ERROR.

3.1.16.2.14.2. Effectivity

Operates on change data.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.16.2.14.3. Sequence

Combined Command State			
unlock_CMDS \ lock_CMDS	ENERGIZED	DEENERGIZED	ERROR
DEENERGIZED	CMD_LOCKED	UNCOMMANDED	ERROR
ENERGIZED	CMD_SNUBBED	CMD_UNLOCKED	ERROR
ERROR	ERROR	ERROR	ERROR

3.1.16.2.15. getIndState()//

3.1.16.2.15.1. Descriptive Text

This method return latch ind state.

3.1.16.2.15.2. Sequence

```
getIndState()
{
        return indState;
}
```

3.1.16.2.16. determineState()

3.1.16.2.16.1. Descriptive Text

This method will return the Enumerated Indicator State of the valve

3.1.16.2.16.2. Sequence

State (overall)					
cmd pos \	cmd lock	cmd unlock	cmd snubbed	uncommanded	error
lock	lock	lock error	lock_error	lock_error	error
unlock	unlock error	unlock	unlock_error	unlock_error	error
transition	transition	transition	error	error	error
error	error	error	error	error	error

3.1.16.2.17. getUnlockSolCmdState()

3.1.16.2.17.1. Descriptive Text

This method will return the Enumerated Command State of the latch unlock solenoid.

3.1.16.2.17.2. Sequence

```
getUnlockSolCmdState()
{
        return unlockSolCmdState;
}
```

3.1.16.2.18. getLockSolCmdState()

3.1.16.2.18.1. Descriptive Text

This method will return the Enumerated Command State of the latch locking solenoid.

3.1.16.2.18.2. Sequence

```
getLockSolCmdState()
{
        return lockSolCmdState;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
}
```

3.1.16.2.19. getCmdState()

3.1.16.2.19.1. Descriptive Text

This method will return the Enumerated Command State of the latch.

3.1.16.2.19.2. Sequence

```
getCmdState()
{
        return cmdState;
}
```

3.1.16.2.20. getState()

3.1.16.2.20.1. Descriptive Text

This method will return the overall Enumerated State of the latch.

3.1.16.2.20.2. Sequence

```
getState()
{
        return state;
}
```

3.1.16.2.21. autoCycle()

3.1.16.2.21.1. Descriptive Text

The following will be called by the Panel Class autoValveCheckout method
to calculate and record the valve cycle times.

3.1.16.2.21.2. Sequence

```
Set autoValveCheckoutError = OFF;

lock();

if valveCycleError is ON, Set autoValveCheckoutError = ON;

unlock();

if valveCycleError is ON, Set autoValveCheckoutError = ON;

Return autoValveCheckoutError;
```

3.1.16.2.22. activateConstraints()

3.1.16.2.22.1. Descriptive Text

This method will activate constraint monitoring of all unbypassed measurement of the latch instance.

3.1.16.2.22.2. Sequence

```
activateConstraints()
        {
        if(<lockPosIndA>.Byp == OFF)
                {
                activate constraint monitoring for <lockPosIndA>
                }
        if(<lockPosIndB>.Byp == OFF)
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
        {
        activate constraint monitoring for <lockPosIndB>
        }
    if(<lockPwr>.Byp == OFF)
        {
        activate constraint monitoring for <lockPwr>
        }
    if(<lockRPCB>.Byp == OFF)
        {
        activate constraint monitoring for <lockRPCB>
        }
    if(<lockRPCC>.Byp == OFF)
        {
        activate constraint monitoring for <lockRPCC>
        }
    if(<unlockPwr>.Byp == OFF)
        {
        activate constraint monitoring for <unlockPwr>
        }
    if(<unlockPosIndA>.Byp == OFF)
        {
        activate constraint monitoring for <unlockPosIndA>
        }
    if(<unlockPosIndB>.Byp == OFF)
        {
        activate constraint monitoring for <unlockPosIndB>
        }
    if(<unlockRPCB>.Byp == OFF)
        {
        activate constraint monitoring for <unlockRPCB>
        }
    if(<unlockRPCC>.Byp == OFF)
        {
        activate constraint monitoring for <unlockRPCC>
        }
```

3.1.16.2.23. inhibitConstraints()

3.1.16.2.23.1. Descriptive Text

This method will inhibit constraint monitoring of all measurements related to the latch instance.

3.1.16.2.23.2. Sequence

```
inhibitConstraints()
    {
    inhibit constraint monitoring of the following:
    <lockPosIndA>
    <lockPosIndB>
    <lockRPCB>
    <lockRPCC>
    <lockPwr>
    <unlockPosIndA>
    <unlockPosIndB>
    <unlockPwr>
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

<unlockRPCB>
<unlockRPCC>
}

3.1.16.2.24. setLockedConstraints()

3.1.16.2.24.1. Descriptive Text

This method changes the constraint criteria of the measurements associated with the latch instance to those expected locked position.

3.1.16.2.24.2. Sequence

```
void setLockedConstraints()
{
        CHANGE <lockPwr> CONSTRAINT CRITERIA TO ON;

      CHANGE <lockPosIndA>  CONSTRAINT CRITERIA TO ON;

      CHANGE <lockPosIndB> CONSTRAINT CRITERIA TO ON;

        CHANGE <lockRPCB> CONSTRAINT CRITERIA TO ON;

      CHANGE <lockRPCC>  CONSTRAINT CRITERIA TO ON;

      CHANGE <unlockPwr> CONSTRAINT CRITERIA TO OFF;

        CHANGE <unlockPosIndA> CONSTRAINT CRITERIA TO OFF;

      CHANGE <unlockPosIndB>  CONSTRAINT CRITERIA TO OFF;

      CHANGE <unlockRPCB> CONSTRAINT CRITERIA TO OFF;

        CHANGE <unlockRPCC> CONSTRAINT CRITERIA TO OFF;


}
```

3.1.16.2.25. setUnlockedConstraints()

3.1.16.2.25.1. Descriptive Text

This method changes the constraint criteria of the measurements associated with the latch instance to those expected in the unlocked position.

3.1.16.2.25.2. Sequence

```
void setUnlockedConstraints()
{
        CHANGE <lockPwr> CONSTRAINT CRITERIA TO OFF;

      CHANGE <lockPosIndA>  CONSTRAINT CRITERIA TO OFF;

      CHANGE <lockPosIndB> CONSTRAINT CRITERIA TO OFF;

        CHANGE <lockRPCB> CONSTRAINT CRITERIA TO OFF;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

CHANGE <lockRPCC> CONSTRAINT CRITERIA TO OFF;

CHANGE <unlockPwr> CONSTRAINT CRITERIA TO ON;

CHANGE <unlockPosIndA> CONSTRAINT CRITERIA TO ON;

CHANGE <unlockPosIndB> CONSTRAINT CRITERIA TO ON;

CHANGE <unlockRPCB> CONSTRAINT CRITERIA TO ON;

CHANGE <unlockRPCC> CONSTRAINT CRITERIA TO ON;

}

3.1.16.2.26. setUnlockedIndConstraints()

3.1.16.2.26.1. Descriptive Text

3.1.16.2.26.2. Sequence

setUnlockedIndConstraints()

{

CHANGE <lockPwr> CONSTRAINT CRITERIA TO OFF;

CHANGE <lockPosIndA> CONSTRAINT CRITERIA TO OFF;

CHANGE <lockPosIndB> CONSTRAINT CRITERIA TO OFF;

CHANGE <lockRPCB> CONSTRAINT CRITERIA TO OFF;

CHANGE <lockRPCC> CONSTRAINT CRITERIA TO OFF;

CHANGE <unlockPwr> CONSTRAINT CRITERIA TO OFF;

CHANGE <unlockPosIndA> CONSTRAINT CRITERIA TO ON;

CHANGE <unlockPosIndB> CONSTRAINT CRITERIA TO ON;

CHANGE <unlockRPCB> CONSTRAINT CRITERIA TO OFF;

CHANGE <unlockRPCC> CONSTRAINT CRITERIA TO OFF;

}

3.1.16.2.27. verifyCommandsOff()

3.1.16.2.27.1. Descriptive Text

This function will verify that both the open and close commands are OFF

3.1.16.2.27.2. Sequence

verifyCommandsOff()

{

lock command status 1 = byteTest4 <lockCMDA>

{

verify lock command status 1 == OFF

else print message to page A : "Byte Test 4 shows", valveID, "lock command" <lockCMDA> "to be ON"

lock command status 2 = byteTest4 <lockCMDB>

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
        {
        verify  lock command status 2 == OFF
        else print message to page A :   "Byte Test 4 shows", valveID, "lock command" <lockCMDB> "to be ON"

lock command status 3 = byteTest4 <lockCMDC>
        {
        verify  lock command status 3 == OFF
        else print message to page A :   "Byte Test 4 shows", valveID, "lock command" <lockCMDC> "to be ON"

unlock command status 1 = byteTest4 <unlockCMDA>
        {
        verify  unlock command status 1 == OFF
        else print message to page A :   "Byte Test 4 shows", valveID, "unlock command" <unlockCMDA> "to be
ON"

unlock command status 2 = byteTest4 <unlockCMDB>
        {
        verify  unlock command status 2 == OFF
        else print message to page A :   "Byte Test 4 shows", valveID, "unlock command" <unlockCMDB> "to be
ON"

unlock command status 3 = byteTest4 <unlockCMDC>
        {
        verify  unlock command status 3 == OFF
        else print message to page A :   "Byte Test 4 shows", valveID, "unlock command" <unlockCMDC> "to be
ON"

        }
{
        verify  lock command status 1 && lock command status 2 && lock command status 3 == OFF
        verify  unlock command status 1 && unlock command status 2 && unlock command status 3 == OFF
        Return TRUE;

        Else return FALSE;

}
end sequence;
```

3.1.16.2.28. sendOrbLatchMsg(int)

3.1.16.2.28.1. Descriptive Text

This method returns generic MPS valve messages to the MPS valve software..

3.1.16.2.28.2. Sequence

```
sendOrbLatchMsg(int MsgNum,  FDname)
        {

        if (MsgNum == 1){msgString =   "Latch is Unlocked, cycling to the locked position."}
        if (MsgNum == 2){msgString =   "Latch is locked, cycling to the unlocked position."}
        if (MsgNum == 3){msgString =  "Latch is already locked, applying locking power."}
        if (MsgNum == 4){msgString =  "Latch is already unlocked, applying unlocking power."}
        if (MsgNum == 5){msgString =   "Latch state must be determinate to run this method with MPS 750
active."}
        if (MsgNum == 6){msgString =   "Energizing Solenoid with 750 pneumatics secured."}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
        if (MsgNum == 7){msgString =   "De-energizing Solenoid with 750 pneumatics secured."}
        if (MsgNum == 8){msgString =   "Latch cycle time is suspect due to failed indicators."}
        if (MsgNum == 9){msgString =   "Locking power,", FDname, " is ON, should be OFF."}
        if (MsgNum == 10){msgString =  "Locking power,", FDname, " is OFF, should be ON."}
        if (MsgNum == 11){msgString =  "Locked indicator,", FDname, " is ON, should be OFF."}
        if (MsgNum == 12){msgString =  "Locked indicator,", FDname, " is OFF, should be ON."}
        if (MsgNum == 13){msgString =  "Unlocked indicator,", FDname, " is ON, should be OFF."}
        if (MsgNum == 14){msgString =  "Unlocked indicator,", FDname, " is OFF, should be ON."}
        if (MsgNum == 15){msgString =  "Unlocked power,", FDname, " is ON, should be OFF."}
        if (MsgNum == 16){msgString =  "Unlocked power,", FDname, " is OFF, should be ON."}
        if (MsgNum == 17){msgString =  "Lock command(s) removed."}
        if (MsgNum == 18){msgString =  "Unlock command(s) removed."}
        if (MsgNum == 19){msgString =  "Lock and Unlock commands removed."}
        if (MsgNum == 20){msgString =  "Latch cycle complete.  No errors."}
        if (MsgNum == 21){msgString =  "Latch cycle errors.  See page A for details."}
        if (MsgNum == 22){msgString =   "Removing Unlocking power,", FDname, "."}
        if (MsgNum == 23){msgString =   "Removing Locking power,", FDname, "."}
        if (MsgNum == 24){msgString =   "Applying Locking power,", FDname, "."}
        if (MsgNum == 25){msgString =   "Applying Unlocking power,", FDname, "."}
        if (MsgNum == 26){msgString =   "Cannot run this method with 750 pneumatics secured."}
        if (MsgNum == 27){msgString =   "Aborting selected method."}
        if (MsgNum == 28){msgString =   "Activating SSME Reg A 750 PSI Pneumatics"}
        if (MsgNum == 29){msgString =   "Activating SSME Reg B 750 PSI Pneumatics"}
        if (MsgNum == 30){msgString =   "The location of the orbiter must be determinate to perform this
method."}
        if (MsgNum == 31){msgString =   "This component cannot be cycled in the OPF.
                                         The solenoids can be energized and deenergized using the appropriate
                                         methods.  Verify that the disconnect GSE is NOT installed and select
                                         the appropriate energize or deenergize solenoid method."}
        if (MsgNum == 32){msgString =   ""
        if (MsgNum == 33){msgString =   "Latch power " & FDname & " has been removed with bypassed
indicators."
        if (MsgNum == 34){msgString =   "Latch power " & FDname & " has been applied with bypassed
indicators."
        if (MsgNum == 35){msgString =   "Lock RPC " & FDname & " is OFF, should be ON."
        if (MsgNum == 36){msgString =   "Lock RPC " & FDname & " is ON, should be OFF."
        if (MsgNum == 37){msgString =   "Unlock RPC " & FDname & " is OFF, should be ON."
        if (MsgNum == 38){msgString =   "Unlock RPC " & FDname & " is ON, should be OFF."
return msgString;

        }
```

3.1.16.3. User Interfaces

3.1.16.4. Safety Requirements

3.1.17. MPS_SolValve_ORB_FCV

3.1.17.1. Description

The MPS flow control valves are used to control the flow of propellant to the external tank (LO2 and LH2). The MPS GO2 flow control valves (FCV's) current configuration is a fixed orifice, with no cycling capability and no active electrical connection. The MPS GH2 FCV's are normally open orificed solenoid valves which are energized

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

to a low flow (more-closed) state. The valves have 2 hi flow commands (which inhibit cycling closed), and 1 closed power indication. Functionally, the GH2 valves serve to maintain ET LH2 tank ullage pressure during ascent by throttling the flow of GH2 from the SSMEs to the ET. The MPS ETOI point sensor electronics assembly contains the electronics to maintain the tank ullage pressure in the specified pressure band, and cycle each GH2 FCV based on the No. 1, 2, or 3 ET GH2 ullage pressure measurement (respectively). For ground test and check-out, the point sensor electronics utilizes check-out commands to simulate tank pressure input into the necessary logic circuits for cycling of the FCVs (which includes the electronics which still provide the capability to cycle the GO2 FCV's if they're ever re-activated). A timer is required to verify OMRS powered-On time limitations are maintained.

3.1.17.1.1. Instance Index

Instance	Display User	Other Users
MPS_E1FCV	MPS	
MPS_E2FCV	MPS	
MPS_E3FCV	MPS	

3.1.17.1.2. Class Diagram

<insert class diagram here>

3.1.17.1.3. Attributes and Aggregations

Attribute	Data Type	Initial Value	Publish / Persistence	Client Access	Description
gh2ClosePwr	DM_ON_OFF			R	
go2ClosePwr	DM_ON_OFF			R	
powerOnTime					
hiFlowCmdA	DS_ON_OFF			R,W	
hiFlowCmdB	DS_ON_OFF			R,W	

3.1.17.1.4. Operations Index

Operation	Description
energize()	This method performs the necessary functions to energize the MPS GH2/GO2 flow control valves. This method calls associated class MPS_ORB_ETOI for setting of shared ETOI check-out commands.
deenergize()	This method performs the necessary functions to deenergize the MPS GH2/GO2 flow control valves. This method calls associated class MPS_ORB_ETOI for setting of shared ETOI check-out commands.
determineCmdState()	This method determines the FCV solenoid valve command state based on power indication and returns an enumerated state based on the logic table. Possible states are:

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

	Energized Deenergized Error
getCmdState()	This method returns the enumerated command state of the FCV valves.
determineState()	This method determines the FCV solenoid valve state based on power indication and returns an enumerated state based on the logic table. Possible states are: Open Closed Error
getState()	This method returns the enumerated state of the FCVs.
startMonitorOnTime()	Initiates or continues tracking of FCV powered-ON time.
termMonitorOnTime()	Terminates tracking of FCV 'powered'-ON time.
monitorOnTime()	This method tracks FCV powered time within any 3 hour span to verify FCV does not remain energized for more than 20 minutes within that span.
verifyCmdsOff()	This method verifies byteTest4() results show valve commands presently OFF.
activateGh2Constraints()	This method activates constraint monitoring on associated GH2 FDs.
activateGo2Constraints()	This method activates constraint monitoring on associated GO2 FDs.
setEnergizedGh2Constraints()	This method sets-up constraint states for a set (GH2) of energized FCV's.
setEnergizedGo2Constraints()	This method sets-up constraint states for a set (GO2) of energized FCV's.
setDeenergizedGh2Constraints()	This method sets-up constraint states for a set (GH2) of deenergized FCV's.
setDeenergizedGo2Constraints()	This method sets-up constraint states for a set (GO2) of deenergized FCV's.
inhibitGh2Constraints()	This method terminates constraint monitoring on associated GH2 FDs.
inhibitGo2Constraints()	This method terminates constraint monitoring on associated GO2 FDs.
turnOnHiFlowCmdA()	This method sets the hiFlowCmdA to on in support of test & checkout
turnOffHiFlowCmdA()	This method sets the hiFlowCmdA to off in support of test & checkout
turnOnHiFlowCmdB()	This method sets the hiFlowCmdB to on in support of test & checkout
turnOffHiFlowCmdB()	This method sets the hiFlowCmdB to off in support of test & checkout

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

#verifyTminus3BT4	
#verifyTminus6BT4	

3.1.17.1.5. Associations

Association	Type	Class	Description
MPS_ETOICNTL		MPS_ET_OI_CNTL	Associated class required for shared command maintenance.

3.1.17.2. State Description

3.1.17.2.1. State Diagram

<insert state diagram here>

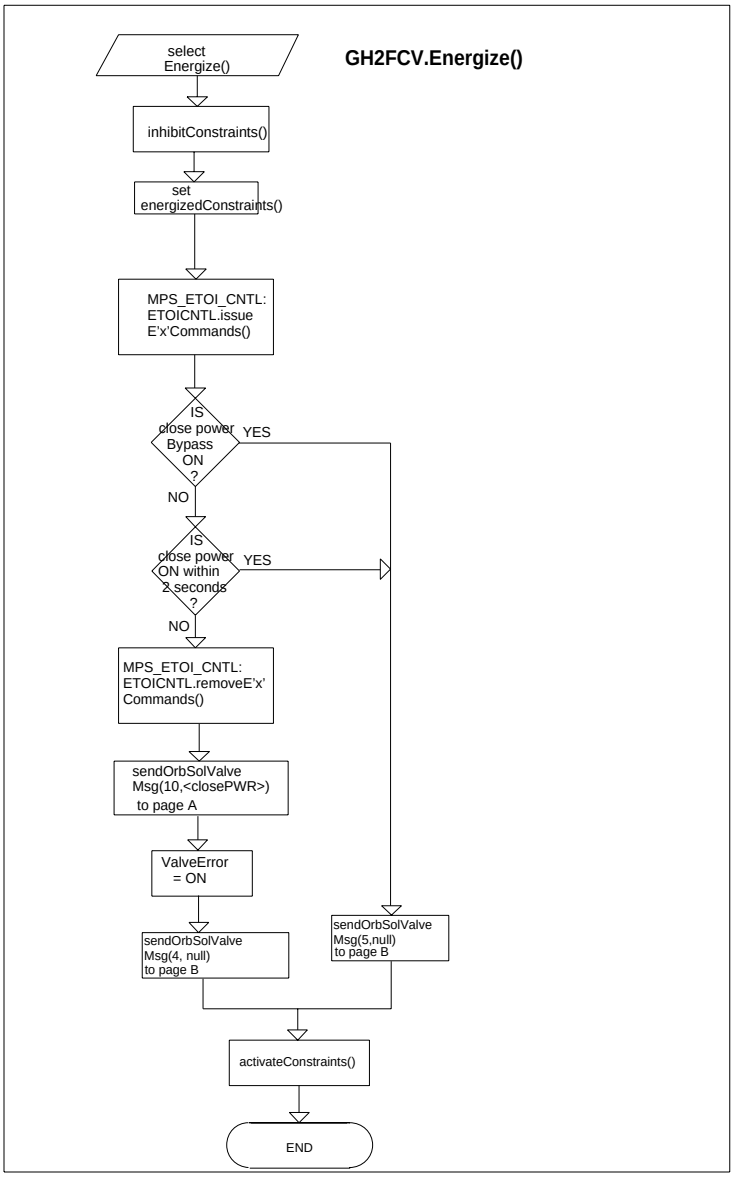
3.1.17.2.2. State Table

3.1.17.3. Operations

E1FCV has an association to MPS_ET_OI_CNTL which is called by this method in order to run the 'issueE1Gh2FcvCmds' method, which will issue commands to energize the engine 1 GH2 FCV. In addition, the association provides for a 'removeE1Gh2FcvCmds' method that deenergizes the valve. E2 and E3 have similar methods.

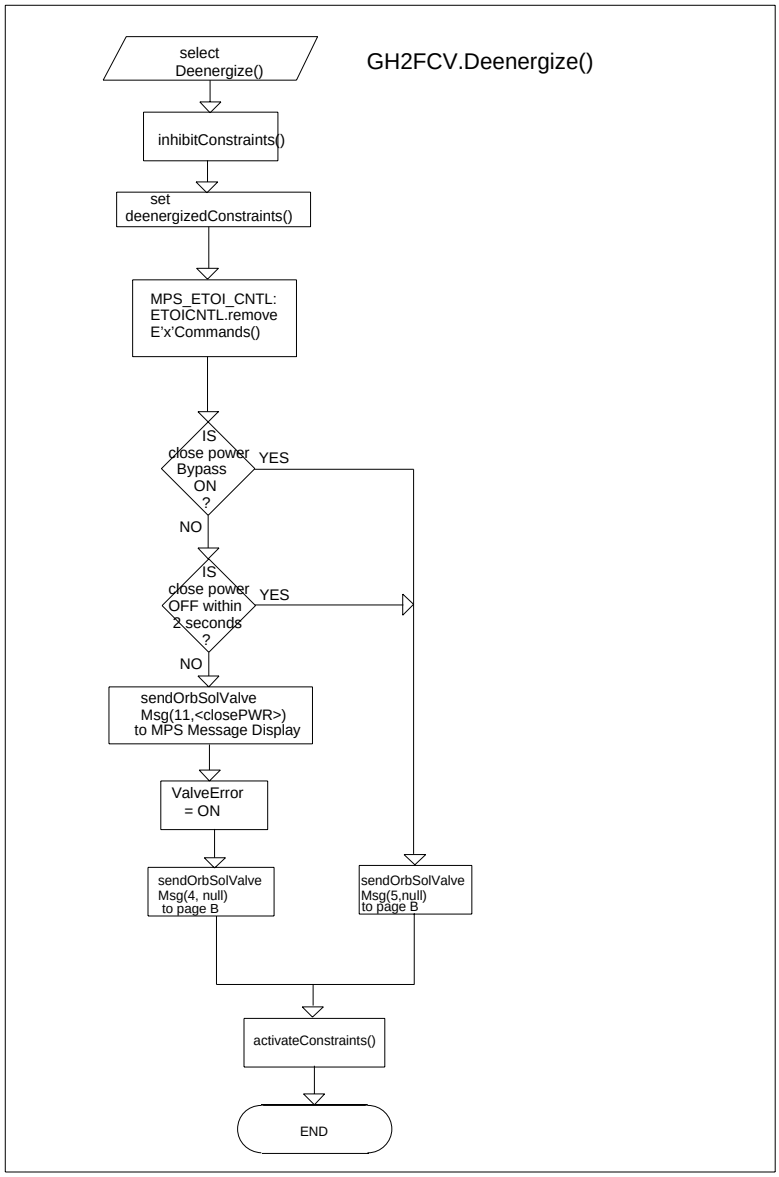
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.17.3.1. energize()



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.17.3.2. deenergize()



3.1.17.3.3. determineCmdState()

3.1.17.3.3.1. Descriptive Text

This method will determine the valve's commanded state based on the valve open power FD. The return value will be an enumerated type with possible values of ENERGIZED, DEENERGIZED, and ERROR.

3.1.17.3.3.2. Effectivity

3.1.17.3.3.3. Sequence

Case:

<closePwr> ON : cmdState = ENERGIZED;

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

<closePwr> OFF : cmdState = DEENERGIZED;

<closePwr>Bypass ON : cmdState = ERROR;

End Case:

3.1.17.3.4. getCmdState()

3.1.17.3.4.1. Descriptive Text

This method will return the Enumerated Command State of the valve.

3.1.17.3.4.2. Sequence

```
getCmdState()
{
        return cmdState;
}
```

3.1.17.3.5. determineState()

3.1.17.3.5.1. Descriptive Text

This method will determine valve state based on power indication. For the two way solenoid valves, the valve state determination is the same functionally to command state: 'energized' is equal to 'closed' and 'deenergized' is equal to 'open'. The return value will be an enumerated type with possible values of OPEN, CLOSED, and ERROR.

3.1.17.3.5.2. Sequence

Case:

<closePwr> ON : State = CLOSED;

<closePwr> OFF : State = OPEN;

<closePwr>.Bypass ON : State = ERROR;

End Case:

3.1.17.3.6. getState()

3.1.17.3.6.1. Descriptive Text

This method will return the Enumerated State of the valve.

3.1.17.3.6.2. Sequence

```
getState()
{
        return State;
}
```

3.1.17.3.7. verifyCommandsOff()

3.1.17.3.7.1. Descriptive Text

This method will status the command states, via byteTest4(), of the FCV sim commands (hiFlowCmdA and hiFlowCmdB) and return TRUE for and energized command status of OFF and FALSE for the opposite.

3.1.17.3.7.2. Sequence

```
verifyCommandsOff()
{
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
        energized command status1 = byteTest4 <hiFlowCmdA>
        energized command status2 = byteTest4 <hiFlowCmdB>

        {
        verify  energized command status 1 == OFF
        else send message to page A: valveID "FCV high flow command" <hiFlowCmdA>" is ON, should be
OFF"
        }
        {
        verify energized command status 2 == OFF
        else send message to page A: valveID "FCV high flow command" <hiFlowCmdB>" is ON, should be OFF"
        }
{
        verify  energized command status 1 && 2 == OFF
        Return TRUE

        else Return FALSE

}
```

Both of these messages are error messages and should be displayed in Red.

3.1.17.3.8. setDeenergizedConstraints()

3.1.17.3.8.1. Descriptive Text

This method changes the constraint criteria of the measurements associated with this valve to those expected with the valve in the closed position.

3.1.17.3.8.2. Sequence

```
setDeenergizedConstraints()
        {
                CHANGE <go2ClosePwr> CONSTRAINT CRITERIA TO OFF;
        }
        {
                CHANGE <gh2ClosePwr> CONSTRAINT CRITERIA TO OFF;
        }
```

3.1.17.3.9. setEnergizedConstraints()

3.1.17.3.9.1. Descriptive Text

This method changes the constraint criteria of the measurements associated with this valve to those expected with the valve in the open position.

3.1.17.3.9.2. Sequence

```
setEnergizedConstraints()
        {
        If <go2ClosePwr>Bypass == OFF

                CHANGE <go2ClosePwr> CONSTRAINT CRITERIA TO ON;

        END IF
        }
        {
        If <gh2ClosePwr>Bypass == OFF
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

CHANGE <gh2ClosePwr> CONSTRAINT CRITERIA TO ON;

END IF
}

3.1.17.3.10. activateConstraints()

3.1.17.3.10.1. Descriptive Text

This operation activates constraint monitoring on valve power indicators.

3.1.17.3.10.2. Sequence

activateConstraints()
{
Activate Constraint Monitoring on
<go2ClosePwr>
<gh2ClosePwr>
}

3.1.17.3.11. inhibitConstraints()

3.1.17.3.11.1. Descriptive Text

This operation de-activates constraint monitoring on valve power indicators.

3.1.17.3.11.2. Sequence

inhibitConstraints()
{
Inhibit Constraint Monitoring on
<go2ClosePwr>
<gh2ClosePwr>
}

3.1.17.3.12. monitorOnTime()

3.1.17.3.13. termMonitorOnTime()

3.1.17.3.14. startMonitorOnTime()

3.1.17.3.15. turnOnHiFlowCmdA()

3.1.17.3.15.1. Descriptive Text

This method sets the hiFlowCmdA to on to support individual command setting for test and checkout via an association with MPS_ET_OI_CONTROL

3.1.17.3.15.2. turnOnHiFlowCmdA()

{
set hiFlowCmdA to on;
}

//Note to Self: Self, don't forget to communicate with DelmaBox (ERROR==FALSE)//

3.1.17.3.16. turnOffHiFlowCmdA()

3.1.17.3.16.1. Descriptive Text

This method sets the hiFlowCmdA to off to support individual command setting for test and checkout via an association with MPS_ET_OI_CONTROL

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.1.17.3.16.2. turnOffHiFlowCmdA()

{
set hiFlowCmdA to off;
}

3.1.17.3.17. turnOnHiFlowCmdB()

3.1.17.3.17.1. Descriptive Text

This method sets the hiFlowCmdB to on to support individual command setting for test and checkout via an association with MPS_ET_OI_CONTROL

3.1.17.3.17.2. turnOnHiFlowCmdB()

{
set hiFlowCmdB to on;
}

3.1.17.3.18. turnOffHiFlowCmdB()

3.1.17.3.18.1. Descriptive Text

This method sets the hiFlowCmdA to off to support individual command setting for test and checkout via an association with MPS_ET_OI_CONTROL

3.1.17.3.18.2. turnOffHiFlowCmdB()

{
set hiFlowCmdB to off;
}

3.1.17.3.19. #turnOnFcvInhibit()//VAB/PAD Method//

3.1.17.3.20. #turnOffFcvInhibit()//VAB/PAD Method//

3.1.17.3.21. #readFcvInhibit()//VAB/PAD Method//

3.1.17.4. User Interfaces

3.1.17.5. Safety Requirements

3.1.18. MPS_SolValve_ORB_SMERegIsoA

3.1.18.1. Description

The MPS SSME pneumatic helium regulator leg A isolation valves (LV1,3,5) are solenoid valves with 1 open command, 1 power indication, and 2 switch scans (1 open, 1 close). Functionally, the valves serve as isolation valves between the onboard high pressure helium supply tanks and the 750 psi regulators and are opened/closed during the activation and securing of SSME 750 psi pneumatics. The valve class inherits from the MPS_SolValve_ORB class. The valve class has an association with the MPS_SME_PneuControl class during pneumatics activation for activating/securing RCL and for regulated outlet pressure monitoring.

3.1.18.1.1. Instance Index

Instance	Display User	Other Users
MPS_LV1	MPS	
MPS_LV3	MPS	

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

MPS_LV5	MPS	

3.1.18.1.2. Class Diagram

<insert class diagram here>

3.1.18.1.3. Attributes and Aggregations

Attribute	Data Type	Initial Value	Publish / Persistence	Client Access	Description
openCmd	DS_ON_OFF			R,W	
openPwr	DM_ON_OFF			R	

3.1.18.1.4. Operations Index

Operation	Description
open()	This operation will perform the necessary functions to open the E1,2, or 3 reg isolation solenoid valves LV1,3, and 5.
close()	This operation will perform the necessary functions to close the E1,2, or 3 reg isolation solenoid valves LV1,3,and 5.
determineCmdState()	This method looks at the open power indicator and returns an enumerated state based on the logic table. Possible states are: Energized Deenergized Error
getCmdState()	This method returns the enumerated command state.
determineState()	This method looks at the open power indicator and returns an enumerated state based on the logic table. Possible states are: Open Closed Error
getState()	This method returns the enumerated valve state.
setOpenConstraints()	This method will establish constraint criteria for an energized (open) reg isolation valve.
setCloseConstraints()	This method will establish constraint criteria for a deenergized (closed) reg isolation valve.
activateConstraints()	This method will activate constraint monitoring for reg isolation valve.
inhibitConstraints()	This method will inhibit constraint monitoring for reg isolation valve.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

verifyCmdsOff()	This method will return a True/False based on results of ByteTest4 for command issuance to solenoid.
verifyTminus6BT4()	This method verifies command is ON at T-6 hours Byte Test 4. This is a VAB/PAD method.
verifyTminus3BT4()	This method verifies command is ON at T-3 hours Byte Test 4. This is a VAB/PAD method.

3.1.18.1.5. Associations

Association	Type	Class	Description
MPS_SME_PneuControl			Calling class for reg activations.

3.1.18.2. State Description

3.1.18.2.1. State Diagram

<insert state diagram here>

3.1.18.2.2. State Table

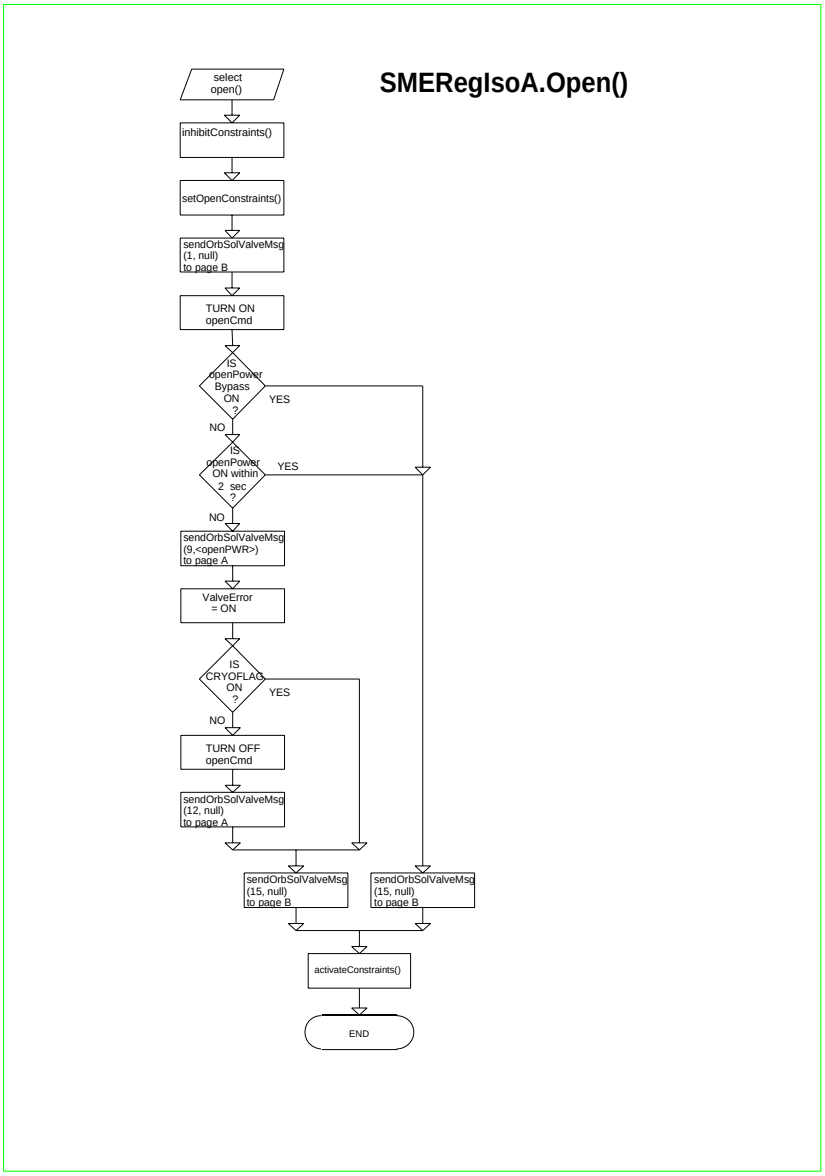
3.1.18.3. Operations

3.1.18.3.1. open()

3.1.18.3.1.1. Descriptive Text

This method will perform the necessary functions for opening (activating) MPS SSME 750 reg A isolation solenoid valves. The method calls associated class MPS_RCL_E1(or E2 or E3)_Reg_Press for activation and inhibiting of reactive control logic during opening of isolation valve MPS_LV1,3,or 5. This method is called by the MPS_SME_PneuControl class 'activateAReg' method or 'activateA&BReg' method. The method also checks for regulator outlet pressure to be within OMRS limits following reg activation and notifies operator if not within OMRS band by its association with the panel pressure transducer.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

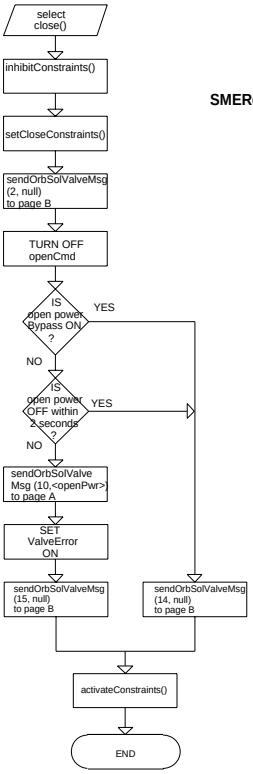


3.1.18.3.2. close()

3.1.18.3.2.1. Descriptive Text

This method will perform closure of the SME reg A solenoid valve. Constraints will be maintained on valve power indications.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



SMERegIsoA.Close()

3.1.18.3.3. determineCmdState()

3.1.18.3.3.1. Descriptive Text

This method will determine the valve's commanded state based on the valve open power FD. The return value will be an enumerated type with possible values of ENERGIZED, DEENERGIZED and ERROR.

3.1.18.3.3.2. Effectivity

Operates on change data.

3.1.18.3.3.3. Sequence

Case:

<openPwr> ON : cmdState = ENERGIZED;

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

<openPwr> OFF : cmdState = DEENERGIZED;

<openPwr>Bypass ON : cmdState = ERROR;

End Case:

3.1.18.3.4. getCmdState()

3.1.18.3.4.1. Descriptive Text

This method will return the Enumerated Command State of the valve.

3.1.18.3.4.2. Sequence

```
getCmdState()
{
        return cmdState;
}
```

3.1.18.3.5. determineState()

3.1.18.3.5.1. Descriptive Text

This method will determine valve state based on power indication. For the two way solenoid valves, the valve state determination is the same functionally to command state: 'energized' is equal to 'open' and 'deenergized' is equal to 'closed'.

3.1.18.3.5.2. Sequence

Case:

<openPwr> ON : State = OPEN;

<openPwr> OFF : State = CLOSED;

<openPwr>Bypass ON : State = ERROR;
End Case:

3.1.18.3.6. getState()

3.1.18.3.6.1. Descriptive Text

This method will return the Enumerated State of the valve.

3.1.18.3.6.2. Sequence

```
getState()
{
        return State;
}
```

3.1.18.3.7. verifyCommandsOff()

3.1.18.3.7.1. Descriptive Text

This function will verify that both the open and close commands are OFF

3.1.18.3.7.2. Sequence

```
verifyCommandsOff()
{
        open command status = byteTest4 <Open Command>
        {
        verify  open command status == OFF
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
        return TRUE
        }
        ELSE return FALSE
}
```

3.1.18.3.8. setCloseConstraints()

3.1.18.3.8.1. Descriptive Text

This method changes the constraint criteria of the measurements associated with this valve to those expected with the valve in the closed position.

3.1.18.3.8.2. Sequence

```
setCloseConstraints()
        {
                CHANGE <openPwr> CONSTRAINT CRITERIA TO OFF;
        }
```

3.1.18.3.9. setOpenConstraints()

3.1.18.3.9.1. Descriptive Text

This method changes the constraint criteria of the measurements associated with this valve to those expected with the valve in the open position.

3.1.18.3.9.2. Sequence

```
setOpenConstraints()
        {
                CHANGE <openPwr> CONSTRAINT CRITERIA TO ON;

        }
```

3.1.18.3.10. activateConstraints()

3.1.18.3.10.1. Descriptive Text

This operation activates constraint monitoring on valve position and power indicators.

3.1.18.3.10.2. Sequence

```
activateConstraints()
{
        If <openPwr>.Byp = OFF
                Activate Constraint Monitoring on <openPwr>
        End if
}
```

3.1.18.3.11. inhibitConstraints()

3.1.18.3.11.1. Descriptive Text

This operation de-activates constraint monitoring on valve position and power indicators.

3.1.18.3.11.2. Sequence

```
inhibitConstraints()
{
        Inhibit Constraint Monitoring on
                <openPwr>
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.18.3.12. verifyTminus3BT4()//VAB/PAD Method//

3.1.18.3.13. verifyTminus6BT4()//VAB/PAD Method//

3.1.18.4. User Interfaces

3.1.18.5. Safety Requirements

3.1.19. MPS_SolValve_ORB_SMERegIsoB

3.1.19.1. Description

The MPS SSME pneumatic helium regulator leg B isolation valves (LV2,4,6) are solenoid valves with 2 open commands, 2 RPCs, 1 power indication, and 2 switch scans (1 open, 1 close). Functionally, the valves serve as isolation valves between the onboard high pressure helium supply tanks and the 750 psi regulators and are opened/closed during the activation and securing of SSME 750 psi pneumatics. The valve class inherits from the MPS_SolValve_ORB class. The valve class has an association with the MPS_SME_PneuControl class which activates/secures RCL and monitors regulated outlet pressure during pneumatics activation..

3.1.19.1.1. Instance Index

Instance	Display User	Other Users
MPS_LV2	MPS	
MPS_LV4	MPS	
MPS_LV6	MPS	

3.1.19.1.2. Class Diagram

<insert class diagram here>

3.1.19.1.3. Attributes and Aggregations

Attribute	Data Type	Initial Value	Publish / Persistence	Client Access	Description
openCmdA	DS_ON_OFF			R,W	
openCmdB	DS_ON_OFF			R,W	
openRpc1	DM_ON_OFF			R	
openRpc2	DM_ON_OFF			R	
openPwr	DM_ON_OFF			R	

3.1.19.1.4. Operations Index

Operation	Description
open()	This operation will perform the necessary functions to open the E1,2, or

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

	3 reg isolation solenoid valves LV2,4, and 6.
close()	This operation will perform the necessary functions to close the E1,2, or 3 reg isolation solenoid valves LV2,4,and 6.
determineCmdState()	This method looks at the open power indicator and returns an enumerated state based on the logic table. Possible states are: Energized Deenergized Error
getCmdState()	This method returns the enumerated command state.
determineState()	This method looks at the open power indicator and returns an enumerated state based on the logic table. Possible states are: Open Closed Error
getState()	This method returns the enumerated valve state.
verifyCmdsOff()	This method will return a True/False based on results of ByteTest4 for command issuance to solenoid.
setOpenConstraints()	This method will establish constraint criteria for an energized (open) reg isolation valve.
setCloseConstraints()	This method will establish constraint criteria for a deenergized (closed) reg isolation valve.
activateConstraints()	This method will activate constraint monitoring for reg isolation valve.
inhibitConstraints()	This method will inhibit constraint monitoring for reg isolation valve.
verifyTminus3BT4()//	This method will run a ByteTest4 to determine if command is on at outlet of MDM for T-3 hour configuration. VAB/PAD method.
verifyTminus6BT4()//	This method will run a ByteTest4 to determine if command is on at outlet of MDM for T-6 hour configuration. VAB/PAD method.

3.1.19.1.5. Associations

Association	Type	Class	Description
MPS_SME_PneuControl			Calling class for reg activations.

3.1.19.2. State Description

3.1.19.2.1. State Diagram

<insert state diagram here>

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

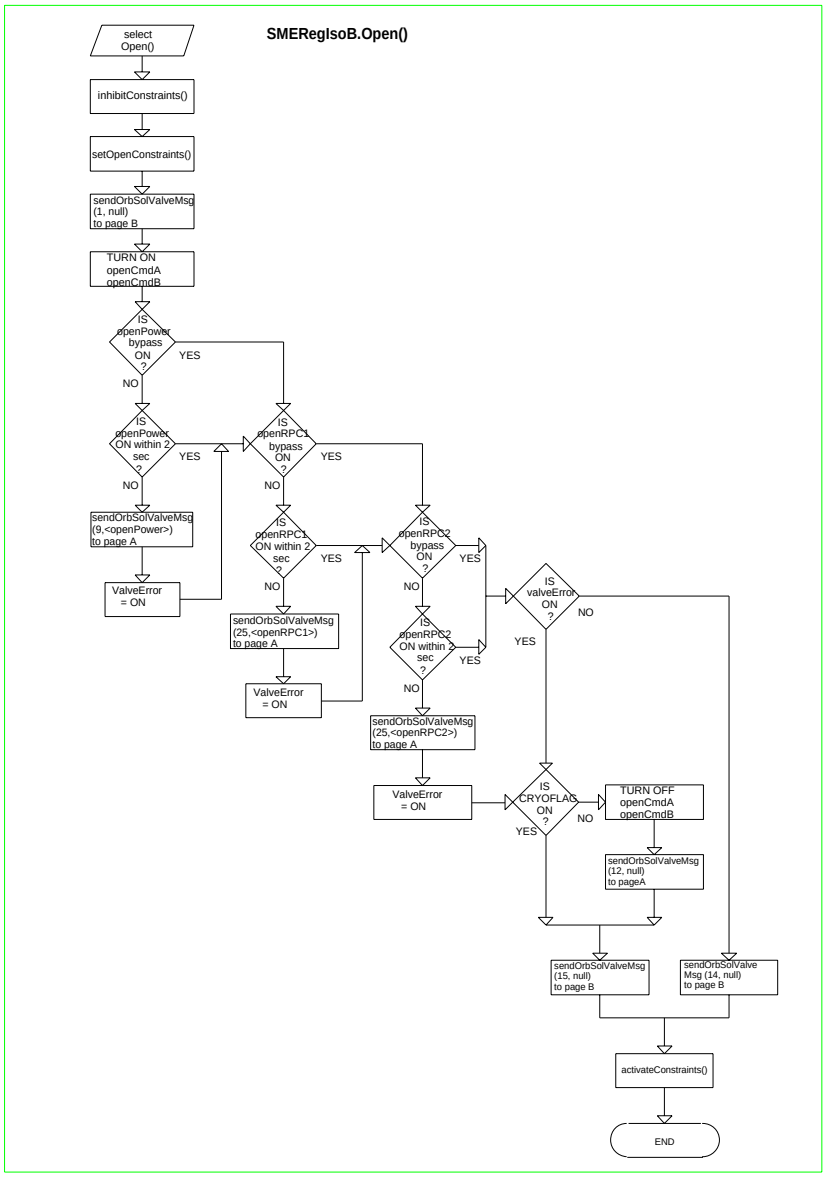
3.1.19.2.2. State Table

3.1.19.3. Operations

3.1.19.3.1. open()

3.1.19.3.1.1. Descriptive Text

This method will perform the necessary functions for opening (activating) MPS SSME 750 reg B isolation solenoid valves MPS_LV2,4,or 6. This method is called by the MPS_SME_PneuControl class 'activateBReg' method or 'activateA&BReg' method. RPCs are checked in addition to the power indicator during solenoid cycling. The MPS_SME_PneuControl class method maintains RCL function during reg activation and also checks for regulator outlet pressure to be within OMRS limits following reg activation. and notifies operator if not within OMRS band by its association with the panel pressure transducer.

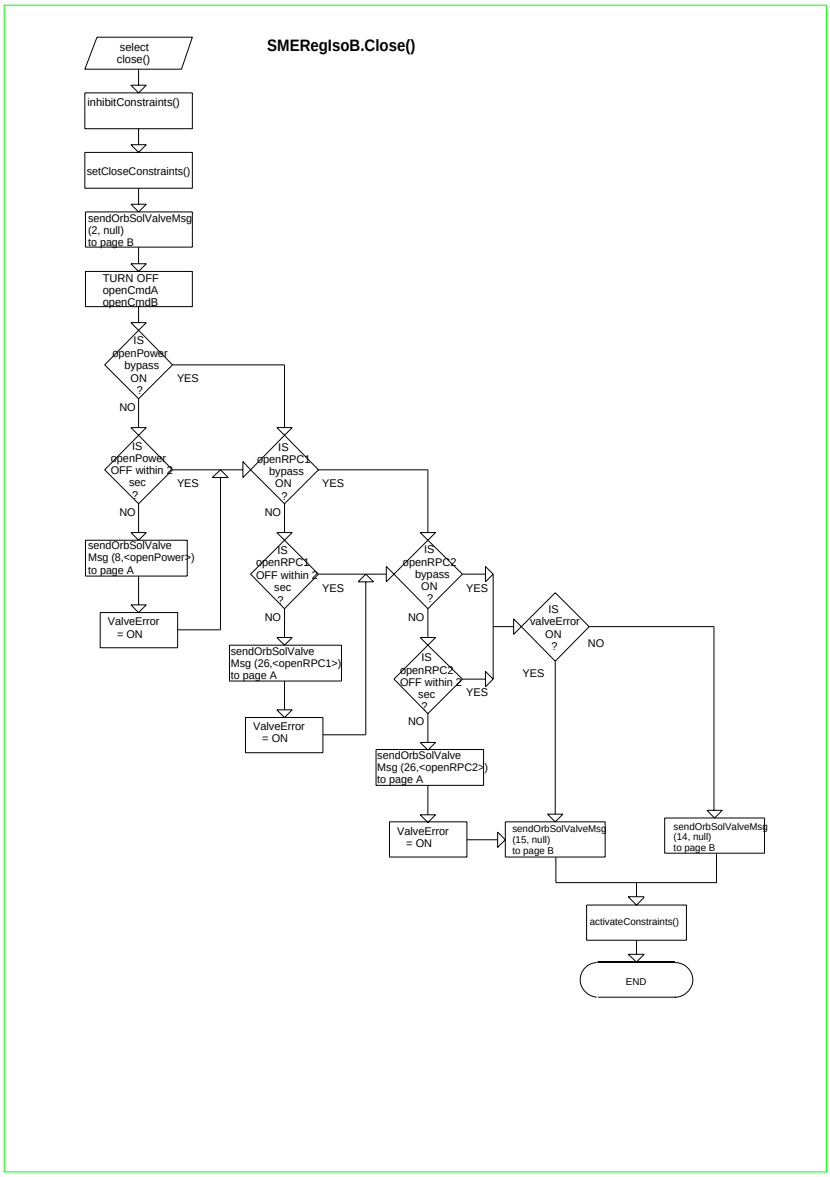


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.19.3.2. close()

3.1.19.3.2.1. Descriptive Text

This method will perform closure of the SME reg isolation A solenoid valve. Constraints will be maintained on valve power indications.



3.1.19.3.3. determineCmdState()

3.1.19.3.3.1. Descriptive Text

This method will determine the valve's commanded state based on the valve open power and open RPC A and RPC B indicators. The return value will be an enumerated type with possible values of ENERGIZED, DEENERGIZED, and ERROR.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.19.3.3.2. Effectivity

Operates on change data.

3.1.19.3.3.3. Sequence

Case:

<openPwr> OFF
<openPwr>.BYP OFF
<openRPC_1> OFF
<openRPC_1>.BYP OFF
<openRPC_2> OFF
<openRPC_2>.BYP OFF
<cmdState> = DEENERGIZED;

<openPwr> OFF
<openPwr>.BYP OFF
<openRPC_1> OFF
<openRPC_1>.BYP OFF
<openRPC_2>.BYP ON
<cmdState> = DEENERGIZED

<openPwr> OFF
<openPwr>.BYP OFF
<openRPC_1>.BYP ON
<openRPC_2> OFF
<openRPC_2>.BYP OFF
<cmdState> = DEENERGIZED;

<openPwr> OFF
<openPwr>.BYP OFF
<openRPC_1>.BYP ON
<openRPC_2>.BYP ON
<cmdState> = DEENERGIZED;

<openPwr>.BYP ON
<openRPC_1> OFF
<openRPC_1>.BYP OFF
<openRPC_2> OFF
<openRPC_2>.BYP OFF
<cmdState> = DEENERGIZED;

<openPwr>.BYP ON
<openRPC_1> OFF
<openRPC_1>.BYP OFF
<openRPC_2>.BYP ON
<cmdState> = DEENERGIZED;

<openPwr>.BYP ON
<openRPC_1>.BYP ON
<openRPC_2> OFF
<openRPC_2>.BYP OFF
<cmdState> = DEENERGIZED;

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
        <openPwr>.BYP ON
        <openRPC_1>.BYP     ON
        <openRPC_2>         ON
        <openRPC_2>.BYP     OFF
                <cmdState> = ENERGIZED;

        <openPwr>.BYP ON
        <openRPC_1>         ON
        <openRPC_1>.BYP     OFF
        <openRPC_2>.BYP     ON
                <cmdState> = ENERGIZED;

        <openPwr>.BYP ON
        <openRPC_1>         ON
        <openRPC_1>.BYP     OFF
        <openRPC_2>         ON
        <openRPC_2>.BYP     OFF
                <cmdState> = ENERGIZED;

        <openPwr>           ON
        <openPwr>.BYP OFF
        <openRPC_1>.BYP     ON
        <openRPC_2>.BYP     ON
                <cmdState> = ENERGIZED;

        <openPwr>           ON
        <openPwr>.BYP OFF
        <openRPC_1>.BYP     ON
        <openRPC_2>         ON
        <openRPC_2>.BYP     OFF
                <cmdState> = ENERGIZED;

        <openPwr>           ON
        <openPwr>.BYP OFF
        <openRPC_1>         ON
        <openRPC_1>.BYP     OFF
        <openRPC_2>.BYP     ON
                <cmdState> = ENERGIZED;

        <openPwr>           ON
        <openPwr>.BYP OFF
        <openRPC_1>         ON
        <openRPC_1>.BYP     OFF
        <openRPC_2>         ON
        <openRPC_2>.BYP     OFF
                <cmdState> = ENERGIZED;

    DEFAULT CASE
                = ERROR;

End Case:
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.1.19.3.4. getCmdState()

3.1.19.3.4.1. Descriptive Text

This method will return the Enumerated Command State of the valve.

3.1.19.3.4.2. Sequence

```
getCmdState()
{
        return cmdState;
}
```

3.1.19.3.5. determineState()

3.1.19.3.5.1. Descriptive Text

This method will determine valve state based on power indication. For the two way solenoid valves, the valve state determination is the same functionally to command state: 'energized' is equal to 'open' and 'deenergized' is equal to 'closed'.

3.1.19.3.5.2. Sequence

Case:

```
  <openPwr>              ON :     State = OPEN;

  <openPwr>              OFF :    State = CLOSED;

 <openPwr>Bypass      ON :     State = ERROR;
```

End Case:

3.1.19.3.6. getState()

3.1.19.3.6.1. Descriptive Text

This method will return the Enumerated State of the valve.

3.1.19.3.6.2. Sequence

```
getState()
{
        return State;
}
```

3.1.19.3.7. verifyCommandsOff()

3.1.19.3.7.1. Descriptive Text

This function will verify that both the open commands are OFF

3.1.19.3.7.2. Sequence

```
verifyCommandsOff()
{
        open command status1 = byteTest4 <Open Command A>
        open command status2 = byteTest4 <Open Command B>
        {
        verify  open command status1 && open command status2  == OFF

        return TRUE
        }
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
        ELSE return FALSE
}
```

3.1.19.3.8. setValveCloseConstraints()

3.1.19.3.8.1. Descriptive Text

This method changes the constraint criteria of the measurements associated with this valve to those expected with the valve in the closed position.

3.1.19.3.8.2. Sequence

```
setCloseConstraints()
        {
                CHANGE <openPwr> CONSTRAINT CRITERIA TO OFF;
                CHANGE <openRPC1> CONSTRAINT CRITERIA TO OFF;
                CHANGE <openRPC2> CONSTRAINT CRITERIA TO OFF;
        }
```

3.1.19.3.9. setValveOpenConstraints()

3.1.19.3.9.1. Descriptive Text

This method changes the constraint criteria of the measurements associated with this valve to those expected with the valve in the open position.

3.1.19.3.9.2. Sequence

```
setOpenConstraints()
        {

                CHANGE <openPwr> CONSTRAINT CRITERIA TO ON;
                CHANGE  <openRPC1> CONSTRAINT CRITERIA TO ON;
                CHANGE  <openRPC2> CONSTRAINT CRITERIA TO ON;

        }
```

3.1.19.3.10. activateConstraints()

3.1.19.3.10.1. Descriptive Text

This operation activates constraint monitoring on valve position and power indicators.

3.1.19.3.10.2. Sequence

```
activateConstraints()
{
        IF <openPwr>.Byp == OFF;
        Activate Constraint Monitoring on
                <openPwr>
        END IF;

        IF <openRPC1>.Byp == OFF;
        Activate Constraint Monitoring on
                <openRPC1>
        END IF;

        IF <openRPC2>.Byp == OFF;
        Activate Constraint Monitoring on
                <openRPC2>
        END IF;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

}

3.1.19.3.11. inhibitConstraints()

3.1.19.3.11.1. Descriptive Text

This operation de-activates constraint monitoring on valve position and power indicators.

3.1.19.3.11.2. Sequence

inhibitConstraints()
{
Inhibit Constraint Monitoring on
<openPwr>
<openRPC1>
<openRPC2>
}
}

3.1.19.3.12. verifyTminus3BT4()//VAB/PAD method//

3.1.19.3.13. verifyTminus6BT4()//VAB/PAD method//

3.1.19.4. User Interfaces

3.1.19.5. Safety Requirements

3.1.20. MPS_SolValve_ORB_MPSRegIso

3.1.20.1. Description

The MPS pneumatic helium regulator isolation valves (LV7,8) are solenoid valves with 1 open command and 1 power indication. The MPS reg isolation valves are two parallel valves upstream of the MPS 750 psi pneumatic regulator on MPS panel 4. Opening either valve allows MPS helium tank TK4 pressure (nominally 2000 psi for check-out, 4500 psi for launch) to flow to the 750 regulator, and results in pneumatic system activation. The valve class inherits from the MPS_SolValve_ORB class. The valve class has an association with the MPS_PneuControl class, which calls instances of this class open() and close() during pneumatics activation. As a safety control for non-hazardous 750 activation with personnel inside the aft fuselage, Prerequisite Control Logic requires all MPS propellant valve solenoids de-energized and the MPS TK4 tank isolated from supply source (interconnects closed and GSE helium bottle fill secured) prior to opening LV7 or LV8 unless regulated outlet pressure is already above 650 psia or the PCL bypass is issued.

3.1.20.1.1. Instance Index

Instance	Display User	Other Users
MPS_LV7	MPS	
MPS_LV8	MPS	

3.1.20.1.2. Class Diagram

<insert class diagram here>

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.20.1.3. Attributes and Aggregations

Attribute	Data Type	Initial Value	Publish / Persistence	Client Access	Description
openCmd	DS_ON_OFF			R,W	
openPwr	DM_ON_OFF			R	

3.1.20.1.4. Operations Index

Operation	Description
open()	This method performs the necessary functions to open MPS 750 reg isolation valves LV7 and LV8.
close()	This method performs the necessary functions to close MPS 750 reg isolation valves LV7 and LV8.
determineCmdState()	This method determines the reg isolation solenoid valve command state based on power indication and returns an enumerated state based on the logic table. Possible states are: Energized Deenergized Error
getCmdState()	This method returns the solenoid valve enumerated command state.
determineState()	This method determines the reg isolation solenoid valve command state based on power indication and returns an enumerated state based on the logic table. Possible states are: Open Closed Error
getState()	This method returns the enumerated valve state.
verifyCmdsOff()	This method verifies byteTest4() results show valve commands presently OFF.
activateConstraints()	This method activates constraint monitoring on associated FDs.
setOpenConstraints()	This method configures constraint monitoring for an energized/open reg iso.
setCloseConstraints()	This method configures constraint monitoring for an deenergized/closed reg iso.
inhibitConstraints()	This method terminates constraint monitoring on associated FDs.
verifyTminus6BT4()// /	This method verifies byteTest4() results show valve command in T-6 HR configuration.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

verifyTminus3BT4()// /	This method verifies byteTest4() results show valve command in T-3 HR configuration.

3.1.20.1.5. Associations

Association	Type	Class	Description
MPS_Pneu		MPS_ORB_PneumaticRegCntl	

3.1.20.2. State Description

3.1.20.2.1. State Diagram

<insert state diagram here>

3.1.20.2.2. State Table

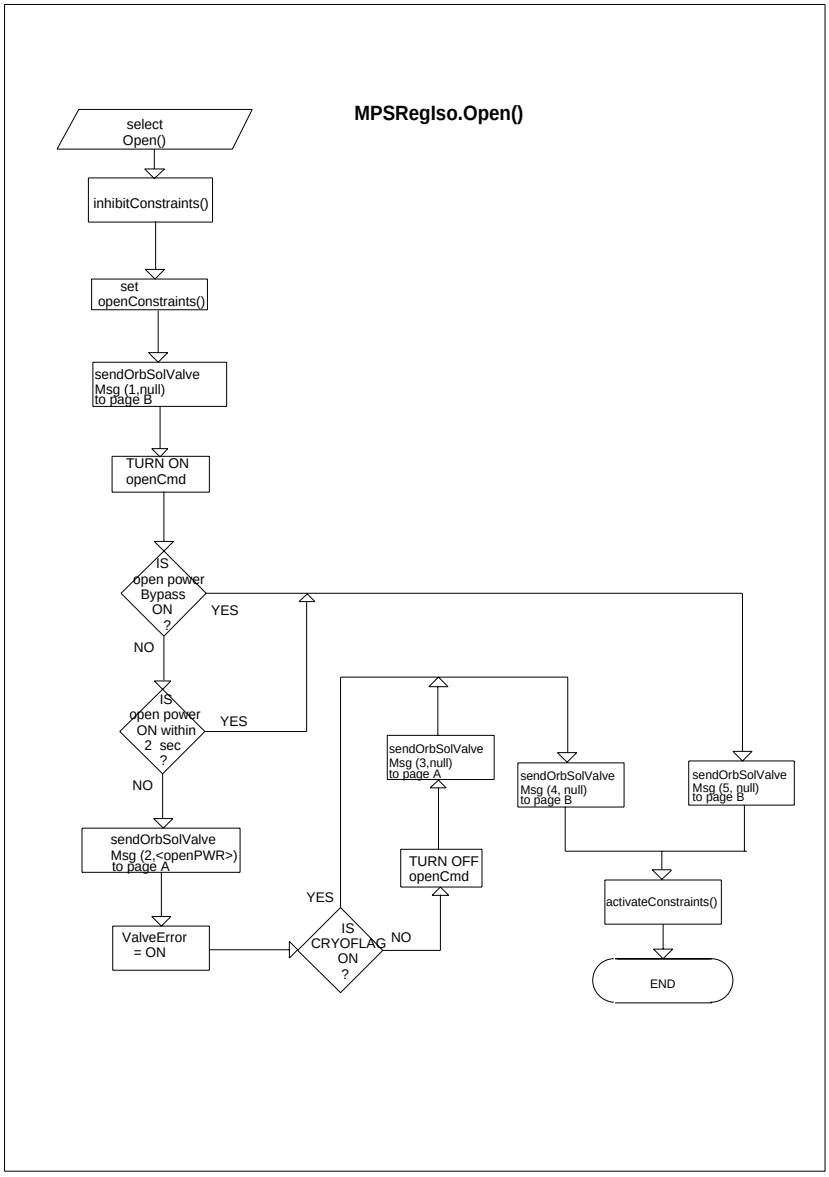
3.1.20.3. Operations

3.1.20.3.1. open()

3.1.20.3.1.1. Descriptive Text

This method will perform the necessary functions for opening (energizing) MPS 750 reg isolation solenoid valves MPS_LV7 and LV8. This method is called by the MPS_ORB_PneumaticRegCntl class activateMPSPneumatics() and secureMPSPneumatics() methods.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

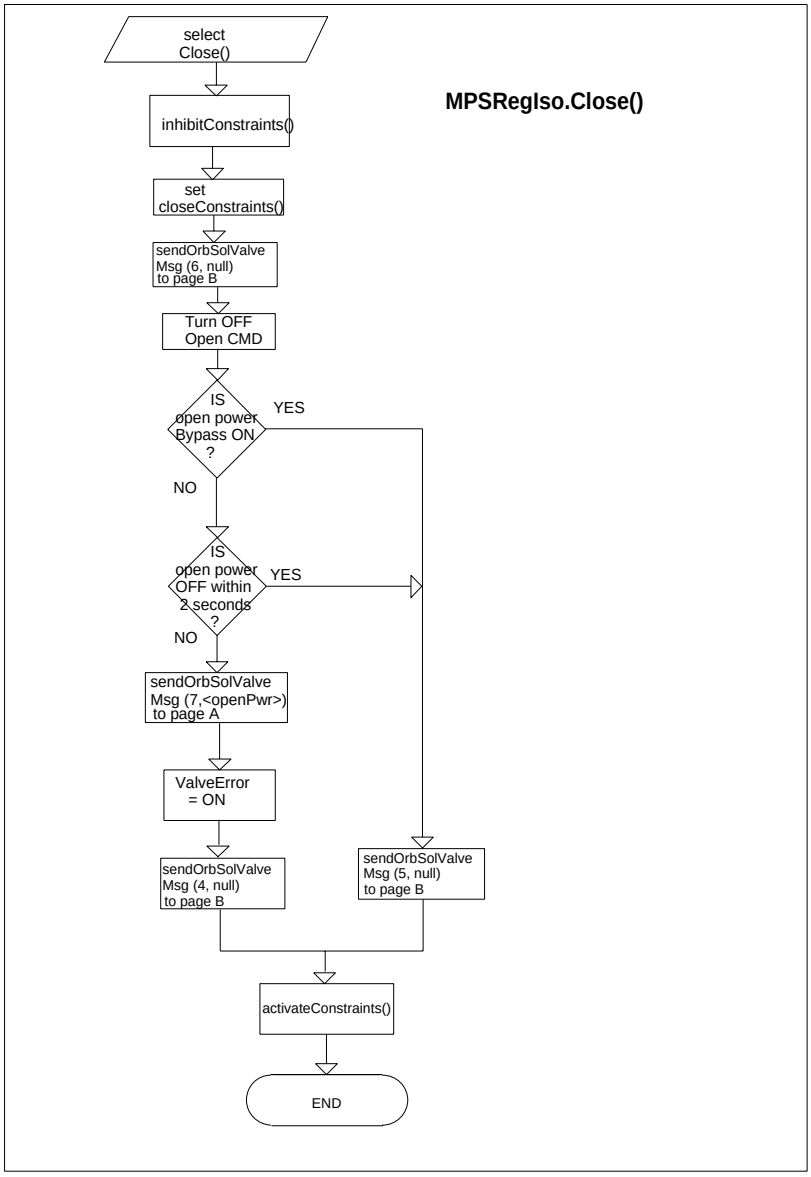


3.1.20.3.2. close()

3.1.20.3.2.1. Descriptive Text

This method will perform closure of the MPS reg isolation solenoid valves LV7 or LV8. Constraints will be maintained on valve power indications. This method is called by the MPS_SME_PneuControl class 'secureMPSPneumatics' method.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



3.1.20.3.3. determineCmdState()

3.1.20.3.3.1. Descriptive Text

This method will determine the valve's commanded state based on the valve open power FD. The return value will be an enumerated type MPS_ORB_SOL_STATE with possible values of ENERGIZED, DEENERGIZED, and ERROR.

3.1.20.3.3.2. Effectivity

3.1.20.3.3.3. Sequence

Case:

<openPwr> ON : cmdState = ENERGIZED;

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

<openPwr> OFF : cmdState = DEENERGIZED;

<openPwr>Bypass ON : cmdState = ERROR;

End Case:

3.1.20.3.4. getCmdState()

3.1.20.3.4.1. Descriptive Text

This method will return the Enumerated Command State of the valve.

3.1.20.3.4.2. Sequence

```
getCmdState()
{
        return cmdState;
}
```

3.1.20.3.5. determineState()

3.1.20.3.5.1. Descriptive Text

This method will determine valve state based on power indication. For the two way solenoid valves, the valve state determination is the same functionally to command state: 'energized' is equal to 'open' and 'deenergized' is equal to 'closed'. The State will be an MPS_VALVE_IND_STATE type enumerated FD with possible values of OPEN, CLOSED, and ERROR.

3.1.20.3.5.2. Sequence

Case:

<openPwr> ON : State = OPEN;

<openPwr> OFF : State = CLOSED;

<openPwr.Bypass> ON : State = ERROR;

End Case:

3.1.20.3.6. getState()

3.1.20.3.6.1. Descriptive Text

This method will return the Enumerated State of the valve.

3.1.20.3.6.2. Sequence

```
getState()
{
        return State;
}
```

3.1.20.3.7. verifyCommandsOff()

3.1.20.3.7.1. Descriptive Text

This function will verify that both the open and close commands are OFF

3.1.20.3.7.2. Sequence

```
verifyCommandsOff()
{
        open command status = byteTest4 <Open Command>
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
        {
        verify  open command status == OFF

        return TRUE
        }
        ELSE
                Send message to MPS message display:
                        ValveID <open Command> "open command is ON, should be OFF"
        and return FALSE
}
```

3.1.20.3.8. setCloseConstraints()

3.1.20.3.8.1. Descriptive Text

This method changes the constraint criteria of the measurements associated with this valve to those expected with the valve in the closed/deenergized position.

3.1.20.3.8.2. Sequence

```
setCloseConstraints()
        {
                CHANGE <openPwr> CONSTRAINT CRITERIA TO OFF;
        }
```

3.1.20.3.9. setOpenConstraints()

3.1.20.3.9.1. Descriptive Text

This method changes the constraint criteria of the measurements associated with this valve to those expected with the valve in the open/energized position.

3.1.20.3.9.2. Sequence

```
setOpenConstraints()
        {
                CHANGE <openPwr> CONSTRAINT CRITERIA TO ON;
        }
```

3.1.20.3.10. activateConstraints()

3.1.20.3.10.1. Descriptive Text

This operation activates constraint monitoring on valve power indicators.

3.1.20.3.10.2. Sequence

```
activateConstraints()
{
        IF <openPwr>Bypass == OFF;
        Activate Constraint Monitoring on <openPwr>
        END IF;
}
```

3.1.20.3.11. inhibitConstraints()

3.1.20.3.11.1. Descriptive Text

This operation de-activates constraint monitoring on valve power indicators.

3.1.20.3.11.2. Sequence

```
inhibitConstraints()
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
{
        Inhibit Constraint Monitoring on <openPwr>
}
```

3.1.20.3.12. verifyTminus6BT4()//

```
verifyTminus6BT4()
{
        command status = byteTest4 <Open Command>
        {
        verify command status == ON

        return TRUE
        }
        ELSE return FALSE
}
```

3.1.20.3.13. verifyTminus3BT4() //

```
verifyTminus3BT4()
{
        command status = byteTest4 <Open Command>
        {
        verify command status == ON

        return TRUE
        }
        ELSE return FALSE
}
```

3.1.20.4. User Interfaces

3.1.20.5. Safety Requirements

3.1.21. MPS_SolValve_ORB_InterconnectIn

3.1.21.1. Description

The MPS helium supply system interconnect in solenoid valves (LV59, LV61, and LV63) are solenoid valves with 2 open commands, 2 RPCs, 1 power indication. Functionally, the valves serve as isolation valves between the onboard high pressure SSME and MPS helium supply tanks and are opened/closed during the configuration of the high pressure helium system to isolate a system leak or maximize supply volume to a given SSME or MPS pneumatic system. The valve class inherits from the MPS_SolValve_ORB class.

3.1.21.1.1. Instance Index

Instance	Display User	Other Users
MPS_LV59	MPS	
MPS_LV61	MPS	
MPS_LV63	MPS	

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.21.1.2. Class Diagram

<insert class diagram here>

3.1.21.1.3. Attributes and Aggregations

Attribute	Data Type	Initial Value	Publish / Persistence	Client Access	Description
openCmdA	DS_ON_OFF			R,W	
openCmdB	DS_ON_OFF			R,W	
openRpc1	DM_ON_OFF			R	
openRpc2	DM_ON_OFF			R	
openPwr	DM_ON_OFF			R	

3.1.21.1.4. Operations Index

Operation	Description
open()	This method performs the necessary steps to open/energize the MPS interconnect-in solenoid valves using both open commands.
openA()	This method performs the necessary steps to open/energize the MPS interconnect-in solenoid valves using command A only.
openB()	This method performs the necessary steps to open/energize the MPS interconnect-in solenoid valves using open command B only.
close()	This method performs the necessary steps to close/deenergize the MPS interconnect-in solenoid valves.
determineCmdState()	This method determines the interconnect-in solenoid valve command state based on power indication and returns an enumerated state based on the logic table. Possible states are: Energized Deenergized Error
getCmdState()	This method returns the enumerated command state for the interconnect-in solenoid valves.
determineIndState()	This method determines the interconnect solenoid valve state based on valve command state. Possible states are: Open Closed Error
getIndState()	This method returns the enumerated state for the interconnect-in

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

	solenoid valves.
verifyCmdsOff()	This method will return a true/false based on the results of a byte test 4 for command issuance to the solenoid.
setOpenConstraints()	This method will establish constraint criteria for an energized/open interconnect in solenoid.
setOpenAConstraints()	This method will establish constraint monitoring criteria for the interconnect in solenoid valves when opened using A command.
setOpenBConstraints()	This method will establish constraint monitoring criteria for the interconnect in solenoid valves when opened using B command.
setCloseConstraints()	This method will establish constraint criteria for a deenergized/closed interconnect in solenoid.
activateConstraints()	This method will activate constraint monitoring for the interconnect in solenoid valves when opened using both commands.
inhibitConstraints()	This method will inhibit constraint monitoring for the interconnect in solenoid valves.
//verifyTminus6BT4()//	This method is a Phase III method to verify commands are OFF at T-6 HRs.
//verifyTminus3BT4()//	This method is a Phase III method to verify commands are OFF at T-3 HRs.

3.1.21.1.5. Associations

Association	Type	Class	Description

3.1.21.2. State Description

3.1.21.2.1. State Diagram

<insert state diagram here>

3.1.21.2.2. State Table

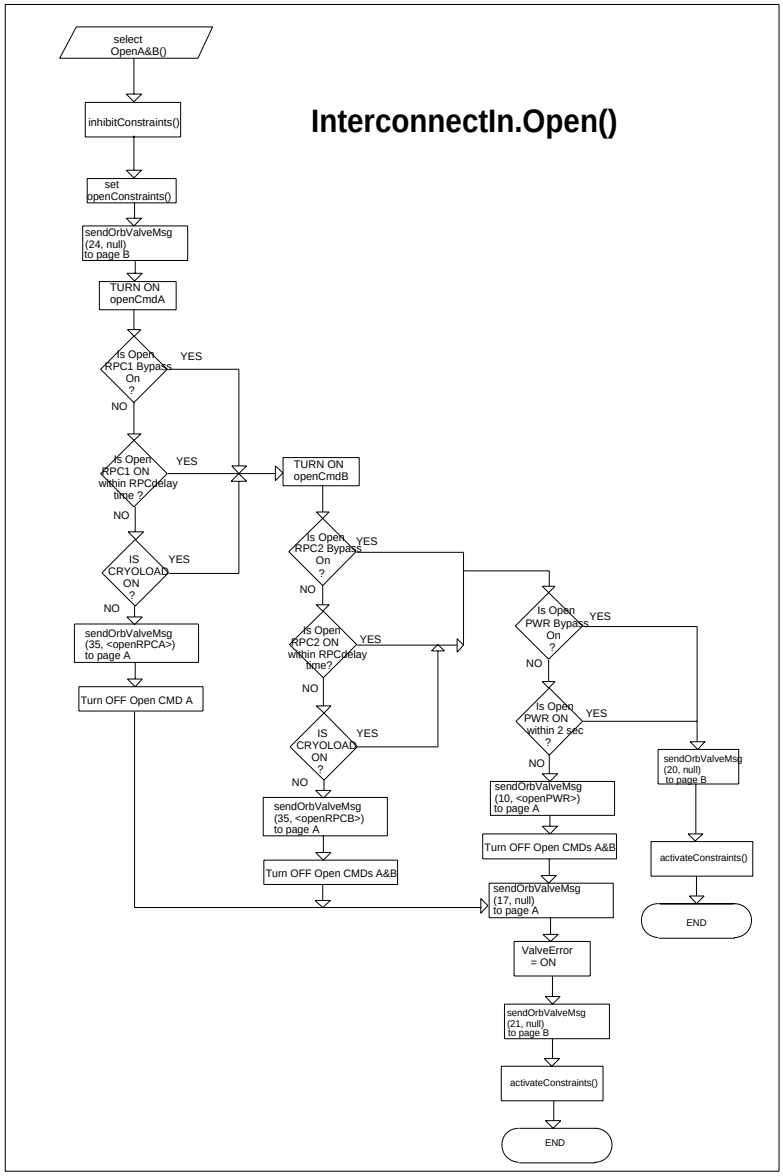
3.1.21.3. Operations

3.1.21.3.1. open() Method

3.1.21.3.1.1. Descriptive

The open method performs steps necessary to energize the interconnect-in 2 way solenoid valves using both open commands. For test and check-out (CRYO flag is OFF), the method verifies all expected RPCs and power indications are received otherwise both open commmands are removed and appropriate error messages are sent to the operator. If in cryo load, the method proceeds on regardless of failures.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

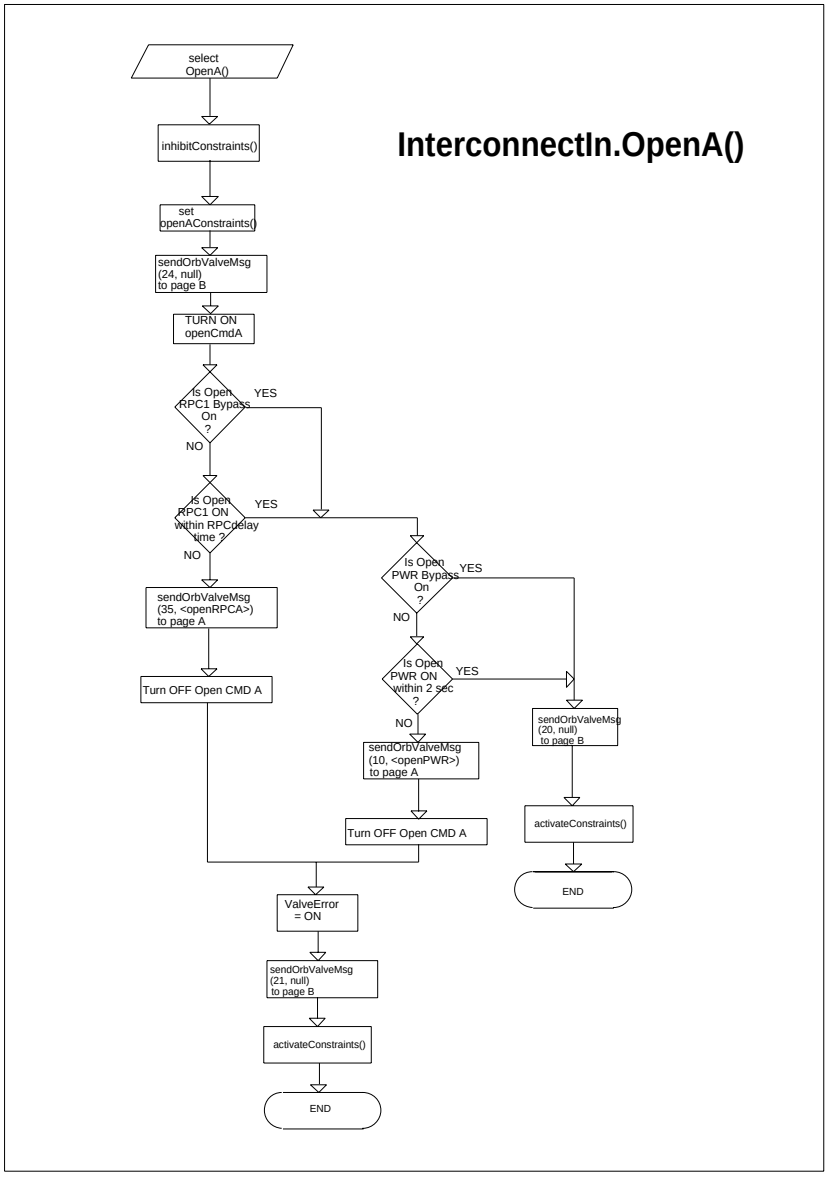


3.1.21.3.2. openA() Method

3.1.21.3.2.1. Description

The open method performs steps necessary to energize the interconnect-in 2 way solenoid valves using open command A only. The method verifies all expected RPCs and power indications are received otherwise open commmand A is removed and appropriate error messages are sent to the operator.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

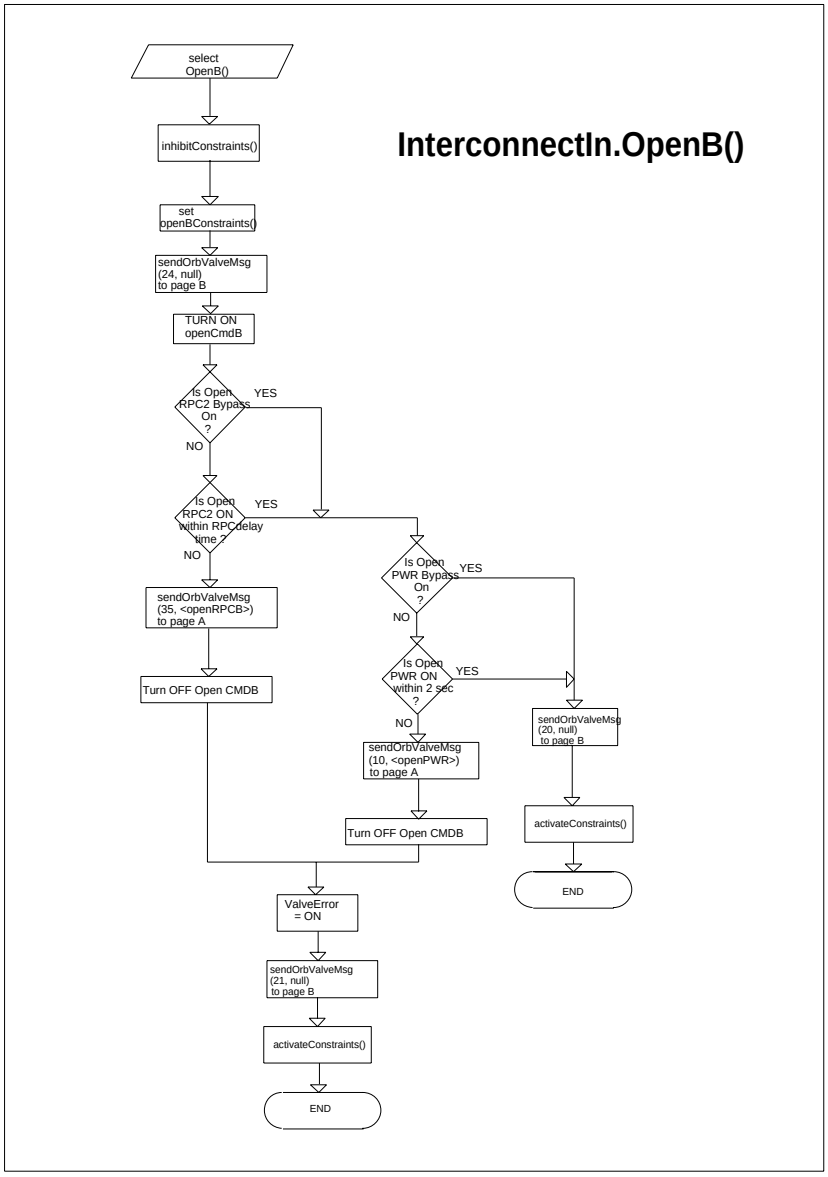


3.1.21.3.3. openB() Method

3.1.21.3.3.1. Description

The open method performs steps necessary to energize the interconnect-in 2 way solenoid valves using open command B only. The method verifies all expected RPCs and power indications are received otherwise open commmand B is removed and appropriate error messages are sent to the operator.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

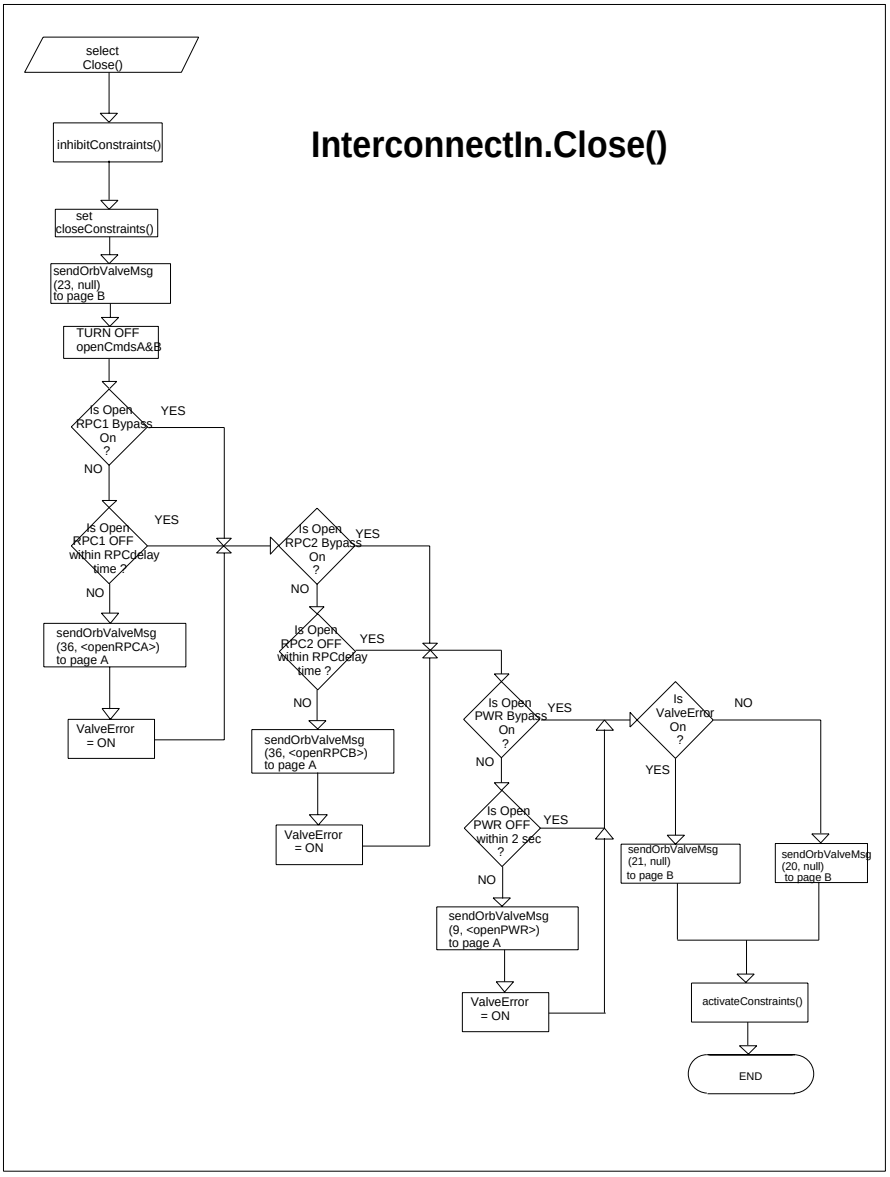


3.1.21.3.4. Close() Method

3.1.21.3.4.1. Description

The open method performs steps necessary to deenergize the interconnect-in 2 way solenoid valves by removing open commands. Constraints on RPCs and Power indicators are maintained and appropriate error messages are sent to the operator (where applicable).

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



3.1.21.3.5. determineCmdState()

3.1.21.3.5.1. Descriptive Text

This method will determine the valve's commanded state based on the valve open power and open RPC 1 and RPC 2 indicators. The return value will be an enumerated type with possible values of ENERGIZED, DEENERGIZED, and ERROR.

3.1.21.3.5.2. Effectivity

Operates on change data.

3.1.21.3.5.3. Sequence

Case:

<openPwr> OFF

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

<openPwr>.BYP OFF
<openRPC_1> OFF
<openRPC_1>.BYP OFF
<openRPC_2> OFF
<openRPC_2>.BYP OFF
<cmdState> = DEENERGIZED;

<openPwr> OFF
<openPwr>.BYP OFF
<openRPC_1> OFF
<openRPC_1>.BYP OFF
<openRPC_2>.BYP ON
<cmdState> = DEENERGIZED

<openPwr> OFF
<openPwr>.BYP OFF
<openRPC_1>.BYP ON
<openRPC_2> OFF
<openRPC_2>.BYP OFF
<cmdState> = DEENERGIZED;

<openPwr> OFF
<openPwr>.BYP OFF
<openRPC_1>.BYP ON
<openRPC_2>.BYP ON
<cmdState> = DEENERGIZED;

<openPwr>.BYP ON
<openRPC_1> OFF
<openRPC_1>.BYP OFF
<openRPC_2> OFF
<openRPC_2>.BYP OFF
<cmdState> = DEENERGIZED;

<openPwr>.BYP ON
<openRPC_1> OFF
<openRPC_1>.BYP OFF
<openRPC_2>.BYP ON
<cmdState> = DEENERGIZED;

<openPwr>.BYP ON
<openRPC_1>.BYP ON
<openRPC_2> OFF
<openRPC_2>.BYP OFF
<cmdState> = DEENERGIZED;

<openPwr>.BYP ON
<openRPC_1>.BYP ON
<openRPC_2> ON
<openRPC_2>.BYP OFF
<cmdState> = ENERGIZED;

<openPwr>.BYP ON

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

<openRPC_1> ON
<openRPC_1>.BYP OFF
<openRPC_2>.BYP ON
<cmdState> = ENERGIZED;

<openPwr>.BYP ON
<openRPC_1> ON
<openRPC_1>.BYP OFF
<openRPC_2> ON
<openRPC_2>.BYP OFF
<cmdState> = ENERGIZED;

<openPwr> ON
<openPwr>.BYP OFF
<openRPC_1>.BYP ON
<openRPC_2>.BYP ON
<cmdState> = ENERGIZED;

<openPwr> ON
<openPwr>.BYP OFF
<openRPC_1>.BYP ON
<openRPC_2> ON
<openRPC_2>.BYP OFF
<cmdState> = ENERGIZED;

<openPwr> ON
<openPwr>.BYP OFF
<openRPC_1> ON
<openRPC_1>.BYP OFF
<openRPC_2>.BYP ON
<cmdState> = ENERGIZED;

<openPwr> ON
<openPwr>.BYP OFF
<openRPC_1> ON
<openRPC_1>.BYP OFF
<openRPC_2> ON
<openRPC_2>.BYP OFF
<cmdState> = ENERGIZED;

<openPwr> ON
<openPwr>.BYP OFF
<openRPC_1> ON
<openRPC_1>.BYP OFF
<openRPC_2> OFF
<openRPC_2>.BYP OFF
<cmdState> = ENERGIZED;

<openPwr> ON
<openPwr>.BYP OFF
<openRPC_1> OFF
<openRPC_1>.BYP OFF
<openRPC_2> ON
<openRPC_2>.BYP OFF

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
                <cmdState> = ENERGIZED;


        DEFAULT CASE
                ELSE            ERROR;
```

End Case:

3.1.21.3.6. getCmdState()

3.1.21.3.6.1. Descriptive Text

This method will return the Enumerated Command State of the valve.

3.1.21.3.6.2. Sequence

```
getCmdState()
{
        return cmdState;
}
```

3.1.21.3.7. determineIndState()

3.1.21.3.7.1. Descriptive Text

This method will determine valve state based on power indication. For the two way solenoid valves, the valve state determination is the same functionally to command state: 'energized' is equal to 'open' and 'deenergized' is equal to 'closed'.

3.1.21.3.7.2. Sequence

Case:

```
        <cmdState> = ENERGIZED
                IndState = OPEN;

        <cmdState> = DEENERGIZED
                IndState = CLOSED;

        <cmdState> = ERROR
                IndState = ERROR;
```

End Case;

3.1.21.3.8. verifyCommandsOff()

3.1.21.3.8.1. Descriptive Text

This function will verify that both the open commands are OFF.

3.1.21.3.8.2. Sequence

```
verifyCommandsOff()
{
        open command status1 = byteTest4 <Open Command A>
        verify  open command status1 == OFF
        ELSE
                Print message to MPS Message Page: valveID:" open command A" <openCMDA> "is ON"

        open command status2 = byteTest4 <Open Command B>
        verify  open command status2 == OFF
        ELSE
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

Print message to MPS Message Page: valveID:" open command B" <openCMDB> "is ON"

{
verify open command status1 && open command status2 == OFF
return TRUE
}

ELSE return FALSE
}

3.1.21.3.9. setCloseConstraints()

3.1.21.3.9.1. Descriptive Text

This method changes the constraint criteria of the measurements associated with this valve to those expected with the valve in the closed position.

3.1.21.3.9.2. Sequence

setCloseConstraints()
{
CHANGE <openPwr> CONSTRAINT CRITERIA TO OFF;
CHANGE <openRPC1> CONSTRAINT CRITERIA TO OFF;
CHANGE <openRPC2> CONSTRAINT CRITERIA TO OFF;
}

3.1.21.3.10. setOpenConstraints()

3.1.21.3.10.1. Descriptive Text

This method changes the constraint criteria of the measurements associated with this valve to those expected with the valve in the open position when commanded with A and B open commands.

3.1.21.3.10.2. Sequence

setOpenConstraints()
{

IF <openPwr>Bypass == OFF;
CHANGE <openPwr> CONSTRAINT CRITERIA TO ON;
END IF;

IF <openRPC1>Bypass == OFF;
CHANGE <openRPC1> CONSTRAINT CRITERIA TO ON;
END IF;

IF <openRPC2>Bypass == OFF;
CHANGE <openRPC2> CONSTRAINT CRITERIA TO ON;
END IF;

}

3.1.21.3.11. setOpenAConstraints()

3.1.21.3.11.1. Descriptive Text

This method changes the constraint criteria of the measurements associated with this valve to those expected with the valve in the open position when commanded with A command.
setOpenAConstraints()

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
{
CHANGE <openPwr> CONSTRAINT CRITERIA TO ON;
END IF;

IF <openRPC1>Bypass == OFF;
CHANGE  <openRPC1> CONSTRAINT CRITERIA TO ON;
END IF;

}
```

3.1.21.3.12. setOpenBConstraints()

3.1.21.3.12.1. Descriptive Text

This method changes the constraint criteria of the measurements associated with this valve to those expected with the valve in the open position when commanded with B command.

```
setOpenBConstraints()
{

IF <openPwr>Bypass == OFF;
CHANGE <openPwr> CONSTRAINT CRITERIA TO ON;
END IF;

IF <openRPC2>Bypass == OFF;
CHANGE  <openRPC2> CONSTRAINT CRITERIA TO ON;
END IF;

}
```

3.1.21.3.13. activateConstraints()

3.1.21.3.13.1. Descriptive Text

This operation activates constraint monitoring on valve position and power indicators.

3.1.21.3.13.2. Sequence

```
activateConstraints()
{

IF <openPwr>Bypass == OFF;
Activate Constraint Monitoring on
        <openPwr>

IF <openRPC1>Bypass == OFF;
Activate Constraint Monitoring on
        <openRPC1>

IF <openRPC2>Bypass == OFF;
Activate Constraint Monitoring on
<openRPC2>
}
```

3.1.21.3.14. inhibitConstraints()

3.1.21.3.14.1. Descriptive Text

This operation inhibits constraint monitoring on valve position and power indicators.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.21.3.14.2. Sequence

```
inhibitConstraints()
{
        Inhibit Constraint Monitoring on
                <openPwr>
                <openRPC1>
                <openRPC2>
}
}
```

3.1.21.3.15. verifyTminus6BT4()//

3.1.21.3.15.1. Descriptive Text

This function will verify that both the open commands are OFF

3.1.21.3.15.2. Sequence

```
verifyTminus6BT4()
{
        open command status1 = byteTest4 <Open Command A>
        verify  open command status1 == OFF
        ELSE
                Print message to MPS Message Page: valveID:" open command A" <openCMDA> "is ON"

        open command status2 = byteTest4 <Open Command B>
        verify  open command status2 == OFF
        ELSE
                Print message to MPS Message Page: valveID:" open command B" <openCMDB> "is ON"


        {
        verify  open command status1 && open command status2  == OFF
        return TRUE
        }

        ELSE return FALSE
}
```

3.1.21.3.16. verifyTminus3BT4()//

3.1.21.3.16.1. Descriptive Text

This function will verify that both the open commands are OFF

3.1.21.3.16.2. Sequence

```
verifyTminus3BT4()

{
//performs identical T-6 hour method//

        verifyTminus6BT4();

}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.1.21.4. User Interfaces

3.1.21.5. Safety Requirements

3.1.22. MPS_SolValve_ORB_2Way

3.1.22.1. Description

This orbiter valkve class inherits from MPS_SolValve_ORB. The valve is a 2-way solenoid valve used in the MPS system and serve as an in-line shut-off valve. Each valve of this class is a normally closed solenoid valve (spring loaded closed) with a single open command and a single power indication.

3.1.22.1.1. Instance Index

Instance	Display User	Other Users
MPS_LV10	MPS	
MPS_LV26	MPS	
MPS_LV27	MPS	
MPS_LV40	MPL	
MPS_LV41	MPL	
MPS_LV42	MPS	
MPS_LV43	MPS	
MPS_LV52	MPS	

3.1.22.1.2. Class Diagram

<insert class diagram here>

3.1.22.1.3. Attributes and Aggregations

Attribute	Data Type	Initial Value	Publish / Persistence	Client Access	Description
openCmd	DS_ON_OFF			R,W	
openPwr	DM_ON_OFF			R	

3.1.22.1.4. Operations Index

Operation	Description
open()	This method opens (energizes) 2 way solenoid valve.
close()	This method closes (deenergizes)two way solenoid valve.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

determineCmdState()	This method determines the enumerated command state for the solenoid based upon the power indicator.
getCmdState()	This method returns enumerated command state.
determineState()	This method determines the enumerated valve state for the solenoid based upon the power indicator.
getState()	This method returns enumerated valve state.
setOpenConstraints()	This method sets constraint monitoring criteria for an energized/open solenoid valve.
setClosedConstraints()	This method sets constraint monitoring criteria for an deenergized/closed solenoid valve.
verifyCmdsOff()	This method does a byte test 4 to verify valve command is OFF.
activateConstraints()	This method activates constraint monitoring on open power indicator (if unbypassed).
inhibitConstraints()	This method inhibits constraint monitoring on open power indicator
//verifyTminus6BT4()//	This method does a byte test 4 to verify valve command is OFF at T-6 hours.
//verifyTminus3BT4()//	This method does a byte test 4 to verify valve command is OFF at T-3 hours.

3.1.22.1.5. Associations

Association	Type	Class	Description

3.1.22.2. State Description

3.1.22.2.1. State Diagram

<insert state diagram here>

3.1.22.2.2. State Table

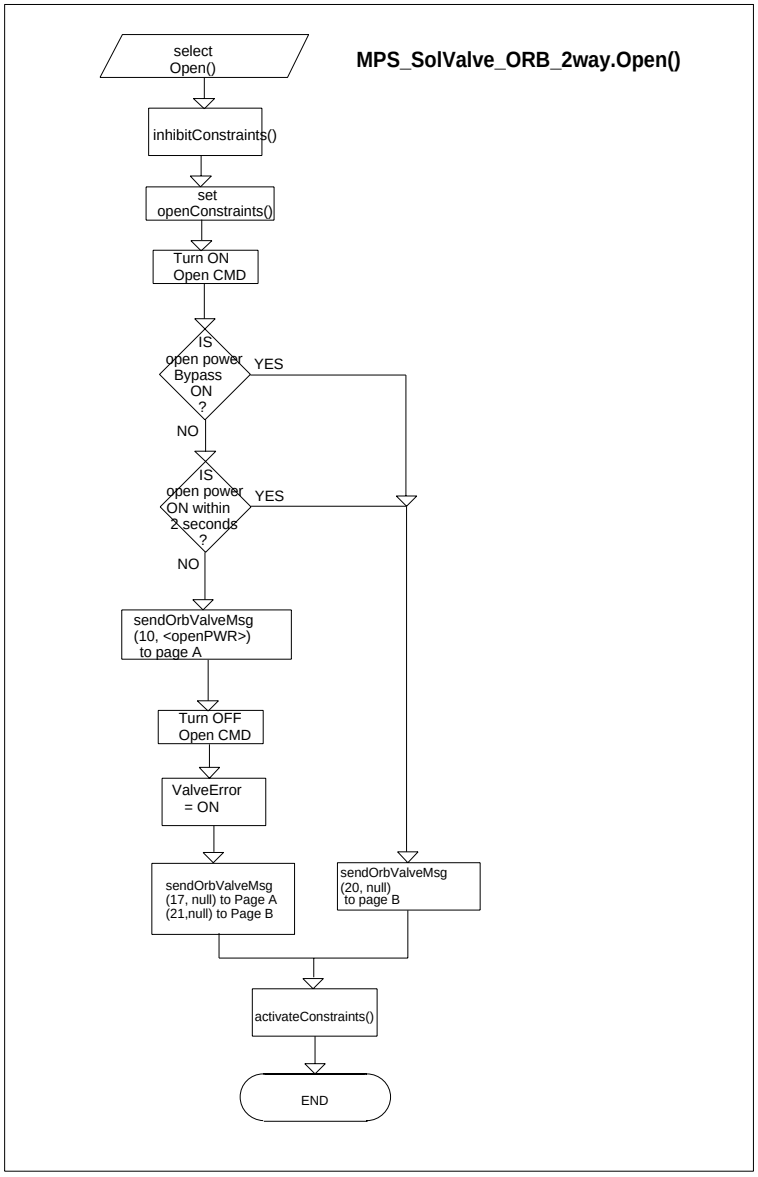
3.1.22.3. Operations

3.1.22.3.1. open() Method

3.1.22.3.1.1. Description

This method performs all operations associated with opening (energizing) the MPS two way solenoid valves. Constraints are maintained on the valve open power indicator. Open power will be removed if the power indicator does not come on within two seconds (unless it is bypassed).

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

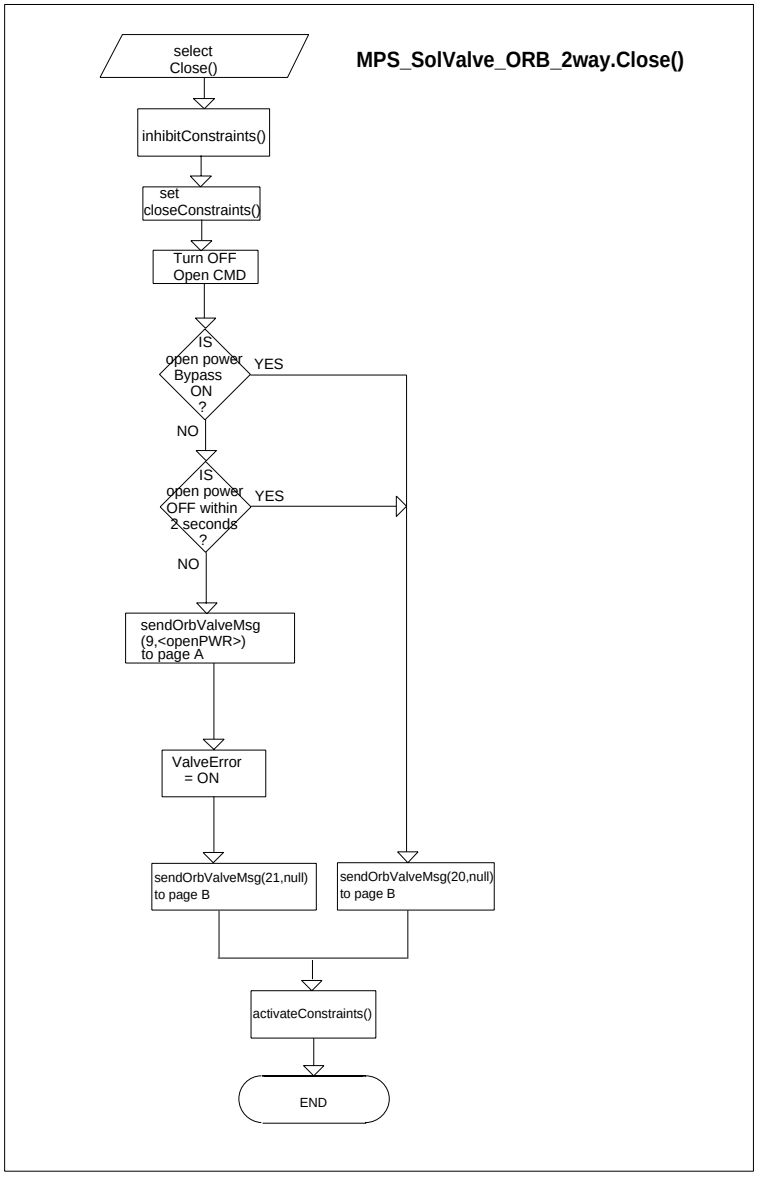


3.1.22.3.2. close() Method

3.1.22.3.2.1. Description

This method performs all operations associated with closing (deenergizing) the MPS high pressure helium system interconnect out valves. Constraints are maintained on the valve open power indicator.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



3.1.22.3.3. determineCmdState()

3.1.22.3.3.1. Descriptive Text

This method will determine the valve's commanded state based on the valve open power FD. The return value will be an enumerated type with possible values of ENERGIZED, DEENERGIZED, and ERROR.

3.1.22.3.3.2. Effectivity

Operates on change data.

3.1.22.3.3.3. Sequence

Case:

<openPwr> ON : cmdState = ENERGIZED;

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

<openPwr> OFF : cmdState = DEENERGIZED;

<openPwr>Bypass ON : cmdState = ERROR;

End Case:

3.1.22.3.4. getCmdState()

3.1.22.3.4.1. Descriptive Text

This method will return the Enumerated Command State of the valve.

3.1.22.3.4.2. Sequence

```
getCmdState()
{
        return cmdState;
}
```

3.1.22.3.5. determineState()

3.1.22.3.5.1. Descriptive Text

This method will determine valve state based on power indication. For the two way solenoid valves, the valve state determination is the same functionally to command state: 'energized' is equal to 'open' and 'deenergized' is equal to 'closed'. The return value will be an enumerated type with possible values of OPEN, CLOSED, and ERROR.

3.1.22.3.5.2. Sequence

Case:

<openPwr> ON : State = OPEN;

<openPwr> OFF : State = CLOSED;

<openPwr.Bypass> ON : State = ERROR;

End Case:

3.1.22.3.6. getState()

3.1.22.3.6.1. Descriptive Text

This method will return the Enumerated State of the valve.

3.1.22.3.6.2. Sequence

```
getState()
{
        return State;
}
```

3.1.22.3.7. verifyCommandsOff()

3.1.22.3.7.1. Descriptive Text

This function will verify that the open command is OFF.

3.1.22.3.7.2. Sequence

```
verifyCommandsOff()
{
        open command status = byteTest4 <Open Command>
        {
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
        verify  open command status == OFF

        return TRUE
        }
        ELSE return FALSE
        and send message to MPS message display:
                valveID "open command" <openCMD> "is ON, should be OFF>."

}
```

3.1.22.3.8. setCloseConstraints()

3.1.22.3.8.1. Descriptive Text

This method changes the constraint criteria of the measurements associated with this valve to those expected with the valve in the closed position.

3.1.22.3.8.2. Sequence

```
setCloseConstraints()
        {

                CHANGE <openPwr> CONSTRAINT CRITERIA TO OFF;
        }
```

3.1.22.3.9. setOpenConstraints()

3.1.22.3.9.1. Descriptive Text

This method changes the constraint criteria of the measurements associated with this valve to those expected with the valve in the open position.

3.1.22.3.9.2. Sequence

```
setOpenConstraints()
        {

                CHANGE <openPwr> CONSTRAINT CRITERIA TO ON;
        }
```

3.1.22.3.10. activateConstraints()

3.1.22.3.10.1. Descriptive Text

This operation activates constraint monitoring on valve power indicators.

3.1.22.3.10.2. Sequence

```
activateConstraints()
{
        IF <openPwr>.Bypass is OFF;

                Activate Constraint Monitoring on:
                        <openPwr>

        END IF
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.1.22.3.11. inhibitConstraints()

3.1.22.3.11.1. Descriptive Text

This operation de-activates constraint monitoring on valve power indicators.

3.1.22.3.11.2. Sequence

```
inhibitConstraints()
{
        Inhibit Constraint Monitoring on
                <openPwr>
}
```

3.1.22.3.12. //verifyTminus6BT4()//

3.1.22.3.12.1. Description

This method verifies open command is off at T-6 hours.

```
verifyTminus6BT4();
{
        open command status = byteTest4 <Open Command>
        {
        verify  open command status == OFF

        return TRUE
        }
        ELSE return FALSE
        and send message to MPS message display:
                valveID "open command" <openCMD> "is ON."

}
```

3.1.22.3.13. //verifyTminus3BT4()//

3.1.22.3.13.1. Description

This method verifies open command is off at T-3 hours.

```
verifyTminus3BT4();
{
//calls T-6 hour method due to identical requirements.//

        verifyTminus6BT4();
}
```

3.1.22.4. User Interfaces

3.1.22.5. Safety Requirements

3.1.23. MPS_SolValve_ORB_2Way_IntOut

3.1.23.1. Description

This orbiter valkve class inherits from MPS_SolValve_ORB_2Way. The valve is a 2-way solenoid valve used in the MPS pneumatic supply system and serves as an in-line interconnect valve to tie-in the high pressure pneumatic bottles. Each valve of this class is a normally closed solenoid valve (spring loaded closed) with a single open command and a single power indication. Per OMRS, the valve require the delta P between the MPS and SSME tanks to be less than 200 psi prior to cycling open (applies only if SSME>MPS).

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.1.23.1.1. Instance Index

Instance	Display User	Other Users
MPS_LV60	MPS	
MPS_LV62	MPS	
MPS_LV64	MPS	

3.1.23.1.2. Class Diagram

<insert class diagram here>

3.1.23.1.3. Attributes and Aggregations

Attribute	Data Type	Initial Value	Publish / Persistence	Client Access	Description

3.1.23.1.4. Operations Index

Operation	Description
deltaPcheck()	This method verifies the delta P across the valve is less than 200 psi prior to opening (energizing) the interconnect_out solenoid valve.
open()	This method opens (energizes) interconnect_out solenoid valve.

3.1.23.1.5. Associations

Association	Type	Class	Description
MPS_MT55		MPS_PRESSURE_SENSOR	
MPS_MT(Engine)		MPS_PRESSURE_SENSOR	

3.1.23.2. State Description

3.1.23.2.1. State Diagram

<insert state diagram here>

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

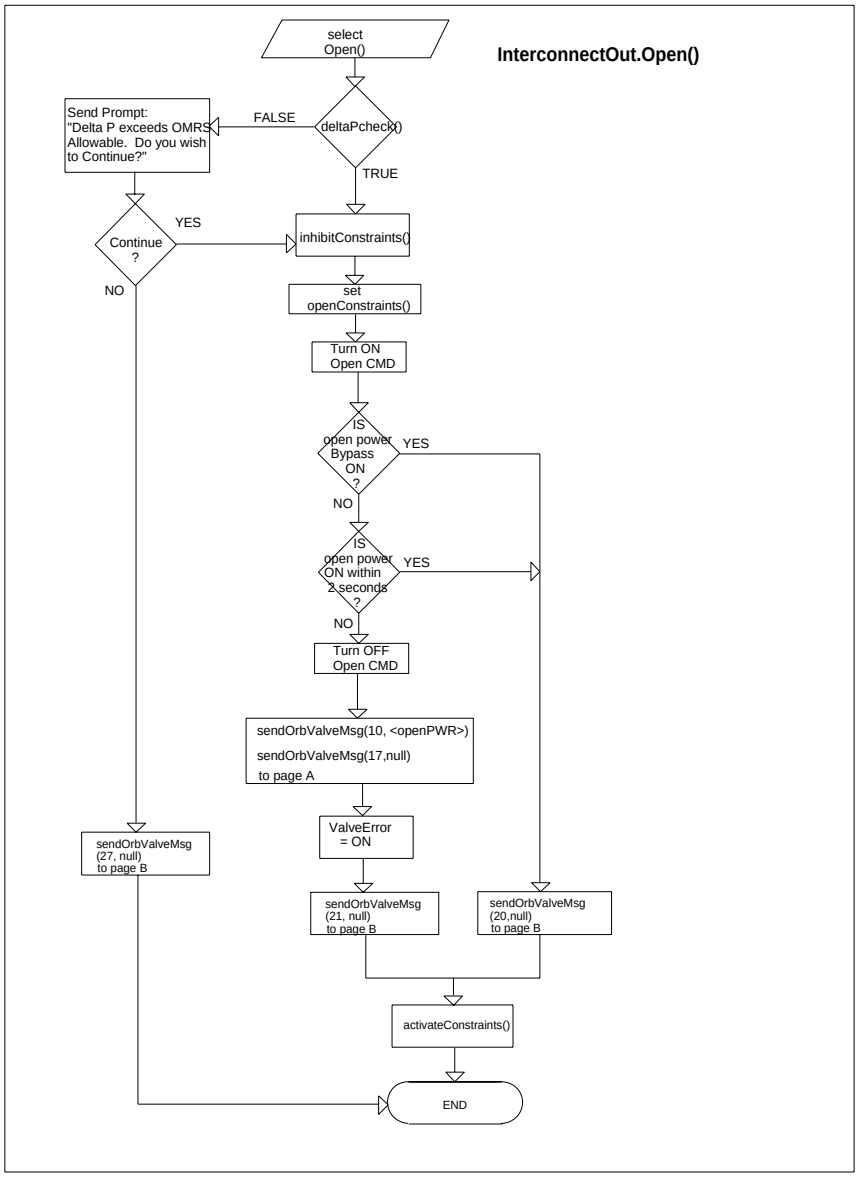
3.1.23.2.2. State Table

3.1.23.3. Operations

3.1.23.3.1. open() Method

3.1.23.3.1.1. Description

This method performs all operations associated with opening (energizing) the MPS high pressure helium system interconnect out valves. Constraints are maintained on the valve open power indicator. Open power will be removed if the power indicator does not come on within two seconds (unless it is bypassed). A check of the pressure differential between the MPS helium tank and the associated SSME tank is done to ensure less than 200 psi differential between tanks (applies only if MPS<SSME) by running deltaPcheck() method.



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.23.3.2. deltaPcheck() Method

3.1.23.3.2.1. Description

This method performs a check of the pressure differential between the MPS helium tank and the associated SSME tank to ensure less than 200 psi differential between tanks (applies only if MPS<SSME). This method is called by open() prior to issuing the open command.

3.1.23.3.2.2. Sequence

begin sequence;
deltaPcheck();

//E1 Tank Pressure Transducer is MPS_MT8
//E2 Tank Pressure Transducer is MPS_MT21
//E3 Tank Pressure Transducer is MPS_MT34

get MPS_ORB_PRESSURE_SENSOR.MT55
and save as mpstankpress

get MPS_ORB_PRESSURE_SENSOR.E#
and save as smetankpress

verify smetankpress - mpstankpress < 200 psia
and return TRUE

ELSE send message to page A:
"Pressure differential between SME tank and MPS tank exceeds 200 psi."
and return FALSE

end sequence;

3.1.23.4. User Interfaces

3.1.23.5. Safety Requirements

3.1.24. MPS_SolValve_ORB_RTLSPress

3.1.24.1. Description

3.1.24.1.1. Instance Index

Instance	Display User	Other Users
MPS_LV74	MPS	
MPS_LV75	MPS	

3.1.24.1.2. Class Diagram

<insert class diagram here>

3.1.24.1.3. Attributes and Aggregations

Attribute	Data Type	Initial	Publish /	Client	Description

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

		Value	Persistence	Access	
openCmdA	DS_ON_OFF				
openCmdB	DS_ON_OFF				
openCmdC	DS_ON_OFF				
openPwr	DM_ON_OFF				
openRpcA	DM_ON_OFF				
openRpcC	DM_ON_OFF				

3.1.24.1.4. Operations Index

Operation	Description
open()	This method opens (energizes) RTLS press solenoid valve.
close()	This method closes (deenergizes) RTLS press solenoid valve.
determineCmdState()	This method determines the enumerated command state for the solenoid based upon the power indicator.
getCmdState()	This method returns enumerated command state.
determineState()	This method determines the enumerated valve state for the solenoid based upon the power indicator.
getState()	This method returns enumerated valve state.
setOpenConstraints()	This method sets constraint monitoring criteria for an energized/open solenoid valve.
setClosedConstraints()	This method sets constraint monitoring criteria for an deenergized/closed solenoid valve.
verifyCmdsOff()	This method does a byte test 4 to verify valve command is OFF.
activateConstraints()	This method activates constraint monitoring on open power indicator (if unbypassed).
inhibitConstraints()	This method inhibits constraint monitoring on open power indicator
//verifyTminus6BT4()//	This method does a byte test 4 to verify valve command is OFF at T-6 hours.
//verifyTminus3BT4()//	This method does a byte test 4 to verify valve command is OFF at T-3 hours.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.24.2. State Description

3.1.24.2.1. State Diagram

<insert state diagram here>

3.1.24.2.2. State Table

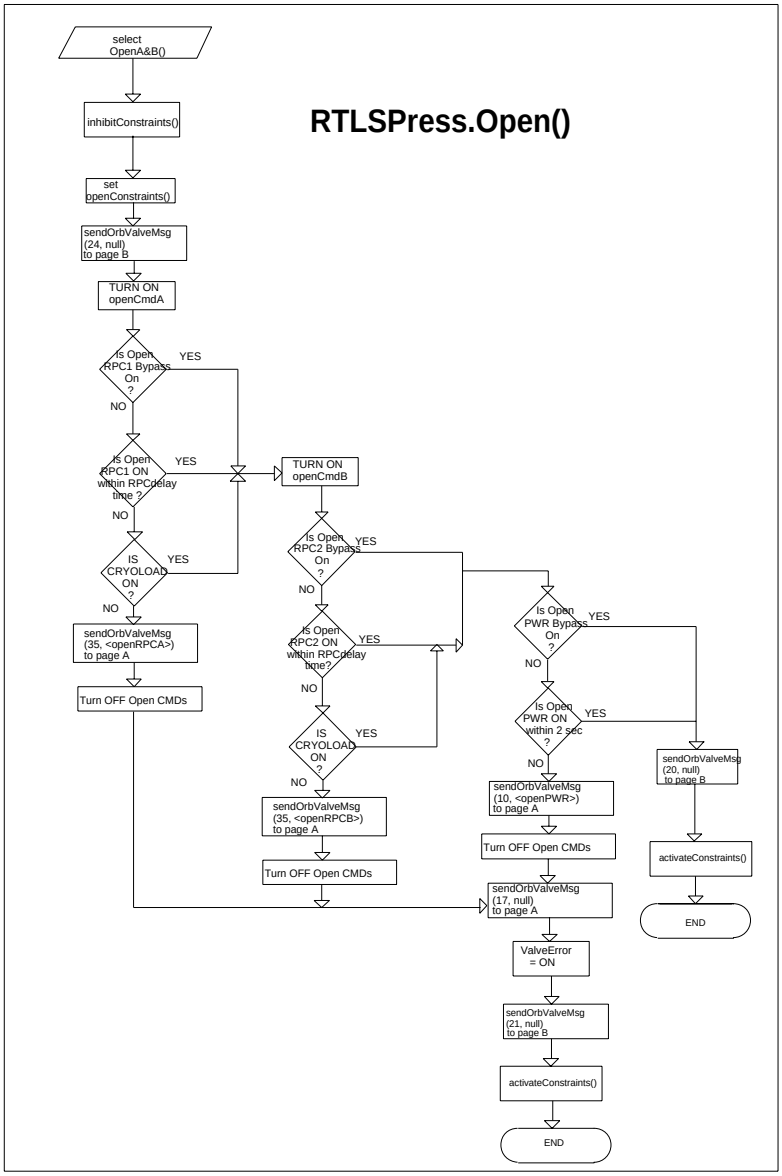
3.1.24.3. Operations

3.1.24.3.1. Open Method()

3.1.24.3.1.1. Descriptive

The open method performs steps necessary to energize (open) the RTLS press solenoid valves. For test and check-out (CRYOflag is OFF), the method verifies all expected RPCs and power indications are received otherwise open commmands are removed (whimp-out option) and appropriate error messages are sent to the operator.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

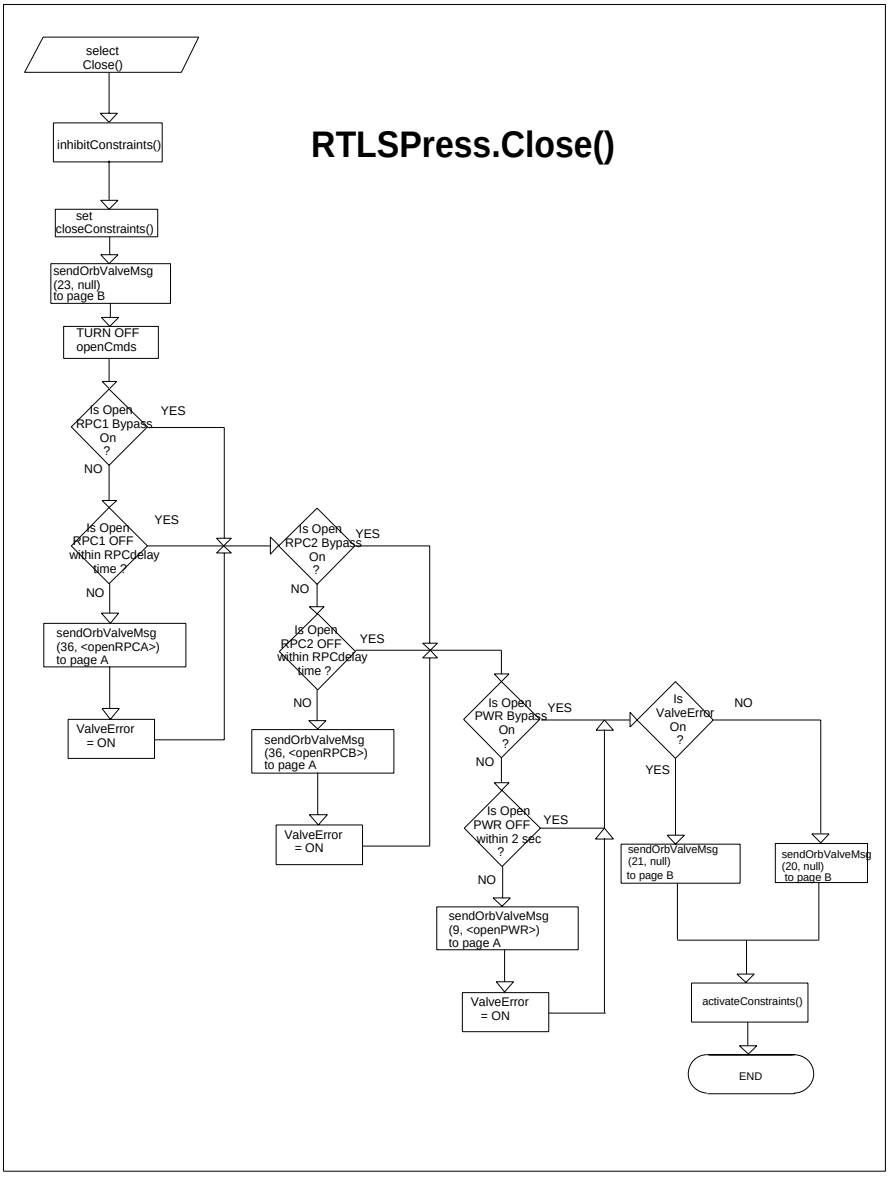


3.1.24.3.2. Close() Method

3.1.24.3.2.1. Description

The open method performs steps necessary to deenergize (close) the RTLS press solenoid valves. For test and check-out (CRYOflag is OFF), the method verifies all expected RPCs and power indications are received else appropriate error message(s) are sent to the operator.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



3.1.24.3.3. determineCmdState()

3.1.24.3.3.1. Descriptive Text

This method will determine the valve's commanded state based on the valve open power and open RPC A and RPC C indicators. The return value will be an enumerated type with possible values of ENERGIZED, DEENERGIZED, and ERROR.

3.1.24.3.3.2. Effectivity

Operates on change data.

3.1.24.3.3.3. Sequence

Case:

<openPwr> OFF

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

<openPwr>.BYP OFF
<openRPCA> OFF
<openRPCA>.BYP OFF
<openRPCC> OFF
<openRPCC>.BYP OFF
<cmdState> = DEENERGIZED;

<openPwr> OFF
<openPwr>.BYP OFF
<openRPCA> OFF
<openRPCA>.BYP OFF
<openRPCC>.BYP ON
<cmdState> = DEENERGIZED

<openPwr> OFF
<openPwr>.BYP OFF
<openRPCA>.BYP ON
<openRPCC> OFF
<openRPCC>.BYP OFF
<cmdState> = DEENERGIZED;

<openPwr> OFF
<openPwr>.BYP OFF
<openRPCA>.BYP ON
<openRPCC>.BYP ON
<cmdState> = DEENERGIZED;

<openPwr>.BYP ON
<openRPCA> OFF
<openRPCA>.BYP OFF
<openRPCC> OFF
<openRPCC>.BYP OFF
<cmdState> = DEENERGIZED;

<openPwr>.BYP ON
<openRPCA> OFF
<openRPCA>.BYP OFF
<openRPCC>.BYP ON
<cmdState> = DEENERGIZED;

<openPwr>.BYP ON
<openRPCA>.BYP ON
<openRPCC> OFF
<openRPCC>.BYP OFF
<cmdState> = DEENERGIZED;

<openPwr>.BYP ON
<openRPCA>.BYP ON
<openRPCC> ON
<openRPCC>.BYP OFF
<cmdState> = ENERGIZED;

<openPwr>.BYP ON

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
        <openRPCA>              ON
        <openRPCA>.BYP          OFF
        <openRPCC>.BYP          ON
                <cmdState> = ENERGIZED;

        <openPwr>.BYP ON
        <openRPCA>              ON
        <openRPCA>.BYP          OFF
        <openRPCC>              ON
        <openRPCC>.BYP          OFF
                <cmdState> = ENERGIZED;

        <openPwr>               ON
        <openPwr>.BYP OFF
        <openRPCA>.BYP          ON
        <openRPCC>.BYP          ON
                <cmdState> = ENERGIZED;

        <openPwr>               ON
        <openPwr>.BYP OFF
        <openRPCA>.BYP          ON
        <openRPCC>              ON
        <openRPCC>.BYP          OFF
                <cmdState> = ENERGIZED;

        <openPwr>               ON
        <openPwr>.BYP OFF
        <openRPCA>              ON
        <openRPCA>.BYP          OFF
        <openRPCC>.BYP          ON
                <cmdState> = ENERGIZED;

        <openPwr>               ON
        <openPwr>.BYP OFF
        <openRPCA>              ON
        <openRPCA>.BYP          OFF
        <openRPCC>              ON
        <openRPCC>.BYP          OFF
                <cmdState> = ENERGIZED;

    DEFAULT CASE
                ELSE            ERROR;

End Case:
```

3.1.24.3.4. getCmdState()

3.1.24.3.4.1. Descriptive Text

This method will return the Enumerated Command State of the valve.

3.1.24.3.4.2. Sequence

```
getCmdState()
{
        return cmdState;
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.24.3.5. determineState()

3.1.24.3.5.1. Descriptive Text

This method will determine valve state based on power indication. For the two way solenoid valves, the valve state determination is the same functionally to command state: 'energized' is equal to 'open' and 'deenergized' is equal to 'closed'. cmdState will be an eneumerated state FD with possible states of ENERGIZED, DEENERGIZED, and ERROR.

3.1.24.3.5.2. Sequence

Case:

<openPwr> OFF
<openPwr>.BYP OFF
<openRPCA> OFF
<openRPCA>.BYP OFF
<openRPCC> OFF
<openRPCC>.BYP OFF
<cmdState> = DEENERGIZED;

<openPwr> OFF
<openPwr>.BYP OFF
<openRPCA> OFF
<openRPCA>.BYP OFF
<openRPCC>.BYP ON
<cmdState> = DEENERGIZED

<openPwr> OFF
<openPwr>.BYP OFF
<openRPCA>.BYP ON
<openRPCC> OFF
<openRPCC>.BYP OFF
<cmdState> = DEENERGIZED;

<openPwr> OFF
<openPwr>.BYP OFF
<openRPCA>.BYP ON
<openRPCC>.BYP ON
<cmdState> = DEENERGIZED;

<openPwr>.BYP ON
<openRPCA> OFF
<openRPCA>.BYP OFF
<openRPCC> OFF
<openRPCC>.BYP OFF
<cmdState> = DEENERGIZED;

<openPwr>.BYP ON
<openRPCA> OFF
<openRPCA>.BYP OFF
<openRPCC>.BYP ON
<cmdState> = DEENERGIZED;

<openPwr>.BYP ON
<openRPCA>.BYP ON

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
<openRPCC>              OFF
<openRPCC>.BYP          OFF
        <cmdState> = DEENERGIZED;


<openPwr>.BYP ON
<openRPCA>.BYP          ON
<openRPCC>              ON
<openRPCC>.BYP          OFF
        <cmdState> = ENERGIZED;

<openPwr>.BYP ON
<openRPCA>              ON
<openRPCA>.BYP          OFF
<openRPCC>.BYP          ON
        <cmdState> = ENERGIZED;

<openPwr>.BYP ON
<openRPCA>              ON
<openRPCA>.BYP          OFF
<openRPCC>              ON
<openRPCC>.BYP          OFF
        <cmdState> = ENERGIZED;

<openPwr>               ON
<openPwr>.BYP OFF
<openRPCA>.BYP          ON
<openRPCC>.BYP          ON
        <cmdState> = ENERGIZED;

<openPwr>               ON
<openPwr>.BYP OFF
<openRPCA>.BYP          ON
<openRPCC>              ON
<openRPCC>.BYP          OFF
        <cmdState> = ENERGIZED;

<openPwr>               ON
<openPwr>.BYP OFF
<openRPCA>              ON
<openRPCA>.BYP          OFF
<openRPCC>.BYP          ON
        <cmdState> = ENERGIZED;

<openPwr>               ON
<openPwr>.BYP OFF
<openRPCA>              ON
<openRPCA>.BYP          OFF
<openRPCC>              ON
<openRPCC>.BYP          OFF
        <cmdState> = ENERGIZED;

DEFAULT CASE
        ELSE            ERROR;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

End Case:

3.1.24.3.6. verifyCommandsOff()

3.1.24.3.6.1. Descriptive Text

This function will verify that both the open commands are OFF

3.1.24.3.6.2. Sequence

```
verifyCommandsOff()
{
        open command status1 = byteTest4 <Open Command A>
        open command status2 = byteTest4 <Open Command B>
        open command status3 = byteTest4 <Open Command C>
        {
        verify  open command status1 == OFF
        Else send message to MPS message display
                ValveID "open command A" <open CMDA> "is ON, should be OFF"
        }
        {
        verify  open command status2 == OFF
        Else send message to MPS message display
                ValveID "open command B" <open CMDB> "is ON, should be OFF"
        }
        {
        verify  open command status3 == OFF
        Else send message to MPS message display
                ValveID "open command C" <open CMDC> "is ON, should be OFF"
        }
        {
        verify  open command status1 && open command status2 && open command status3 == OFF

        return TRUE
        }
        ELSE return FALSE
}
```

3.1.24.3.7. setValveCloseConstraints()

3.1.24.3.7.1. Descriptive Text

This method changes the constraint criteria of the measurements associated with this valve to those expected with the valve in the closed position.

3.1.24.3.7.2. Sequence

```
setCloseConstraints()
        {
                CHANGE <openPwr> CONSTRAINT CRITERIA TO OFF;
                CHANGE <openRPCA> CONSTRAINT CRITERIA TO OFF;
                CHANGE <openRPCC> CONSTRAINT CRITERIA TO OFF;
        }
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.1.24.3.8. setValveOpenConstraints()

3.1.24.3.8.1. Descriptive Text

This method changes the constraint criteria of the measurements associated with this valve to those expected with the valve in the open position.

3.1.24.3.8.2. Sequence

```
setOpenConstraints()
        {

        IF <openPwr>Bypass == OFF;
        CHANGE <openPwr> CONSTRAINT CRITERIA TO ON;
        END IF;

        IF <openRPCA>Bypass == OFF;
        CHANGE  <openRPCA> CONSTRAINT CRITERIA TO ON;
        END IF;

        IF <openRPCC>Bypass == OFF;
        CHANGE  <openRPCC> CONSTRAINT CRITERIA TO ON;
        END IF;

        }
```

3.1.24.3.9. activateConstraints()

3.1.24.3.9.1. Descriptive Text

This operation activates constraint monitoring on valve position and power indicators.

3.1.24.3.9.2. Sequence

```
activateConstraints()
{
        Activate Constraint Monitoring on
                <openPwr>
                <openRPCA>
                <openRPCC>
}
```

3.1.24.3.10. inhibitConstraints()

3.1.24.3.10.1. Descriptive Text

This operation inhibits constraint monitoring on valve position and power indicators.

3.1.24.3.10.2. Sequence

```
inhibitConstraints()
{
        Inhibit Constraint Monitoring on
                <openPwr>
                <openRPCA>
                <openRPCC>
}
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.1.24.4. User Interfaces

3.1.24.5. Safety Requirements

3.1.25. MPS_ORB_Pressure_Sensor_wBias

3.1.25.1. Description

MPS_ORB_Pressure_Sensor is an abstract representation of an MPS system pressure transducer which senses fluid system pressure.

3.1.25.1.1. MPS LH2 System Instance Index

Instance	Display User	Description	EMON Upper Limit
MPS_MT11	MPS	E1 GH2 Press	425
MPS_MT14	MPS	E2 LH2 Feedline Press	50
MPS_MT24	MPS	E2 GH2 Press	425
MPS_MT27	MPS	E3 LH2 Feedline Press	50
MPS_MT37	MPS	E3 GH2 Press	425
MPS_MT43	MPS	LH2 Manifold Press	50
MPS_MT44	MPS	LH2 Delta Press	3.25
MPS_MT47	MPS	GH2 Disc Press	425
MPS_MT54	MPS	E1 LH2 Feedline Press	50

3.1.25.1.2. MPS LO2 System Instance Index

Instance	Display User	Description	EMON Upper Limit
MPS_MT5	MPL	E1 LO2 Feedline Press	65
MPS_MT18	MPL	E2 LO2 Feedline Press	65
MPS_MT31	MPL	E3 LO2 Feedline Press	65
MPS_MT49	MPL	LO2 Manifold Press	65
MPS_MT50	MPL	LO2 Delta Press	25
MPS_MT53	MPL	GO2 Disc Press	425

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.25.1.3. MPS GHe System Instance Index

Instance	Display User	Description	EMON Upper Limit
MPS_MT8	MPS	E1 Helium Supply Press	4500
MPS_MT10	MPS	E1 Reg A Outlet Press	800
MPS_MT21	MPS	E2 Helium Supply Press	4500
MPS_MT23	MPS	E2 Reg A Outlet Press	800
MPS_MT34	MPS	E3 Helium Supply Press	4500
MPS_MT36	MPS	E3 Reg A Outlet Press	800
MPS_MT55	MPS	MPS Pneu Supply Press	4500
MPS_MT55	MPS	MPS Reg Outlet Press	800
MPS_MT74	MPS	E1 Reg B Outlet Press	800
MPS_MT75	MPS	E2 Reg B Outlet Press	800
MPS_MT76	MPS	E3 Reg B Outlet Press	800
MPS_MT518	MPS	MPS Accum Press	800

3.1.25.1.4. Class Diagram

<insert class diagram here>

3.1.25.1.5. Attributes and Aggregations

Attribute	Data Type	Initial Value	Publish / Persistence	Client Access	Description
press	AM_PSIA			R	This is the pressure FD
ambientBias	double		Persistent. KEY = Vehicle Tail number	R	This is the Bias offset
biasCorrectedValue	PA_PSIA			R	This will the the Pseudo that has the corrected pressure reading
emonUpperLimit	double			R	An EMON Constraint will be created with NO Low Limit and this value as the upper limit
bluelineLowerLimit	double			R	A Display Attribute will be created with this value as the

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

					lower limit.
bluelineUpperLimit	double			R	A Display Attribute will be created with this value as the upper limit.

3.1.25.1.6. Operations Index

Operation	Description
setPowerUpConstraints()	This method will initiate constraint monitoring on pressure transducers for MPS power-up for test.
setPowerDownConstraints()	This method will discontinue constraint monitoring on pressure transducers for MPS power-down from test.
applyBiasCorrection()	This method will remove ambient bias from transducer output value.
changeBiasValue()	This method allows the operator to alter the bias value for a given transducer in the bias table for test and check-out.
setMonitorLimits()	This method allows the operator to set monitor 'blue line' limits for a given transducer test and check-out.

3.1.25.1.7. Associations

Association	Type	Class	Description

3.1.25.2. State Description

3.1.25.2.1. State Diagram

<insert state diagram here>

3.1.25.2.2. State Table

3.1.25.3. Operations

3.1.25.3.1. setPowerUpConstraints()

3.1.25.3.1.1. Description

This method activates constraint monitoring on MPS pressure transducers.

3.1.25.3.1.2. Sequence

setPowerDownConstraints();
{
activate constraint monitoring on <pressureFD>
}

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.25.3.2. setPowerDownConstraints()

3.1.25.3.2.1. Description

This method terminates constraint monitoring on MPS pressure transducers.

3.1.25.3.2.2. Sequence

setPowerDownConstraints();
{
inhibit constraint monitoring on <pressureFD>
}

3.1.25.3.3. applyBiasCorrection()

3.1.25.3.3.1. Description

This method applies a bias correction factor for MPS pressure transducers for use during ambient conditions test and check-out. This operation will work on Change Data.

3.1.25.3.3.2. Sequence

Sequence;
//Calculate bias corrected pressure
{
//Read <pressureFD>
//assign pseudo value

<biasCorrectedValue>.Apply(<press>.getValue() - ambientBias)

// <NMPSMTxx> = (Read Value) - ambientBias

Return biasCorrectedValue
}
end sequence;

3.1.25.3.4. changeBiasValue(newBias)

3.1.25.3.4.1. Parameters

INPUT : double newBias

3.1.25.3.4.2. Description

This method allows operator to input new bias value for a given MPS pressure transducer due to component change-out or ambient bias drift.

3.1.25.3.4.3. Sequence

ambiantBias = newBias

3.1.25.3.5. setMonitorLimits(newLow, newHigh)

3.1.25.3.5.1. Description

This method allows the operator to set monitor 'blue line' limits for a given transducer to support test and check-out. These limits will go into a Display Attribute for Both the press and biasCorrectedValue.

3.1.25.3.5.2. Paramenters

INPUT: double newLow
INPUT: double newHigh

3.1.25.3.5.3. Sequence

Alter the Display attribute with the limits being newLow and newHigh

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.25.4. User Interfaces

3.1.25.5. Safety Requirements

3.1.26. MPS_ORB_Temp_Sensor

3.1.26.1. Description

3.1.26.1.1. Instance Index

Instance	Display User	Other Users
MPS_MT12	MPS	
MPS_MT25	MPS	
MPS_MT38	MPS	
MPS_MT42	MPS	
MPS_MT9	MPS	
MPS_MT22	MPS	
MPS_MT35	MPS	
MPS_MT56	MPS	
MPS_MT6	MPL	
MPS_MT13	MPL	
MPS_MT19	MPL	
MPS_MT26	MPL	
MPS_MT32	MPL	
MPS_MT39	MPL	
MPS_MT48	MPL	
MPS_MT512	MPL	

3.1.26.1.2. Class Diagram

<insert class diagram here>

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.1.26.1.3. Attributes and Aggregations

Attribute	Data Type	Initial Value	Publish / Persistence	Client Access	Description

3.1.26.1.4. Operations Index

Operation	Description

3.1.26.1.5. Associations

Association	Type	Class	Description

3.1.26.2. State Description

3.1.26.2.1. State Diagram

<insert state diagram here>

3.1.26.2.2. State Table

3.1.26.3. Operations

3.1.26.3.1. Name of Operation

3.1.26.3.1.1. Parameters

3.1.26.3.1.2. Effectivity

3.1.26.3.1.3. Prerequisites

3.1.26.3.1.4. Sequence

3.1.26.3.1.5. Post-Requisites

3.1.26.3.1.6. State Diagram

<insert state diagram here>

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.1.26.3.1.7. State Table

3.1.26.3.1.8. Descriptive Text

3.1.26.4. User Interfaces

3.1.26.5. Safety Requirements

3.1.27. MPS_ORB_SpeedSensor//VAB/PAD Class//

3.1.27.1. Description

3.1.27.1.1. Instance Index

Instance	Display User	Other Users
MPS_PP1	MPS	
MPS_PP2	MPS	
MPS_PP3	MPS	

3.1.27.1.2. Class Diagram

<insert class diagram here>

3.1.27.1.3. Attributes and Aggregations

Attribute	Data Type	Initial Value	Publish / Persistence	Client Access	Description

3.1.27.1.4. Operations Index

Operation	Description

3.1.27.1.5. Associations

Association	Type	Class	Description

3.1.27.2. State Description

3.1.27.2.1. State Diagram

<insert state diagram here>

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.1.27.2.2. State Table

3.1.27.3. Operations

3.1.27.3.1. Name of Operation

3.1.27.3.1.1. Parameters

3.1.27.3.1.2. Effectivity

3.1.27.3.1.3. Prerequisites

3.1.27.3.1.4. Sequence

3.1.27.3.1.5. Post-Requisites

3.1.27.3.1.6. State Diagram

<insert state diagram here>

3.1.27.3.1.7. State Table

3.1.27.3.1.8. Descriptive Text

3.1.27.4. User Interfaces

3.1.27.5. Safety Requirements

3.1.28. MPS_ORB_Analog_wLimits

3.1.28.1. Description

The details of this class can be found in the MPS SRS 1 Module

3.2. COMPOSITE CLASSES

3.2.1. MPS_Recirc

3.2.1.1. Description

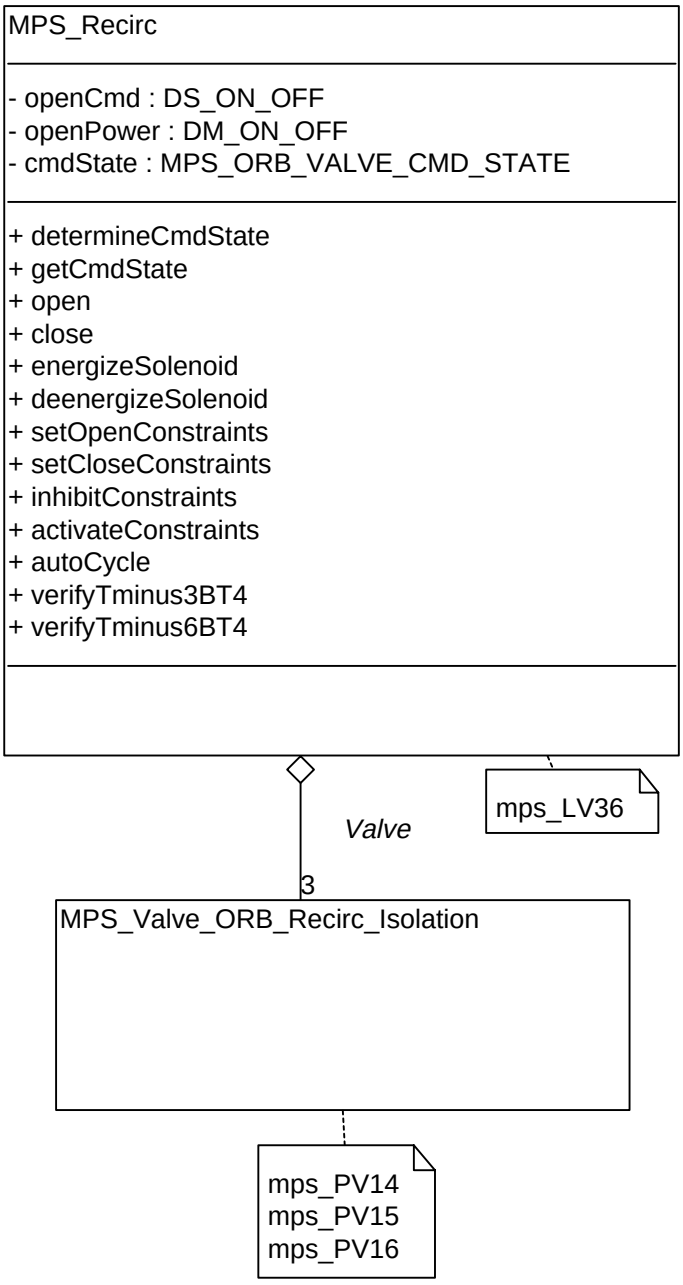
3.2.1.1.1. Instance Index

Instance	Other Users
none	GLS

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.2.1.1.2. Class Diagram

MPS_Recirc



3.2.1.1.3. Attributes and Aggregations

Attribute	Data Type	Initial	Publish /	Client	Description

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

		Value	Persistence	Access	
openCmd	DS_ON_OFF				LV36 Open Power Cmd FD
openPower	DM_ON_OFF				LV36 Open Power Ind FD
cmdState	MPS_ORB_VALVE_CMD_STATE				

3.2.1.1.4. Operations Index

Operation	Description
open()	This method opens the recirc valves.
close()	This method closes the recirc valves.
energizeSolenoid()	This method energizes LV36 for test and checkout.
deenergizeSolenoid()	This method deenergizes LV36 for test and checkout.
determineCmdState()	This method determines the enumerated command state of the recircs valves.
getCmdState()	This method returns the enumerated command state.
verifyCmdsOff()	
autocycle()	
setOpenConstraints()	
setCloseConstraints()	
activateConstraints()	
inhibitConstraints()	
verifyTminus6BT4()	
verifyTminus3BT4()	

3.2.1.1.5. Associations

Association	Type	Class	Description

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.2.1.2. State Description

3.2.1.2.1. State Diagram

<insert state diagram here>

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

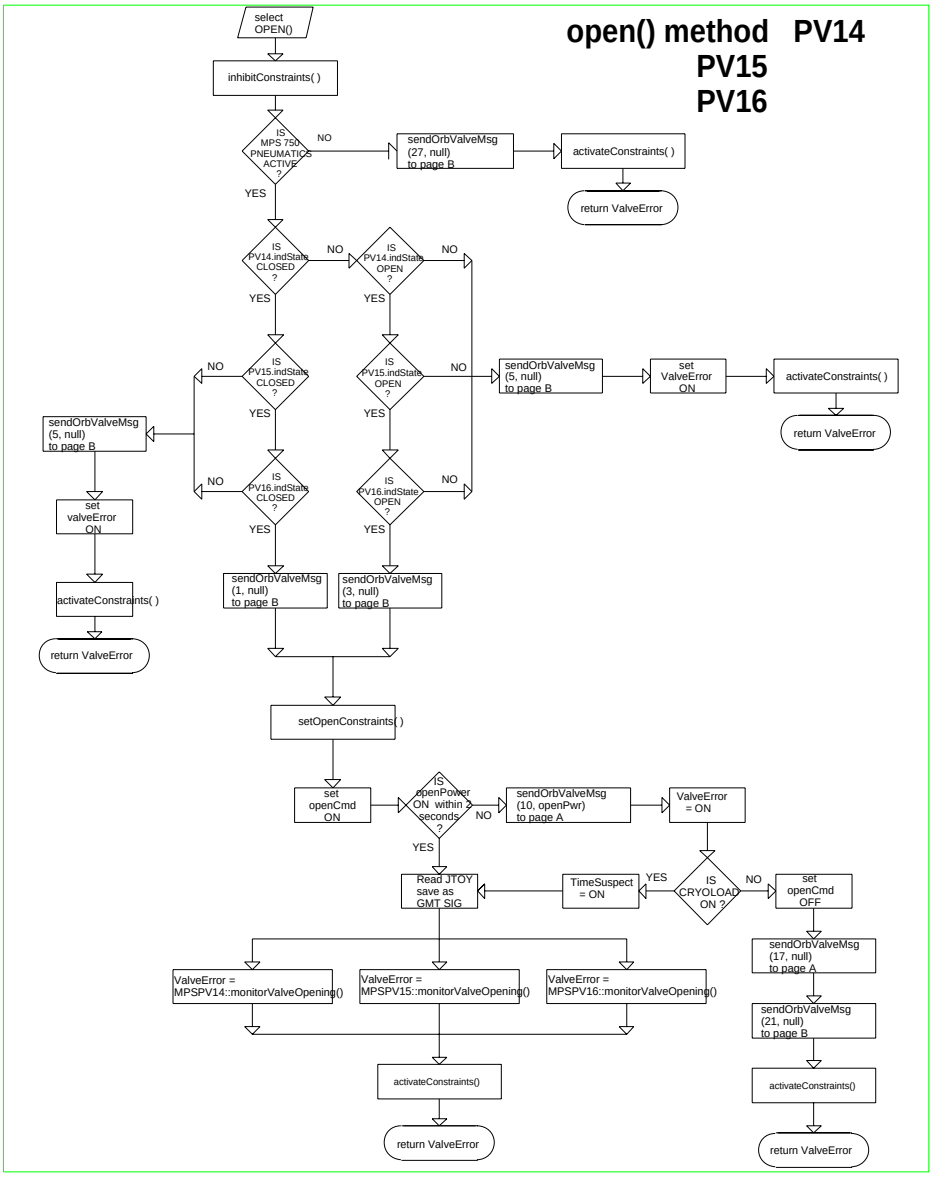
3.2.1.2.2. State Table

3.2.1.3. Operations

3.2.1.3.1. open()

3.2.1.3.1.1. Descriptive Text

3.2.1.3.1.2. Sequence

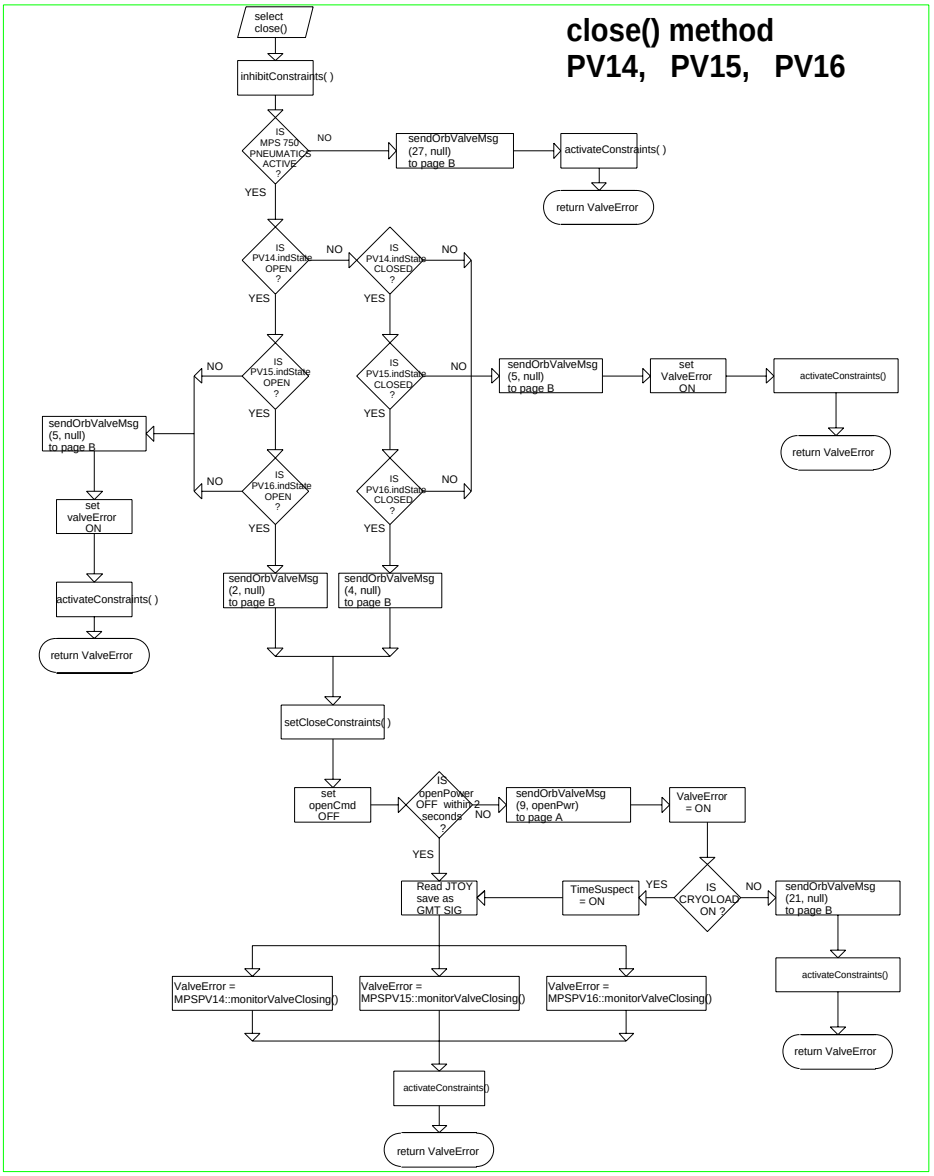


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.2.1.3.2. close()

3.2.1.3.2.1. Descriptive Text

3.2.1.3.2.2. Sequence



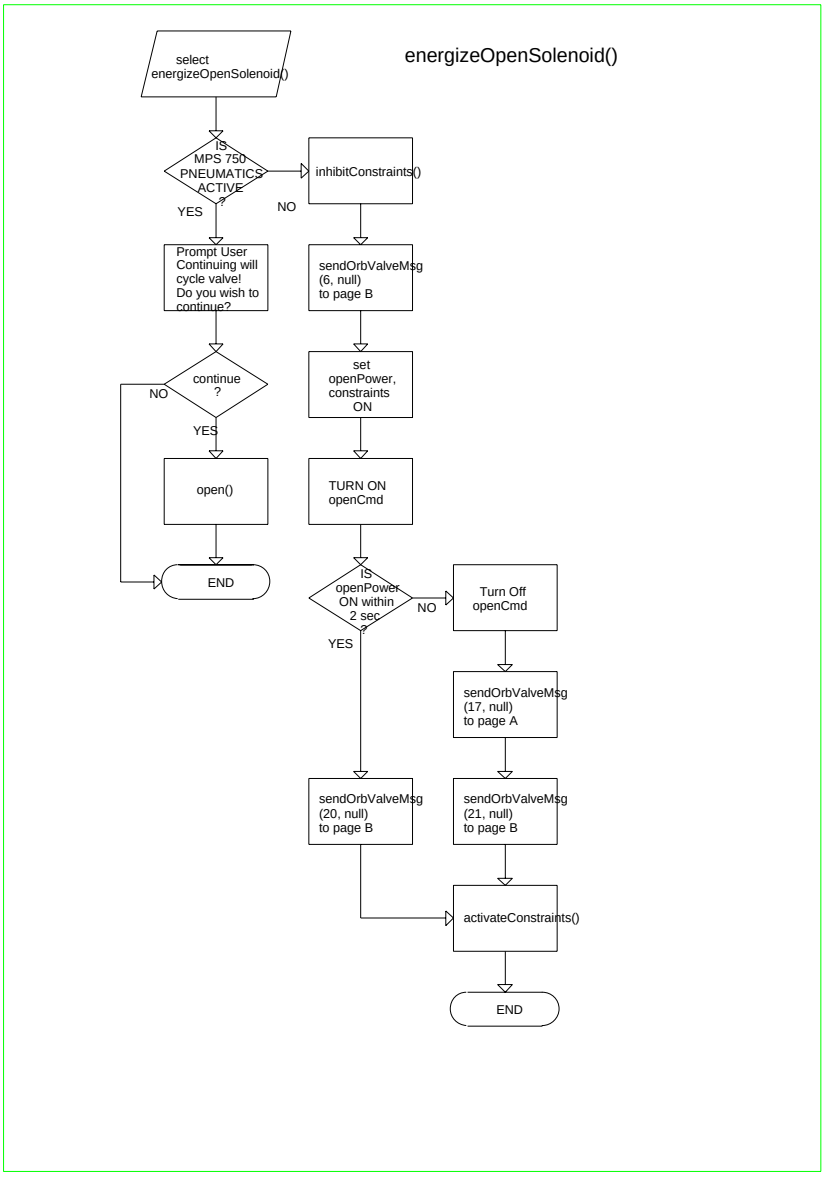
3.2.1.3.3. energizeOpenSolenoid()

3.2.1.3.3.1. Descriptive Text

This method performs applies power to the open solenoid. If MPS 750 pneumatics are active the open() method is called. The flow chart provides the necessary logic for the method, including checks for the proper response of power indicators. The valve cycles in a fail safe mode and 'wimps-out' (all commands removed) at the first indication of hardware failure

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.2.1.3.3.2. Sequence



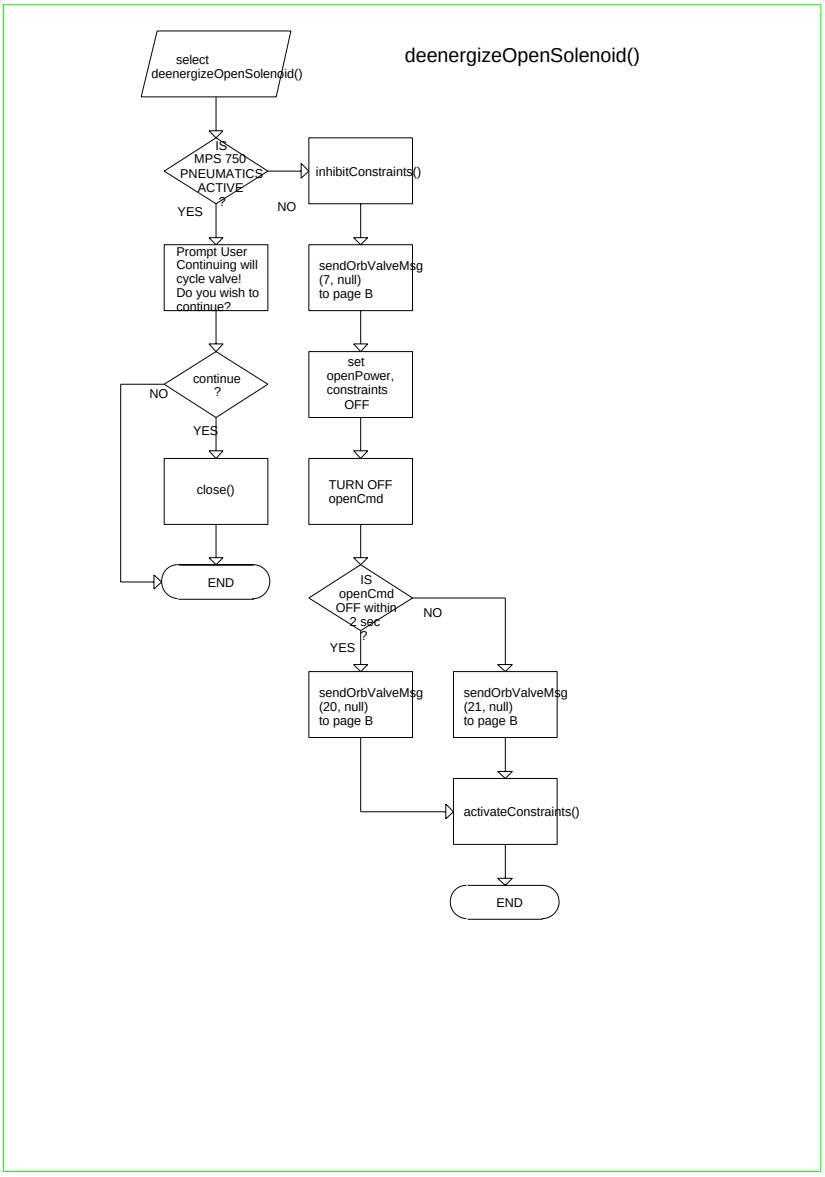
3.2.1.3.4. deenergizeOpenSolenoid()

3.2.1.3.4.1. Descriptive Text

This method performs removes power from the open solenoid. If MPS 750 pneumatics are active the close() method is called. The flow chart provides the necessary logic for the method, including checks for the proper response of power indicators. The valve cycles in a fail safe mode and 'wimps-out' (all commands removed) at the first indication of hardware failure The implementation of the details is left up to the coders and is acceptable as long as the overall logic is maintained. It is expected that cycle timing will be accomplished using the most accurate source. All messages will include a time stamp, and the appropriate FD description.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.2.1.3.4.2. Sequence



3.2.1.3.5. determineCmdState()

3.2.1.3.5.1. Descriptive Text

This method will determine the valve's commanded state based on the HIM Relay Status FDs and the close command indicator. The return value will be an enumerated type with possible values of CMD_OPEN, CMD_CLOSED and ERROR.

3.2.1.3.5.2. Effectivity

Operates on change data.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.2.1.3.5.3. Sequence

Case:

<openPwr> ON : cmdState = CMD_OPEN;

<openPwr> OFF : cmdState = CMD_CLOSED;

End Case:

Command states

open power	state
0	cmd closed
1	cmd open

3.2.1.3.6. getCmdState()

3.2.1.3.6.1. Descriptive Text

This method will return the Enumerated Command State of the valve.

3.2.1.3.6.2. Sequence

```
getCmdState()
{
        return cmdState;
}
```

3.2.1.3.7. autoCycle()

3.2.1.3.7.1. Descriptive Text

The following will be called by the Panel Class autoValveCheckout method to calculate and record the valve cycle times.

3.2.1.3.7.2. Sequence

Set autoValveCheckoutError = OFF;

close();

if valveCycleError is ON, Set autoValveCheckoutError = ON;

open();

if valveCycleError is ON, Set autoValveCheckoutError = ON;

Return autoValveCheckoutError;

3.2.1.3.8. activateConstraints()

3.2.1.3.8.1. Descriptive Text

This method will activate constraint monitoring of all unbypassed measurement of this valve.

3.2.1.3.8.2. Sequence

```
activateConstraints()
        {
        if(<openPwr>.BYP == OFF)
                {
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
            activate constraint monitoring for <openPwr>
            }
        if(<MPS_PV14.openInd>.BYP == OFF)
            {
            activate constraint monitoring for <MPS_PV14.openInd>
            }
        if(<MPS_PV15.openInd>.BYP == OFF)
            {
            activate constraint monitoring for <MPS_PV15.openInd>
            }
        if(<MPS_PV16.openInd>.BYP == OFF)
            {
            activate constraint monitoring for <MPS_PV16.openInd>
            }
        if(<MPS_PV14.closeInd>.BYP == OFF)
            {
            activate constraint monitoring for <MPS_PV14.closeInd>
            }
        if(<MPS_PV15.closeInd>.BYP == OFF)
            {
            activate constraint monitoring for <MPS_PV15.closeInd>
            }
        if(<MPS_PV16.closeInd>.BYP == OFF)
            {
            activate constraint monitoring for <MPS_PV16.closeInd>
            }


        }
```

3.2.1.3.9. inhibitConstraints()

3.2.1.3.9.1. Descriptive Text

This method will inhibit constraint monitoring of all measurements related to this valve.

3.2.1.3.9.2. Sequence

```
inhibitConstraints()
        {
        inhibit constraint monitoring of the following:

        <openPwr>
        <MPS_PV14.openInd>
        <MPS_PV15.openInd>
        <MPS_PV16.openind>
        <MPS_PV14.closeInd>
        <MPS_PV15.closeInd>
        <MPS_PV16.closeind>
        }
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.2.1.3.10. setOpenConstraints()

3.2.1.3.10.1. Descriptive Text

This method changes the constraint criteria of the measurements associated with this valve to those expected with the valve in the open position.

3.2.1.3.10.2. Sequence

```
void setOpenConstraints()
{

        CHANGE (openPower)  CONSTRAINT CRITERIA TO ON;
         CHANGE (MPS_PV14.openInd)  CONSTRAINT CRITERIA TO ON;
         CHANGE (MPS_PV15.openInd)  CONSTRAINT CRITERIA TO ON;
         CHANGE (MPS_PV16.openInd)  CONSTRAINT CRITERIA TO ON;
         CHANGE (MPS_PV14.closeInd)  CONSTRAINT CRITERIA TO OFF;
         CHANGE (MPS_PV15.closeInd)  CONSTRAINT CRITERIA TO OFF;
         CHANGE (MPS_PV16.closeInd)  CONSTRAINT CRITERIA TO OFF;



}
```

3.2.1.3.11. setClosedConstraints()

3.2.1.3.11.1. Descriptive Text

This method changes the constraint criteria of the measurements associated with this valve to those expected with the valve in the closed position.

3.2.1.3.11.2. Sequence

```
void setCloseConstraints()
{

        CHANGE (openPower)  CONSTRAINT CRITERIA TO OFF;
         CHANGE (MPS_PV14.openInd)  CONSTRAINT CRITERIA TO OFF;
         CHANGE (MPS_PV15.openInd)  CONSTRAINT CRITERIA TO OFF;
         CHANGE (MPS_PV16.openInd)  CONSTRAINT CRITERIA TO OFF;
         CHANGE (MPS_PV14.closeInd)  CONSTRAINT CRITERIA TO ON;
         CHANGE (MPS_PV15.closeInd)  CONSTRAINT CRITERIA TO ON;
         CHANGE (MPS_PV16.closeInd)  CONSTRAINT CRITERIA TO ON;



}
```

3.2.1.3.12. autoCycle()

3.2.1.3.12.1. Descriptive Text

This method will check the present state of the valve, then cycle the valve to the opposite state and back to the original state using the open() and close() methods to perform an autocycle check

3.2.1.3.12.2. Sequence

```
{

    If  instance IndState == CLOSED
            open();
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
                If valveCycleError is ON, set autoValveCheckoutError = ON
                close();
                If valveCycleError is ON, set autoValveCheckoutError = ON


        If  instance IndState == OPEN
                close();
                If valveCycleError is ON, set autoValveCheckoutError = ON
                open();
                If valveCycleError is ON, set autoValveCheckoutError = ON

Return autoValveCheckoutError;

}
```

3.2.1.3.13. verifyCmdsOff()

3.2.1.3.13.1. Descriptive Text

This function will verify that both the open and close commands are OFF.

3.2.1.3.13.2. Sequence

```
verifyCommandsOff()
{
        open command status = byteTest4 <Open Command>
                verify  open command status == OFF
                else send message ",valveID, open command;<openCMD>; " is ON, should be OFF" to Page A

        close command status = byteTest4 <Close Command>
                verify close command status == OFF
                else print message ",valveID, close command;<closeCMD>; " is ON, should be OFF" to Page A

        {
        verify  open command status && close command status == OFF

        return TRUE
        }
        ELSE return FALSE
}
```

3.2.1.4. User Interfaces

3.2.1.5. Safety Requirements

3.2.2. MPS_ORB_PneumaticRegCntl

3.2.2.1. Description

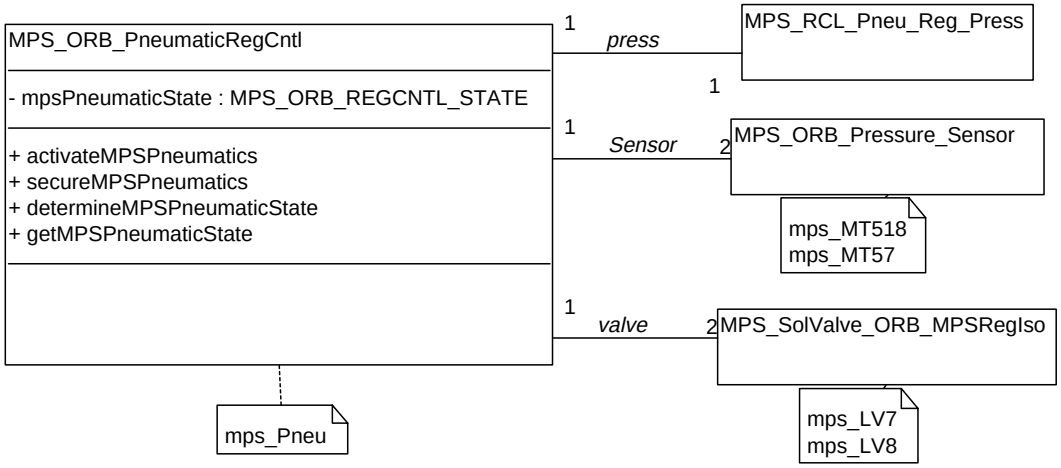
3.2.2.1.1. Instance Index

Instance	Display User	**Other Users**
MPS_Pneu	MPS	

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.2.2.1.2. Class Diagram

MPS_ORB_PneumaticRegCntl



3.2.2.1.3. Attributes and Aggregations

Attribute	Data Type	Initial Value	Publish / Persistence	Client Access	Description
mpsPneumaticState	ENUM		PUB	R	

3.2.2.1.4. Operations Index

Operation	Description
activateMPSPneumatics()	Performs steps necessary to activate MPS 750 pneumatics and maintain Constraints. In OPF, opens LV7 prior to LV8. In vertical, opens LV8 prior to LV7. OMRS pressure requirements are verified after reg isolation valves are opened.
secureMPSPneumatics()	Secures MPS pneumatics by closing reg isolation valves.
determineMPSPneumaticsState()	This method will determine the state of MPS 750 pneumatics and return: ACTIVE SECURED LOCKED_UP ERROR
getMPSPneumaticsState()	This method will return enumerated state of MPS pneumatics.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.2.2.1.5. Associations

Association	Type	Class	Description
MPS_RCL_Pneu_Reg_Press		MPS_RCL_Pneu_Reg_Press	RCL for reg over-pressure protection.
MPS_MT518		MPS_ORB_Pressure_Sensor	MPS Accum Transducer.
MPS_MT57		MPS_ORB_Pressure_Sensor	MPS Reg-Out pressure transducer.
MPS_LV7		MPS_SolValve_ORB_MPSRegIso	MPS reg isolation vlv #1.
MPS_LV8		MPS_SolValve_ORB_MPSRegIso	MPS reg isolation vlv #2.

3.2.2.2. State Description

3.2.2.2.1. State Diagram

<insert state diagram here>

3.2.2.2.2. State Table

3.2.2.3. Operations

3.2.2.3.1. activateMPSPneumatics()

3.2.2.3.1.1. Descriptive Text

This method will perform activation of MPS 750 psi pneumatics. A check of the reg outlet pressure transducer is performed to verify reg outlet pressure is within OMRS limits. RCL is inhibited prior to reg iso opening to allow for system pressure stabilization, and RCL is activated (after waiting 2 seconds) following isolation valve opening.

3.2.2.3.1.2. Sequence

begin sequence;

activateMPSPneumatics()

//open reg isolation valves//
{
MPS_RCL_Pneu_Reg_Press.inhibit()

//read MPS GHe Tank initial pressure and save//

MT55.getvalue() and save as startpress

{
//In OPF, open LV7 first. AT VAB/PAD, open LV8 first//

IF Orbiter location == OPF
{

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
                MPS_LV7.open()
                then
                MPS_LV8.open()
                }
                ELSE
                {
                MPS_LV8.open()
                then
                MPS_LV7.open()
                }
        }
//Reactivate RCL//
        MPS_RCL_Pneu_Reg_Press::activate()

//verify stable pneumatic system after activation//
//read MPS GHe Tank final pressure and save//
        {

        MPS_RCL_Pneu_Reg_Press::activate()

        wait 5 seconds

        MT55.getvalue() and save as finalpress

        verify startpress - finalpress < 200 psia
        else print message to MPS Message display:
                "Drop in MPS Pneumatic Tank exceeds expected.  Engineering evaluation is required."
                and set Valve Error ON

        {
        Verify:
                732 psia < MT57.getvalue() <784 psia
                &&
                732 psia < MT518.getvalue() <784 psia

        else set Valve ERROR ON and print message to MPS Message display:
                 "Regulated outlet pressure is not within OMRS limits of 732-784 psia"
        }
        {
        If Valve ERROR is ON, print message to Page B:
                "Error occurred during MPS pneumatics activation.  See MPS Message display for details."

        }
end sequence;
```

3.2.2.3.2. secureMPSPneumatics()

3.2.2.3.2.1. Descriptive text

This method will perform securing (closure of reg iso valves) of MPS 750 psi pneumatics.

3.2.2.3.2.2. Sequence

```
secureMPSPneumatics()

begin sequence;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

//close reg isolation valves//

MPS_LV7.close()
MPS_LV8.close()

MPS_RCL_Pneu_Reg_Press.inhibit()

Send message to Page B:
"MPS reg isolation valve closure is complete. See MPS message page for valve cycle details."

end sequence;

3.2.2.3.3. determineMPSPneuState()

This method will determine the state of MPS pneumatic system with possible states of ACTIVE, SECURED, LOCKED_UP, and ERROR.

3.2.2.3.3.1. Sequence

case;

1. NMPSLV7STATE == DEENERGIZED
   NMPSLV8STATE == DEENERGIZED
   MPS_MT57.isAmbient == TRUE
   MPS_MT518.isAmbient == TRUE

   MPSPneuState : SECURED

2. NMPSLV7STATE == ENERGIZED
   OR
   NMPSLV8STATE == ENERGIZED

   MPS_MT57.isAmbient == FALSE
   &&
   MPS_MT518.isAmbient == FALSE

   MPSPneuState = ACTIVE

3. NMPSLV7STATE == DEENERGIZED
   AND
   NMPSLV8STATE == DEENERGIZED

   MPS_MT57.isAmbient
   OR
   MPS_MT518.isAmbient
   == FALSE

   MPSPneuState : LOCKED_UP

4. NMPSLV7STATE
   OR

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

NMPSLV8STATE
== ENERGIZED

&&

MPS_MT57.isAmbient
OR
MPS_MT518.isAmbient
== TRUE

MPSPneuState : ERROR

5. Default Case : MPSPneuState : ERROR

end case;

3.2.2.3.4. getMPSPneuState()

3.2.2.3.4.1. Descriptive Text

This method will return the Enumerated State of the MPS Pneumatic system.

3.2.2.3.4.2. Sequence

```
getMPSPneuState()
{
    return MPSPneuState;
}
```

3.2.2.4. User Interfaces

3.2.2.5. Safety Requirements

3.2.3. MPS_ORB_SMERegCntl

3.2.3.1. Description

The MPS SSME Reg isolation control class provides command control over the 6 onboard MPS SSME 750 psi pneumatic regulators. This class has an association with each of the two MPS SME Reg Isolation valve classes (MPS_SolValve_ORB_SMERegIsoA and ..._SMERegIsoB). This class 'maintains' Reactive Control Logic via an association with MPS_RCL_E#_Reg_Press during reg activation and monitors regulated outlet pressure via association with MPS_ORB_Pressure_Sensor class after isolation valve opening and notifies operator of any OMRS violations.

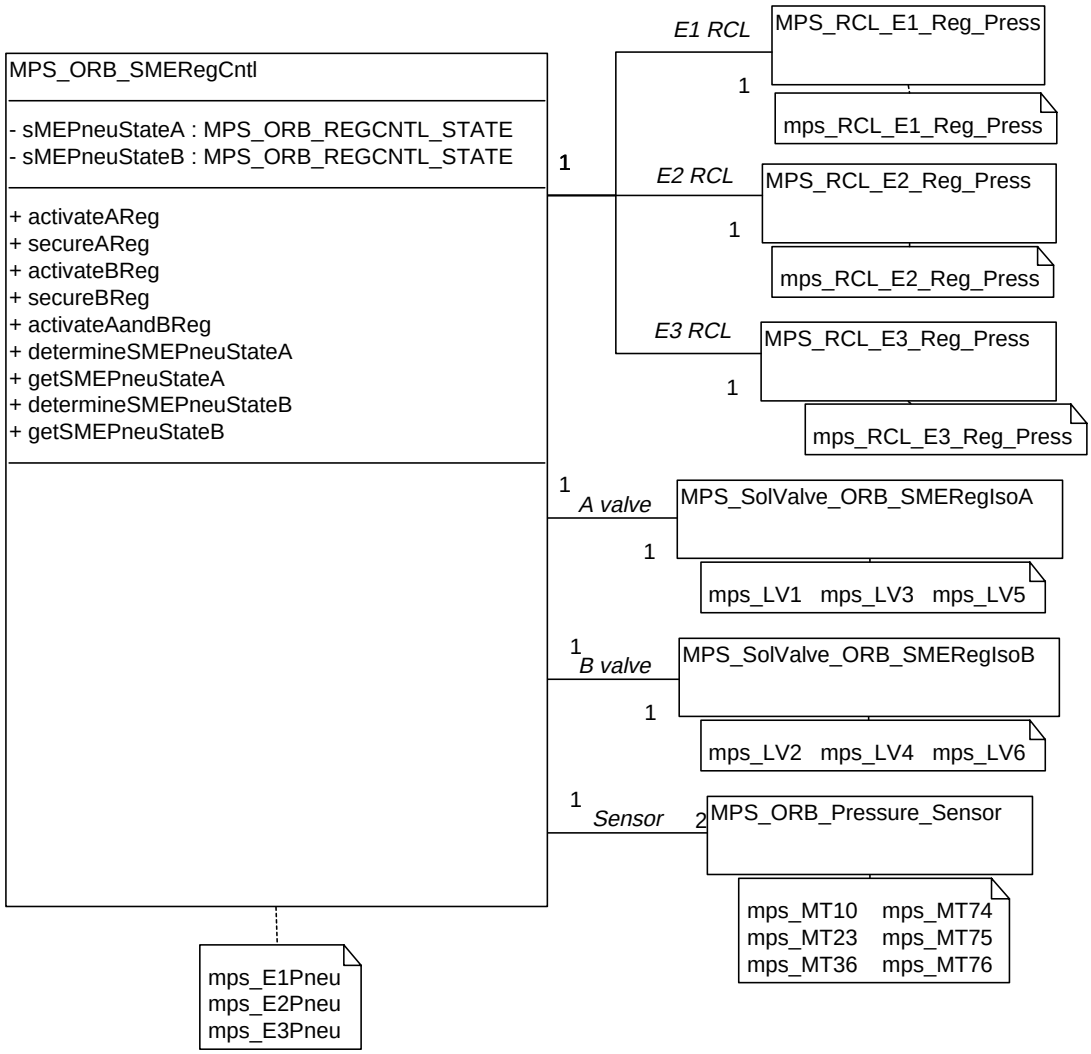
3.2.3.1.1. Instance Index

Instance	Display User	Other Users
MPS_E1Pneu	MPS	
MPS_E2Pneu	MPS	
MPS_E3Pneu	MPS	

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.2.3.1.2. Class Diagram

MPS_ORB_SMERegCntl



3.2.3.1.3. Attributes and Aggregations

Attribute	Data Type	Initial Value	Publish / Persistence	Client Access	Description
sMEPneuStateA	MPS_ORB_REGCNTL_STATE				
sMEPneuStateB	MPS_ORB_REGCNTL_STATE				

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.2.3.1.4. Operations Index

Operation	Description
activateAReg()	Activates SSME A leg pneumatics for either Engine 1, 2, or 3.
secureAReg()	Secures SSME A leg pneumatics for either Engine 1, 2, or 3.
activateBReg()	Activates SSME B leg pneumatics for either Engine 1,2, or 3.
secureBReg()	Activates SSME B leg pneumatics for either Engine 1,2, or 3.
activateAandBReg()	Activates A and B leg pneumatics for either Engfine 1,2, or 3.
determineSMEPneuStateA()	Determines state of SSME pneumatics based on reg isolation power indications and regulated outlet pressure for each SSME. Possible states are: Active Secured Locked-up Error
getSMEPneuStateA()	Returns enumerated state of SSME pneumatics.
determineSMEPneuStateB()	Determines state of SSME pneumatics based on reg isolation power indications and regulated outlet pressure for each SSME. Possible states are: Active Secured Locked-up Error
getSMEPneuStateB()	Returns enumerated state of SSME pneumatics.

3.2.3.1.5. Associations

Association	Type	Class	Description
MPS_RCL_E1_Reg_Press		Prereq Control Logic	
MPS_RCL_E1_Reg_Press		Prereq Control Logic	
MPS_RCL_E3_Reg_Press		Prereq Control Logic	
MPS_LV1		MPS_SolValve_Orb_SMERegIso A	
MPS_LV2		MPS_SolValve_Orb_SMERegIso B	
MPS_LV3		MPS_SolValve_Orb_SMERegIso A	

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

MPS_LV4		MPS_SolValve_Orb_SMERegIso B	
MPS_LV5		MPS_SolValve_Orb_SMERegIso A	
MPS_LV6		MPS_SolValve_Orb_SMERegIso B	
MPS_MT10		MPS_ORB_Pressure_Sensor	E1A
MPS_MT74		MPS_ORB_Pressure_Sensor	E1B
MPS_MT23		MPS_ORB_Pressure_Sensor	E2A
MPS_MT75		MPS_ORB_Pressure_Sensor	E2B
MPS_MT36		MPS_ORB_Pressure_Sensor	E3A
MPS_MT76		MPS_ORB_Pressure_Sensor	E3B

3.2.3.2. State Description

3.2.3.2.1. State Diagram

<insert state diagram here>

3.2.3.2.2. State Table

3.2.3.3. Operations

3.2.3.3.1. activateAReg()

3.2.3.3.1.1. Descriptive Text

This method will perform activation of SSME 750 psi pneumatics on the 'A' leg pneumatic panel. A check of the reg outlet pressure transducer is performed to verify reg outlet pressure is within OMRS limits.

3.2.3.3.1.2. Sequence

begin sequence;

//open reg isolation valve//

//inhibit RCL on applicable E(#) leg//

MPS_RCL_E#_Reg_Press::inhibit()

MPS_Valve_ORB_SMERegIsoA::open()

wait 2 sconds

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

//activate RCL//

MPS_RCL_E#_Reg_Press::activate()

Verify engine outlet pressure 740< MPS_ORB_Pressure_Sensor <784 psia
else print message to page A "regulated outlet pressure is not within OMRS limits of 740-784 psia"
and print message to page B " Cycle complete with errors, see page A for details"

end sequence;

3.2.3.3.2. secureAReg()

3.2.3.3.2.1. Descriptive text

This method will perform securing (closure of reg iso valve) of SSME 750 psi pneumatics on the 'A' leg pneumatic panel. A check of the reg outlet pressure transducer is performed to verify reg outlet pressure is within OMRS limits.

3.2.3.3.2.2. Sequence

begin sequence;

//close reg isolation valve//

MPS_Valve_ORB_SMERegIsoA::close()

end sequence;

3.2.3.3.3. activateBReg()

3.2.3.3.3.1. Descriptive Text

This method will perform activation of SSME 750 psi pneumatics on the 'B' leg pneumatic panel.

3.2.3.3.3.2. Sequence

begin sequence;

//open reg isolation valve//

//inhibit RCL on applicable E(#) leg//

MPS_RCL_E#_Reg_Press::inhibit()

MPS_Valve_ORB_SMERegIsoB::open()

wait 2 seconds

MPS_RCL_E#_Reg_Press::activate()

Verify engine outlet pressure 740< MPS_ORB_Pressure_Sensor <784 psia
else print message to page A "regulated outlet pressure is not within OMRS limits of 740-784 psia"
and print message to page B " Cycle complete with errors, see page A for details"

end sequence;

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.2.3.3.4. secureBReg()

3.2.3.3.4.1. Descriptive text

This method will perform securing (closure of reg iso valve) of SSME 750 psi pneumatics on the 'B' leg pneumatic panel.

3.2.3.3.4.2. Sequence

begin sequence;

//close reg isolation valve//

MPS_Valve_ORB_SMERegIsoB::close()

end sequence;

3.2.3.3.5. activateAandBReg()

3.2.3.3.5.1. Descriptive Text

This method will perform activation of SSME 750 psi pneumatics on both the 'A' and the 'B' leg pneumatic panels.

3.2.3.3.5.2. Sequence

begin sequence;

//open reg isolation valves//
//inhibit RCL on applicable SSME//

MPS_RCL_E#_Reg_Press::inhibit()

//invokation of both methods is to be done simultaneously (as practicle).
MPS_Valve_ORB_SMERegIsoA::open()
MPS_Valve_ORB_SMERegIsoB::open()

wait 2 seconds;

MPS_RCL_E#_Reg_Press::activate()

Verify regulated A outlet pressure 740< MPS_ORB_Pressure_Sensor <784 psia
else print message to page A "regulated outlet pressure is not within OMRS limits of 740-784 psia"
and print message to page B " Cycle complete with errors, see page A for details"

Verify regulated B outlet pressure 740< MPS_ORB_Pressure_Sensor <784 psia
else print message to page A "regulated outlet pressure is not within OMRS limits of 740-784 psia"
and print message to page B " Cycle complete with errors, see page A for details"
end sequence;

3.2.3.3.6. determineSMEPneuStateA()

case;

MPS_Valve_ORB_SMERegIsoA.cmdState == DEENERGIZED
MPS_ORB_Pressure_Sensor.isAmbient == TRUE
SMEPneuStateA : SECURED

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

MPS_Valve_ORB_SMERegIsoA.cmdState == DEENERGIZED
MPS_ORB_Pressure_Sensor.isAmbient == FALSE
SMEPneuStateA : LOCKED_UP

MPS_Valve_ORB_SMERegIsoA.cmdState == ENERGIZED
MPS_ORB_Pressure_Sensor.isAmbient == FALSE
SMEPneuStateA : ACTIVE

MPS_Valve_ORB_SMERegIsoA.cmdState == ENERGIZED
MPS_ORB_Pressure_Sensor.isAmbient == TRUE
SMEPneuStateA : ERROR

Default Case : SMEPneuStateA : ERROR

end case;

3.2.3.3.7. getSMEPneuStateA()

3.2.3.3.7.1. Descriptive Text

This method will return the Enumerated State of the A leg SSME Pneumatic system.

3.2.3.3.7.2. Sequence

```
getSMEPneuStateA()
{
    return SMEPneuStateA;
}
```

3.2.3.3.8. determineSMEPneuStateB()

case;

MPS_Valve_ORB_SMERegIsoB.cmdState == DEENERGIZED
MPS_ORB_Pressure_Sensor.isAmbient == TRUE
SMEPneuStateB : SECURED

MPS_Valve_ORB_SMERegIsoB.cmdState == DEENERGIZED
MPS_ORB_Pressure_Sensor.isAmbient == FALSE
SMEPneuStateB : LOCKED_UP

MPS_Valve_ORB_SMERegIsoB.cmdState == ENERGIZED
MPS_ORB_Pressure_Sensor.isAmbient == FALSE
SMEPneuStateB : ACTIVE

MPS_Valve_ORB_SMERegIsoB.cmdState == ENERGIZED
MPS_ORB_Pressure_Sensor.isAmbient == TRUE
SMEPneuStateB : ERROR

Default Case : SMEPneuStateB : ERROR

end case;

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.2.3.4. User Interfaces

3.2.3.5. Safety Requirements

3.2.4. MPS_ORB_17inch_Disconnect_Assembly

3.2.4.1. Description

The MPS_ORB_17inch_Disconnect_Assembly class consists of the MPS_Valve_ORB_17inch_Disc and MPS_Valve_ORB_Latch classes. The Assembly class performs the checks required to ensure the latch is configured for disconnect cycling. The Assembly class also determines the state of the disconnect for launch support displays.

3.2.4.1.1. Instance Index

Instance	Display Users	Other Users
MPS_PD1_ASSEMBLY	MPL	
MPS_PD2_ASSEMBLY	MPS	

3.2.4.1.2. Class Diagram

<insert class diagram here>

3.2.4.1.3. Attributes and Aggregations

Attribute	Data Type	Initial Value	Publish / Persistence	Client Access	Description
discAssemblyState	MPS_ORB_DISC_ASSEMBLY_ STATE ENUM				

3.2.4.1.4. Operations Index

Operation	Description
determineDiscAssemblyState()	
getDiscAssemblyState()	
emergencyClose()//VAB/PAD//	//VAB/PAD method for emergency ORB/ET isolation.//

3.2.4.1.5. Associations

Association	Type	Class	Description

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.2.4.2. State Description

3.2.4.2.1. State Diagram

<insert state diagram here>

3.2.4.2.2. State Table

3.2.4.3. Operations

3.2.4.3.1. determineDiscAssemblyState()

3.2.4.3.1.1. Description

This method determines the over-all state of the 17" disconnect assembly based upon the 17" disconnect valve and the latch. Possible values are: OPEN_LATCHED, CLOSED, TRANSITION, and ERROR. This is derived from the disconnect having a VALVE_STATE of: OPEN, CLOSED, OPEN_ERROR, CLOSED_ERROR, and TRANSITION and the latch having a LATCH_STATE of: LOCKED, LOCKED_ERROR, UNLOCKED, UNLOCKED_ERROR, and TRANSITION.

3.2.4.3.1.2. Sequence

determineDiscAssemblyState()

Case:
    VALVE_STATE : OPEN
    LATCH_STATE : LOCKED
        AssemblyState = OPEN_LATCHED

    VALVE_STATE : CLOSED
    LATCH_STATE : UNLOCKED
        AssemblyState = CLOSED

    VALVE_STATE : TRANSITION
        AssemblyState = TRANSITION

    Default Case = ERROR

3.2.4.3.2. getDiscAssemblyState()

3.2.4.3.2.1. Description

This method returns the enumerated value of the disconnect assembly over-all state.

3.2.4.3.2.2. Sequence

```
getDiscAssemblyState()
{
    return discAssemblyState;
}
```

3.2.4.3.3. emergencyClose()//VAB/PAD method//

3.2.4.3.3.1. Description

This method is used for emergency isolation of the Orbiter from the ET.

3.2.4.3.3.2. Sequence

unlock()

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

close()

3.2.4.4. User Interfaces

3.2.4.5. Safety Requirements

3.2.5. MPS_ORB_ETOI

3.2.5.1. Description

MPS_ET_OI_Control will encompass ET OI power up/down and checkout, GO2/GH2 flow control valve (FCV) control and checkout, LO2/LH2 ullage pressure control and checkout as well as point sensor verification. The following OMRSD's are associated with and will be included in this section: S00FAO.160, S00FF0.204, S00000.064, V41AC0.010/011/020/030/060, V41AD0.010/015/020/025/030/035, V41AH0.020/025/030/035/040/045//050/, V41AI0.050/055/060/065/070/075/080

3.2.5.1.1. Instance Index

Instance	Display User	Other Users
MPS_ETOICNTL	MPS	

3.2.5.1.2. Class Diagram

<insert class diagram here>

3.2.5.1.3. Attributes and Aggregations

Attribute	Data Type	Initial Value	Publish / Persistence	Client Access	Description
etOiPwrCmd1	DS_ON_OFF			R,W	
etOiPwrCmd2	DS_ON_OFF			R,W	
etOiPwrCmd3	DS_ON_OFF			R,W	
etOiRpcB	DM_ON_OFF			R	
etOiRpcC	DM_ON_OFF			R	
e1FcvHiPressCheckoutCmd	DS_ON_OFF			R,W	
e2FcvHiPressCheckoutCmd	DS_ON_OFF			R,W	
e3FcvHiPressCheckoutCmd	DS_ON_OFF			R,W	
fcvAllLoPressCheckoutCmd	DS_ON_OFF			R,W	

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

fcvAllCheckoutArmCmd	DS_ON_OFF			R,W	
llsAllOpenCmd	DS_ON_OFF			R,W	
llsAllDryCmd	DS_ON_OFF			R,W	
llsAllWetCmd	DS_ON_OFF			R,W	
eco1SimWetCmd	DS_ON_OFF			R,W	
eco2SimWetCmd	DS_ON_OFF			R,W	
eco3SimWetCmd	DS_ON_OFF			R,W	
eco4SimWetCmd	DS_ON_OFF			R,W	
ecosSimAllOpenCmd	DS_ON_OFF			R,W	
ecosSimAllDryCmd	DS_ON_OFF			R,W	
e1Lh2UllageDucrSwapCmd	DS_ON_OFF			R,W	
e2Lh2UllageDucrSwapCmd	DS_ON_OFF			R,W	
e3Lh2UllageDucrSwapCmd	DS_ON_OFF			R,W	
e1Lo2UllageDucrSwapCmd	DS_ON_OFF			R,W	
e2Lo2UllageDucrSwapCmd	DS_ON_OFF			R,W	
e3Lo2UllageDucrSwapCmd	DS_ON_OFF			R,W	
e1UllageDucerSwapPseudo	DM_ON_OFF			R	
e2UllageDucerSwapPseudo	DM_ON_OFF			R	
e3UllageDucerSwapPseudo	DM_ON_OFF			R	
fcvInhibitState	ENUM			R	
etOiState	ENUM			R	

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

lh2EcosState	ENUM			R	
Lox5LiqLevelInd	DM_ON_OFF				Lite FD
Lox98LiqLevelInd1	DM_ON_OFF				Lite FD
Lox98LiqLevelInd2	DM_ON_OFF				Lite FD
Lox100LiqLevelInd 1	DM_ON_OFF				Lite FD
Lox100LiqLevelInd 2	DM_ON_OFF				Lite FD
Minus100LoxLiqLe velInd	DM_ON_OFF				Lite FD
Plus100LoxLiqLeve lInd	DM_ON_OFF				Lite FD
overFillLoxLiqLeve lInd	DM_ON_OFF				Lite FD
Lh5LiqLevelInd	DM_ON_OFF				Lite FD
Lh98LiqLevelInd1	DM_ON_OFF				Lite FD
Lh98LiqLevelInd2	DM_ON_OFF				Lite FD
100%MinusLhLiqL evelInd	DM_ON_OFF				Lite FD
Lh100LiqLevelInd1	DM_ON_OFF				Lite FD
Lh100LiqLevelInd2	DM_ON_OFF				Lite FD
Plus100LhLiqLevel Ind	DM_ON_OFF				Lite FD
overFillLhLiqLevelI nd	DM_ON_OFF				Lite FD
lh2Eco1	DM_WET_DRY				Lite FD
lh2Eco2	DM_WET_DRY				Lite FD
lh2Eco3	DM_WET_DRY				Lite FD
lh2Eco4	DM_WET_DRY				Lite FD
lh2UllagePress1	AM_PSIA				Lite FD
lh2UllagePress2	AM_PSIA				Lite FD

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

lh2UllagePress3	AM_PSIA				Lite FD
lh2UllagePress4	AM_PSIA				Lite FD
lh2UllPressHiFlwSwScn	DM_ON_OFF				Lite FD
lo2UllagePress1	AM_PSIG				Lite FD
lo2UllagePress2	AM_PSIG				Lite FD
lo2UllagePress3	AM_PSIG				Lite FD
lo2EcoLeft1	DM_WET_DRY				Lite FD
lo2EcoLeft2	DM_WET_DRY				Lite FD
lo2EcoRight1	DM_WET_DRY				Lite FD
lo2EcoRight2	DM_WET_DRY				Lite FD

3.2.5.1.4. Methods Index

Operation	**Description**
turnOff	This method sets the ETOI Power buses to Off
turnOn	This method sets the ETOI Power buses to On
setPwrUpConstraints	Sets up Constraints for ETOI Power-up conditions
setPwrDownConstraints	Sets up Constraints for ETOI Power down conditions
getLh2EcosState	This method will return LH2 Engine Cutoff Sensor Enumerated state
getLo2EcosState//Phase3	This method will return LO2 Engine Cutoff Sensor Enumerated state
determineLh2EcosState	This method will return state of LH2 Engine Cutoff Sensor by voting on unbypassed ECO sensor indicators and will return either WET, DRY or TRANSITION
determineLo2EcosState//Phase3	This method will return state of LO2 Engine Cutoff Sensor by voting on unbypassed ECO sensor indicators and will return either WET, DRY or TRANSITION
getState	This method will return ETOI power Enumerated state
determineState	This method will return state of ETOI power by voting on unbypassed RPC's B & C and will return either ON OFF

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

	ERROR
inhibitConstraints	This method will inhibit Constraints for RPC's B & C during ETOI power up/down operations
activateConstraints	This method will activate Constraints for RPC's B & C after power up/down operations
turnOnFcvInhibit//Phase3	
turnOffFcvInhibit//Phase3	
determineFcvInhibitState//Phase3	
getFcvInhibitState//Phase3	This method will return the enumerated state for readFcvInhibit
signalConditionPwrCntlVerif()//Phase 3	This method performs signal conditioner power bus control verification per OMRSD's S00000.064 and S00FA0.160 (partial)
ptSensTestDuringReplenish()//Phase3	This method performs Pt sensor checkout during replenish, per OMRSD S00FF0.204
etOiSigCondPwrCtrlVerif()	This method performs MPS signal conditioner power control verification per OMRSD V41AI0.080.
ullagePressCtrlVerifWithCkOutCmds( )	This method performs engines 1, 2 & 3 FCV ullage pressure control verification using checkout commands per OMRSD's V41AD0.010/.020/.030
llsOpenDryWetSimMated()//Phase3	This method performs liquid level sensor verification in the mated configuration, per OMRSD's V41AH0.020/.030/.040 & V41AI0.050/.060/.075
llsOpenDryWetSimUnmated()	This method performs liquid level sensor verification in the mated configuration, per OMRSD's V41AH0.025/.035/.045 & V41AI0.055/.065/.070
ullagePressCntlEtSimGox()	This method performs FCV ullage pressure control verification of the GO2 system for the active and non active (orificed) FCV configuration
ullagePressCntlEtSimGh2()	This method performs FCV ullage pressure control verification of the GH2 system per OMRSD V41AC0.020
gh2UllagePressRedundancyWithEtSim()	This method performs GH2 ullage pressure transducer redundancy verificaton per OMRSD V41AC0.060
go2UllagePressRedundancyWithEtSim()	This method performs GO2 ullage pressure transducer signal conditioner redundancy verificaton
llsVerifEtSim	This method performs Et liquid level sensor verification with the ET simulator per OMRSD V41AH0.050.
swapUllageDucer1	This method will swap in the spare GH2 ullage pressure transducer

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

	number 4 into the engine number 1 location and set its associated pseudo accordingly.
swapUllageDucer2	This method will swap in the spare GH2 ullage pressure transducer number 4 into the engine number 2 location and set its associated pseudo accordingly.
swapUllageDucer3	This method will swap in the spare GH2 ullage pressure transducer number 4 into the engine number 3 location and set its associated pseudo accordingly.
verifyCommandsOff()	This method will return a True/False based on results of ByteTest4 for command issuance to solenoid.
verifyTminus6BT4()//Phase3	This method verifies command is ON at T-6 hours Byte Test 4. This is a VAB/PAD method.
verifyTminus3BT4()//Phase3	This method verifies command is ON at T-3 hours Byte Test 4. This is a VAB/PAD method.
issueE1Gh2FcvCmds()	This method will set the fcvAllCheckoutArmCmd and the e1FcvHiPressCheckoutCmd to ON to allow energization of a FCV via the MPS_VALVE_ORB_GH2_FCV class
issueE2Gh2FcvCmds()	This method will set the fcvAllCheckoutArmCmd and the e2FcvHiPressCheckoutCmd to ON to allow energization of a FCV via the MPS_VALVE_ORB_GH2_FCV class
issueE3Gh2FcvCmds()	This method will set the fcvAllCheckoutArmCmd and the e3FcvHiPressCheckoutCmd to ON to allow energization of a FCV via the MPS_VALVE_ORB_GH2_FCV class
removeE1Gh2FcvCmds()	This method will set the fcvAllCheckoutArmCmd and the e1FcvHiPressCheckoutCmd to OFF to allow deenergization of a FCV via the MPS_VALVE_ORB_GH2_FCV class
removeE2Gh2FcvCmds()	This method will set the fcvAllCheckoutArmCmd and the e2FcvHiPressCheckoutCmd to OFF to allow deenergization of a FCV via the MPS_VALVE_ORB_GH2_FCV class
removeE3Gh2FcvCmds()	This method will set the fcvAllCheckoutArmCmd and the e3FcvHiPressCheckoutCmd to OFF to allow deenergization of a FCV via the MPS_VALVE_ORB_GH2_FCV class

3.2.5.1.5. Associations

Association	Type	Class	Description
MPS_E1FCV		MPS_SolValve_ORB_FCV	Association required to manipulate GH2/GO2 FCV's
MPS_E2FCV		MPS_SolValve_ORB	Association required to manipulate GH2/GO2

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

		_FCV	FCV's
MPS_E3FCV		MPS_SolValve_ORB _FCV	Association required to manipulate GH2/GO2 FCV's

3.2.5.2. State Description

3.2.5.2.1. State Diagram

<insert state diagram here>

3.2.5.2.2. State Table

3.2.5.3. Operations

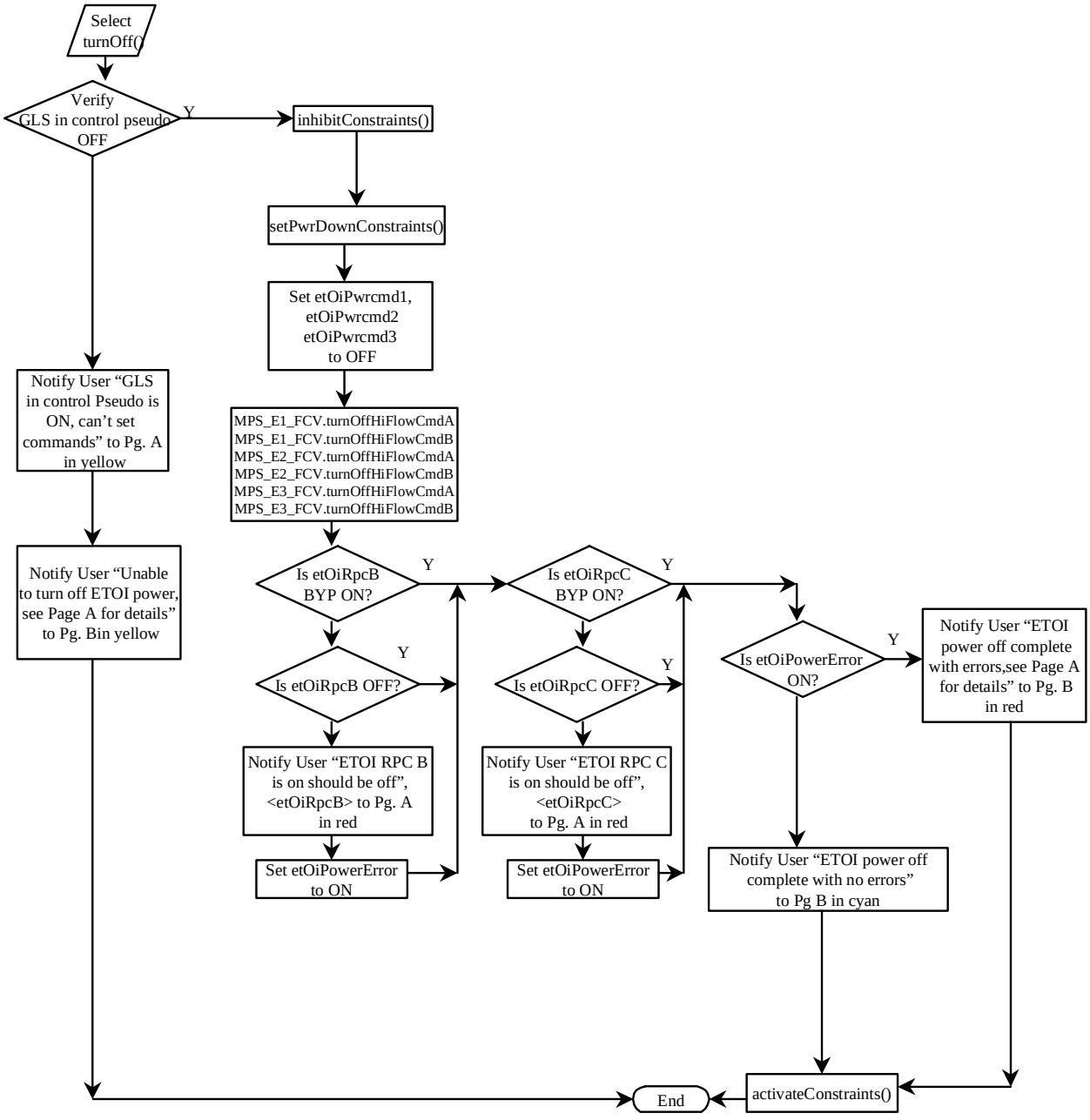
3.2.5.3.1. turnOff()

3.2.5.3.1.1. Descriptive Text

This method performs the necessary functions to terminate ETOI Power, which includes removing the GH2 FCV High flow commands. This method not applicable/available when cryo flag is on.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.2.5.3.1.2. Sequence



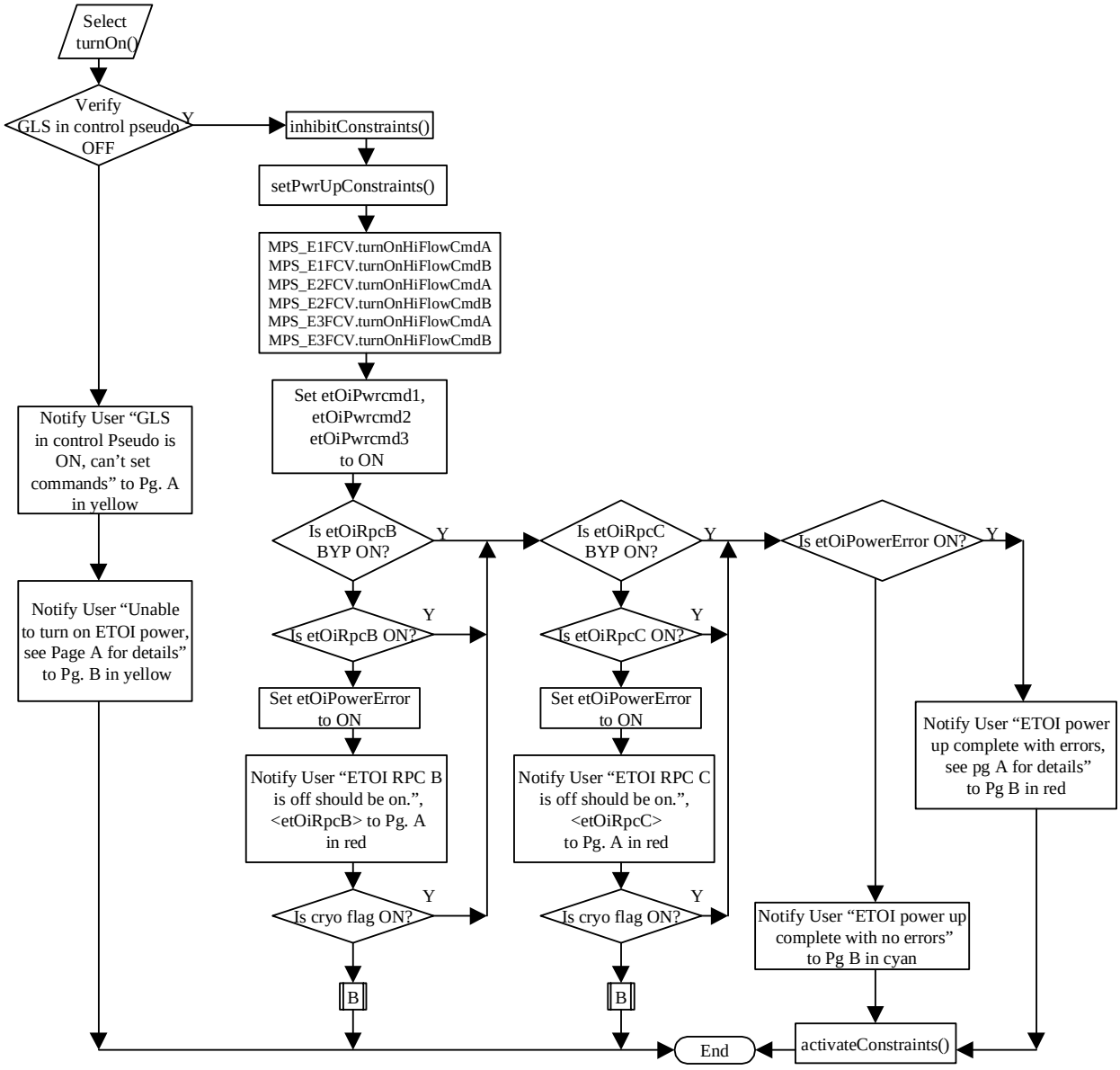
3.2.5.3.2. turnOn()

3.2.5.3.2.1. Descriptive Text

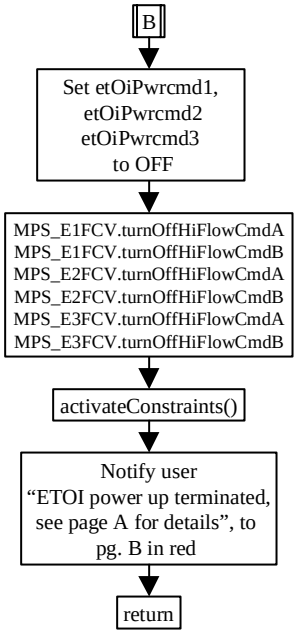
This method performs the necessary functions to activate ETOI Power, which includes setting the GH2 FCV High flow commands ON. This method not applicable/available when cryo flag is on.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.2.5.3.2.2. Sequence



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



3.2.5.3.3. setPwrUpConstraints()

3.2.5.3.3.1. Descriptive Text

This method changes the constraint criteria of the measurements associated with ETOI Pwr Up to those expected.

3.2.5.3.3.2. Sequence

```
setPwrUpConstraints()
{
        CHANGE <etOiRpcB> CONSTRAINT CRITERIA TO ON;

        CHANGE <etOiRpcC>  CONSTRAINT CRITERIA TO ON;

}
```

3.2.5.3.4. setPwrDownConstraints()

3.2.5.3.4.1. Descriptive Text

This method changes the constraint criteria of the measurements associated with ETOI Pwr Down to those expected.

3.2.5.3.4.2. Sequence

```
setPwrDownConstraints()
{
        CHANGE <etOiRpcB> CONSTRAINT CRITERIA TO OFF;

        CHANGE <etOiRpcC>  CONSTRAINT CRITERIA TO OFF;

}
```

3.2.5.3.5. getLh2EcosState()

3.2.5.3.5.1. Descriptive Text

This method will return the overall state of the LH2 engine cut-off sensors.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.2.5.3.5.2. Sequence

```
getLh2EcosState()
{
        return Lh2EcosState;
}
```

3.2.5.3.6. getLo2EcosState()//Phase3

3.2.5.3.7. determineLh2EcosState()

3.2.5.3.7.1. Descriptive Text

This method is used to determine the state of the LH2 liquid level engine cut-off sensors based on majority voting. This method will be used to support PCL voting.

3.2.5.3.7.2. Effectivity

Operates on change data.

3.2.5.3.7.3. Sequence

Indicator State				
lh2Eco1	**lh2Eco2**	**lh2Eco3**	**lh2Eco4**	**State**
DRY	DRY	DRY	DRY	DRY
WET	DRY	DRY	DRY	DRY
DRY	WET	DRY	DRY	DRY
DRY	DRY	WET	DRY	DRY
DRY	DRY	DRY	WET	DRY
WET	WET	WET	WET	WET
DRY	WET	WET	WET	WET
WET	DRY	WET	WET	WET
WET	WET	DRY	WET	WET
WET	WET	WET	DRY	WET
WET	WET	DRY	DRY	TRANSITION
WET	DRY	WET	DRY	TRANSITION
WET	DRY	DRY	WET	TRANSITION
DRY	WET	WET	DRY	TRANSITION
DRY	WET	DRY	WET	TRANSITION
DRY	DRY	WET	WET	TRANSITION

3.2.5.3.8. determineLo2EcosState()//Phase3

3.2.5.3.9. getState()

3.2.5.3.9.1. Descriptive Text

This method will return the overall Enumerated State of ETOI Power.

3.2.5.3.9.2. Sequence

```
getState()
{
        return state;
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.2.5.3.10. determineState()

3.2.5.3.10.1. Descriptive Text

This method is used to determine the state of ETOI power based on its open RPC's (B&C) and associated indicator bypasses. The return value will be an enumerated type with possible values of ON, OFF and ERROR.

3.2.5.3.10.2. Effectivity

Operates on change data.

3.2.5.3.10.3. Sequence

RPC State				
RPC B	RPC B Byp	RPC C	RPC C Byp	State
OFF	OFF	OFF	OFF	OFF
OFF	OFF	--	ON	OFF
--	ON	OFF	OFF	OFF
ON	OFF	ON	OFF	ON
ON	OFF	--	ON	ON
--	ON	ON	OFF	ON
--	ON	--	ON	ERROR
ON	OFF	OFF	OFF	ERROR
OFF	OFF	ON	OFF	ERROR

3.2.5.3.11. activateConstraints()

3.2.5.3.11.1. Descriptive Text

This method will activate constraint monitoring of all unbypassed measurement of this valve.

3.2.5.3.11.2. Sequence

```
activateConstraints()
        {
        if(<etOiRpcB>.Byp == OFF)
                {
                activate constraint monitoring for <etOiRpcB>
                }
        if(<etOiRpcC>.Byp == OFF)
                {
                activate constraint monitoring for <etOiRpcC>
                }

        }
```

3.2.5.3.12. inhibitConstraints()

3.2.5.3.12.1. Descriptive Text

This method will inhibit constraint monitoring of all measurements related to this valve.

3.2.5.3.12.2. Sequence

```
inhibitConstraints()
        {
        inhibit constraint monitoring of the following:
        <etOiRpcB>
        <etOiRpcC>
        }
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.2.5.3.13. signalConditionPwrCntlVerif()//Phase3

3.2.5.3.13.1. Descriptive Text

This method performs signal conditioner power bus control verification per OMRSD's S00000.064 and S00FA0.160

3.2.5.3.14. ptSensTestDuringReplenish()//Phase3

3.2.5.3.14.1. Descriptive Text

This method performs Pt sensor checkout during replenish, per OMRSD S00FF0.204

3.2.5.3.14.2. Sequence

Inhibit emons for etOiRpcB and etOiRpcC

Set llsAllDryCmd on, llsAllOpenCmd off & llsAllWetCmd off

Delay 3 seconds

Verify 5%Lh2LiqLevelInd wet, else notify console operator, include the test conditions in message

Verify 98%Lh2LiqLevelInd1 wet or flashing dry within 10 seconds, else notify console operator, include the test conditions in message

Verify 98%Lh2LiqLevelInd2 wet or flashing dry within 10 seconds, else notify console operator, include the test conditions in message

Verify 5%LoxLiqLevelInd wet, else notify console operator, include the test conditions in message

Verify 98%LoxLiqLevelInd1 wet or flashing dry within 10 seconds, else notify console operator, include the test conditions in message

Verify 98%LoxLiqLevelInd2 wet or flashing dry within 10 seconds, else notify console operator, include the test conditions in message

Set llsAllDryCmd off, llsAllOpenCmd off & llsAllWetCmd on

Delay 3 seconds

Verify 5%Lh2LiqLevelInd wet, else notify console operator, include the test conditions in message

Verify 98%Lh2LiqLevelInd1 wet or flashing dry within 10 seconds, else notify console operator, include the test conditions in message

Verify 98%Lh2LiqLevelInd2 wet or flashing dry within 10 seconds, else notify console operator, include the test conditions in message

Verify 5%LoxLiqLevelInd wet, else notify console operator, include the test conditions in message

Verify 98%LoxLiqLevelInd1 wet or flashing dry within 10 seconds, else notify console operator, include the test conditions in message

Verify 98%LoxLiqLevelInd2 wet or flashing dry within 10 seconds, else notify console operator, include the test conditions in message

Set llsAllDryCmd on, llsAllOpenCmd off & llsAllWetCmd on

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

Delay 3 seconds

Verify 5%Lh2LiqLevelInd dry, else notify console operator, include the test conditions in message

Verify 98%Lh2LiqLevelInd1 dry, else notify console operator, include the test conditions in message

Verify 98%Lh2LiqLevelInd2 dry, else notify console operator, include the test conditions in message

Verify 100%MinusLh2LiqLevelInd dry, else notify console operator, include the test conditions in message

Verify 100%Lh2LiqLevelInd1 dry, else notify console operator, include the test conditions in message

Verify 100%Lh2LiqLevelInd2 dry, else notify console operator, include the test conditions in message

Verify 100%PlusLh2LiqLevelInd dry, else notify console operator, include the test conditions in message

Verify overfillLh2LiqLevelInd dry, else notify console operator, include the test conditions in message

Verify 5%LoxLiqLevelInd dry, else notify console operator, include the test conditions in message

Verify 98%LoxLiqLevelInd1 dry, else notify console operator, include the test conditions in message

Verify 98%LoxLiqLevelInd2 dry, else notify console operator, include the test conditions in message

Verify 100%MinusLoxLiqLevelInd dry, else notify console operator, include the test conditions in message

Verify 100%LoxLiqLevelInd1 dry, else notify console operator, include the test conditions in message

Verify 100%LoxLiqLevelInd2 dry, else notify console operator, include the test conditions in message

Verify 100%PlusLoxLiqLevelInd dry, else notify console operator, include the test conditions in message

Verify overfillLoxLiqLevelInd1 wet or flashing dry within 10 seconds, else notify console operator, include the test conditions in message

Set llsAllDryCmd off, llsAllOpenCmd off & llsAllWetCmd off

Delay 3 seconds

Verify 5%Lh2LiqLevelInd wet, else notify console operator, include the test conditions in message

Verify 98%Lh2LiqLevelInd1 wet or flashing dry within 10 seconds, else notify console operator, include the test conditions in message

Verify 98%Lh2LiqLevelInd2 wet or flashing dry within 10 seconds, else notify console operator, include the test conditions in message

Verify 5%LoxLiqLevelInd wet, else notify console operator, include the test conditions in message

Verify 98%LoxLiqLevelInd1 wet or flashing dry within 10 seconds, else notify console operator, include the test conditions in message

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

Verify 98%LoxLiqLevelInd2 wet or flashing dry within 10 seconds, else notify console operator, include the test conditions in message

Set ecosSimAllDryCmd on, eco1SimWetCmd off, eco2SimWetCmd off, eco3SimWetCmd off & eco4SimWetCmd off

Delay 3 seconds

Verify lh2EcoInd1 wet, else notify console operator, include the test conditions in message

Verify lh2EcoInd2 wet, else notify console operator, include the test conditions in message

Verify lh2EcoInd3 wet, else notify console operator, include the test conditions in message

Verify lh2EcoInd4 wet, else notify console operator, include the test conditions in message

Verify lo2EcoInd1 wet, else notify console operator, include the test conditions in message

Verify lo2EcoInd2 wet, else notify console operator, include the test conditions in message

Verify lo2EcoInd3 wet, else notify console operator, include the test conditions in message

Verify lo2EcoInd4 wet, else notify console operator, include the test conditions in message

Set ecosSimAllDryCmd off, eco1SimWetCmd on, eco2SimWetCmd on, eco3SimWetCmd on & eco4SimWetCmd on

Delay 3 seconds

Verify lh2EcoInd1 wet, else notify console operator, include the test conditions in message

Verify lh2EcoInd2 wet, else notify console operator, include the test conditions in message

Verify lh2EcoInd3 wet, else notify console operator, include the test conditions in message

Verify lh2EcoInd4 wet, else notify console operator, include the test conditions in message

Verify lo2EcoInd1 wet, else notify console operator, include the test conditions in message

Verify lo2EcoInd2 wet, else notify console operator, include the test conditions in message

Verify lo2EcoInd3 wet, else notify console operator, include the test conditions in message

Verify lo2EcoInd4 wet, else notify console operator, include the test conditions in message

Set ecosSimAllDryCmd on, eco1SimWetCmd on, eco2SimWetCmd off, eco3SimWetCmd off & eco4SimWetCmd off

Delay 3 seconds

Verify lh2EcoInd1 dry, else notify console operator, include the test conditions in message

Verify lo2EcoInd1 dry, else notify console operator, include the test conditions in message

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

Set ecosSimAllDryCmd on, eco1SimWetCmd off, eco2SimWetCmd off, eco3SimWetCmd off & eco4SimWetCmd off

Delay 3 seconds

Verify lh2EcoInd1 wet, else notify console operator, include the test conditions in message

Verify lo2EcoInd1 wet, else notify console operator, include the test conditions in message

Set ecosSimAllDryCmd on, eco1SimWetCmd off, eco2SimWetCmd on, eco3SimWetCmd off & eco4SimWetCmd off

Delay 3 seconds

Verify lh2EcoInd2 dry, else notify console operator, include the test conditions in message

Verify lo2EcoInd2 dry, else notify console operator, include the test conditions in message

Set ecosSimAllDryCmd on, eco1SimWetCmd off, eco2SimWetCmd off, eco3SimWetCmd off & eco4SimWetCmd off

Delay 3 seconds

Verify lh2EcoInd2 wet, else notify console operator, include the test conditions in message

Verify lo2EcoInd2 wet, else notify console operator, include the test conditions in message

Set ecosSimAllDryCmd on, eco1SimWetCmd off, eco2SimWetCmd off, eco3SimWetCmd on & eco4SimWetCmd off

Delay 3 seconds

Verify lh2EcoInd3 dry, else notify console operator, include the test conditions in message

Verify lo2EcoInd3 dry, else notify console operator, include the test conditions in message

Set ecosSimAllDryCmd on, eco1SimWetCmd off, eco2SimWetCmd off, eco3SimWetCmd off & eco4SimWetCmd off

Delay 3 seconds

Verify lh2EcoInd3 wet, else notify console operator, include the test conditions in message

Verify lo2EcoInd3 wet, else notify console operator, include the test conditions in message

Set ecosSimAllDryCmd on, eco1SimWetCmd off, eco2SimWetCmd off, eco3SimWetCmd off & eco4SimWetCmd on

Delay 3 seconds

Verify lh2EcoInd4 dry, else notify console operator, include the test conditions in message

Verify lo2EcoInd4 dry, else notify console operator, include the test conditions in message

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

Set ecosSimAllDryCmd on, eco1SimWetCmd off, eco2SimWetCmd off, eco3SimWetCmd off & eco4SimWetCmd off

Delay 3 seconds

Verify lh2EcoInd4 wet, else notify console operator, include the test conditions in message

Verify lo2EcoInd4 wet, else notify console operator, include the test conditions in message

Set ecosSimAllDryCmd off, eco1SimWetCmd off, eco2SimWetCmd off, eco3SimWetCmd off & eco4SimWetCmd off

Delay 3 seconds

Verify lh2EcoInd1 wet, else notify console operator, include the test conditions in message

Verify lh2EcoInd2 wet, else notify console operator, include the test conditions in message

Verify lh2EcoInd3 wet, else notify console operator, include the test conditions in message

Verify lh2EcoInd4 wet, else notify console operator, include the test conditions in message

Verify lo2EcoInd1 wet, else notify console operator, include the test conditions in message

Verify lo2EcoInd2 wet, else notify console operator, include the test conditions in message

Verify lo2EcoInd3 wet, else notify console operator, include the test conditions in message

Verify lo2EcoInd4 wet, else notify console operator, include the test conditions in message

Record GMT along with message stating “End of ET Point Sensor Checkout During Replenish”. In addition notify operator that any anomalies are listed on page B. In addition, notify operator that “ET pwr has been left on”

Activate emons for etOiRpcB and etOiRpcC

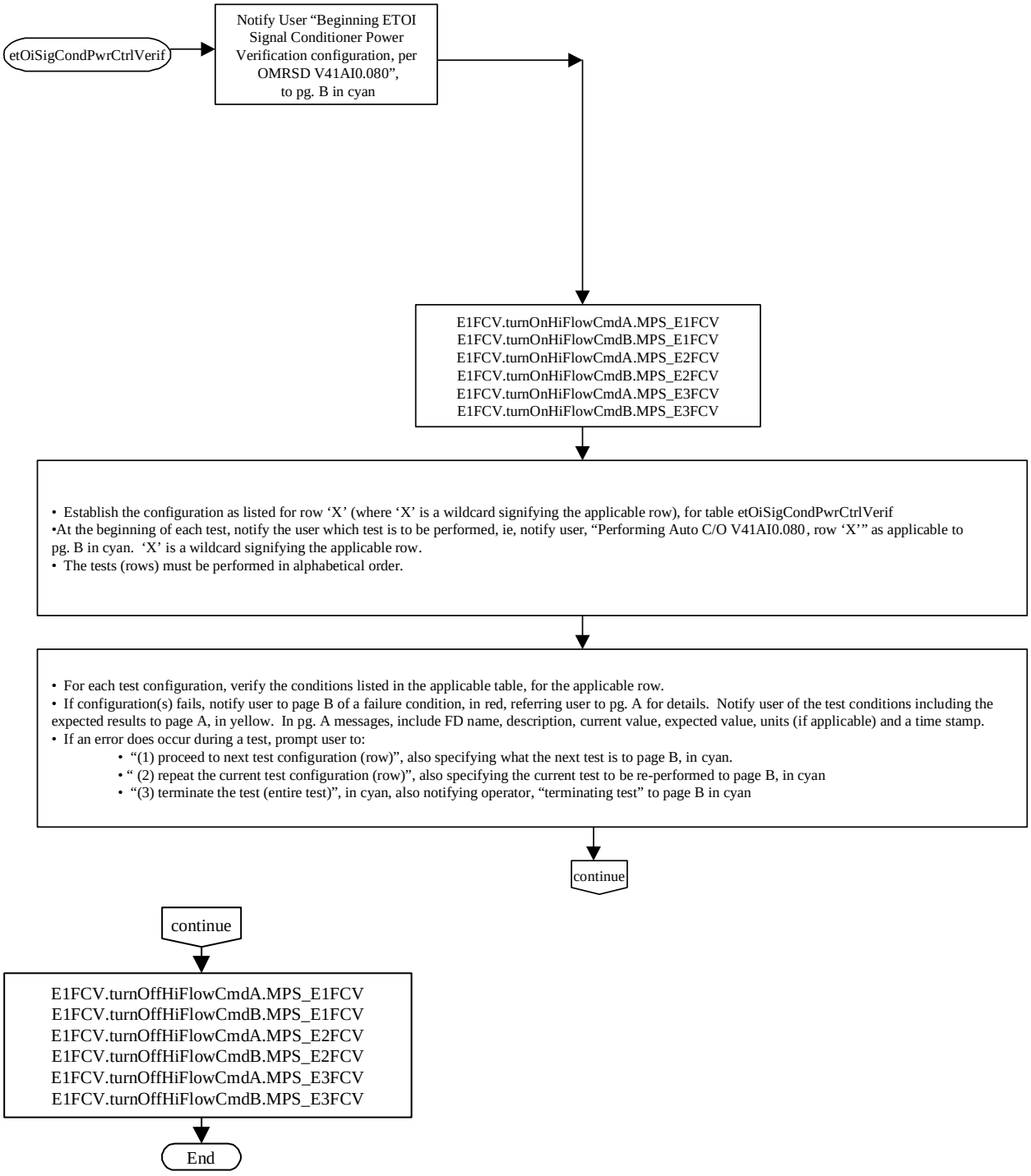
3.2.5.3.15. etOiSigCondPwrCtrlVerif()

3.2.5.3.15.1. Descriptive Text

This method performs ETOI Signal Conditioner Power Verification using checkout commands per OMRSD V41AI0.080

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.2.5.3.15.2. Sequence



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

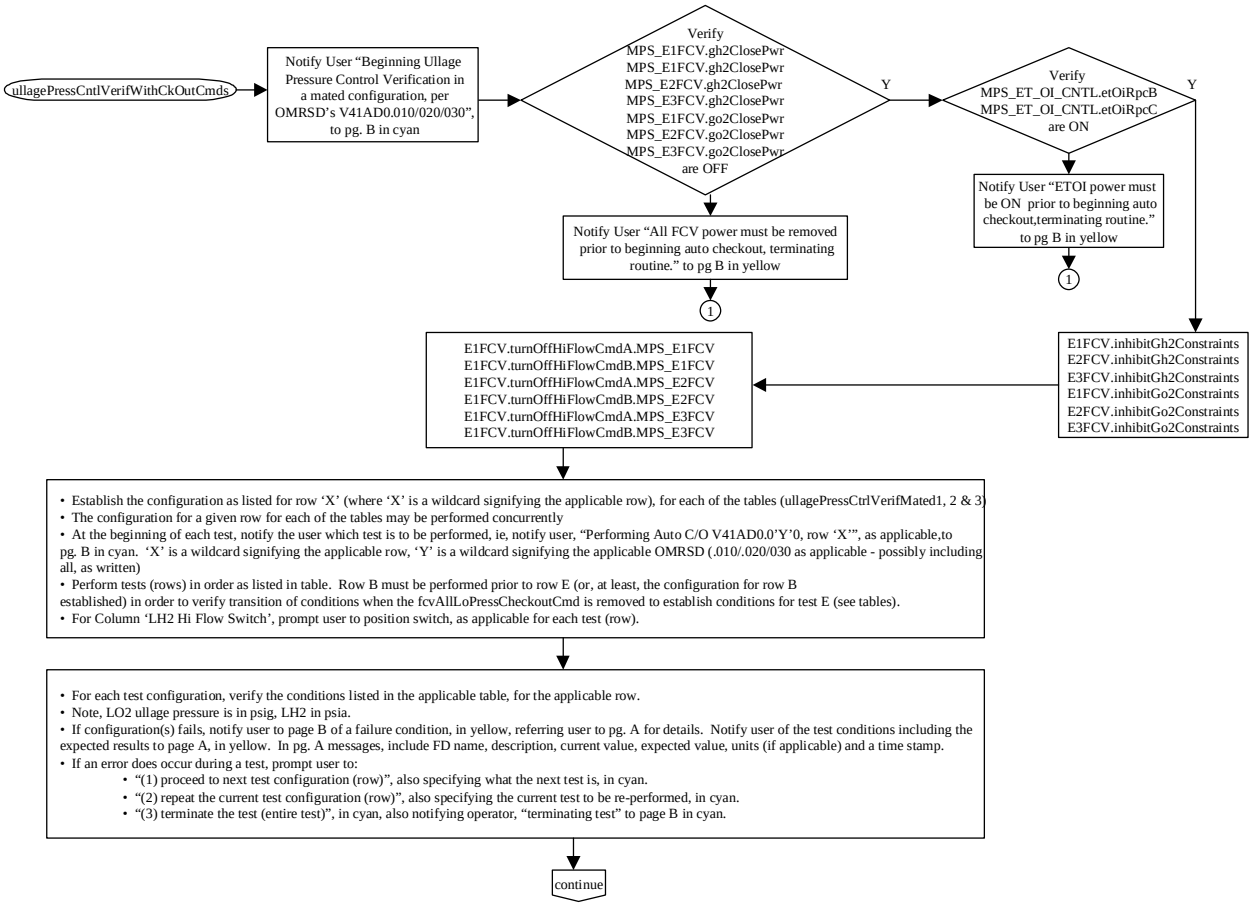
etoiSigCondPwrCtrlVerif, V41AI0.080									
	Test Conditions			Verification Requirements					
	Signal Condition Pwr Cmds			RPC Msmts.		ECO Sensors			
	etOiPwrCmd1, V41K0075X	etOiPwrCmd2, V41K0076X	etOiPwrCmd3, V41K0077X	etOiRpcB, V76X3050E	etOiRpcC, V76X3055E	o2EcoLeft1, V41X1555X	o2EcoLeft2, V41X1556X	o2EcoRight1, V41X1557X	o2EcoRight2, V41X1558X
A	ON	OFF	OFF	OFF	OFF	DRY	WET	WET	WET
B	OFF	ON	OFF	ON	OFF	WET	DRY	DRY	WET
C	OFF	OFF	ON	OFF	ON	WET	WET	DRY	DRY
D	OFF	OFF	OFF	OFF	OFF	WET	WET	WET	WET

3.2.5.3.16. ullagePressCntlVerifWithCkOutCmds()

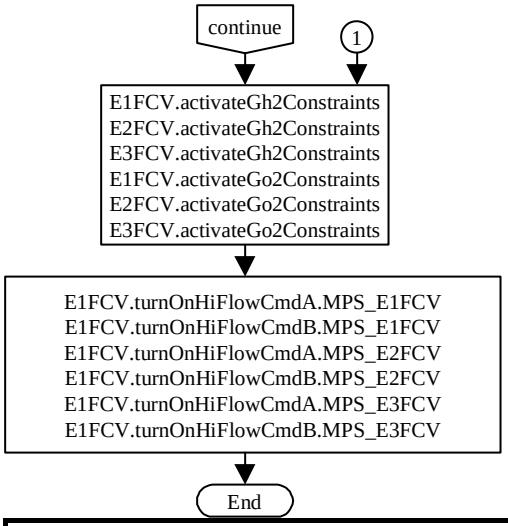
3.2.5.3.16.1. Descriptive Text

This method performs engines 1, 2 & 3 FCV ullage pressure control verification using checkout commands per OMRSD's V41AD0.010/.020/.030

3.2.5.3.16.2. Sequence



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



ullagePressCtrlVerifWithCkOutCmds1											
Test Conditions							Verification Requirements				
Test	E1 Hi Press Cmds		Sim Cmds			LH2 Hi Flow Switch	Hi Flow Sw Scn	E1 Flow Control Valves Close Solenoid		Ullage Pressure	
	hiFlowCmdA, V41K1481N	hiFlowCmdB, V41K1482N	fcvAllLoPressCheckoutCmd, V41K0050N	fcvAllCheckoutArmCmd, V41K0051N	E1FcvHiPressCheckoutCmd, V41K0052N			E1FCV.go2ClPwr, V41X1596t	E1FCV.gh2ClPwr, V41X1661E	lo2UllagePress1 (psig), T41P1750C	lh2UllagePress1 (psia), T41P1700C
A	OFF	OFF	ON	OFF	OFF	AUTO	OFF	OFF	OFF	< 10	< 24.7
C	OFF	OFF	ON	ON	ON	AUTO	OFF	ON	ON	21.6-23.4	33.3-35.7
D	OFF	OFF	OFF	OFF	OFF	AUTO	OFF	OFF	OFF	< 10	< 24.7
B	OFF	OFF	ON	ON	OFF	AUTO	OFF	OFF	OFF	18.6-20.4	30.3-32.7
E	OFF	OFF	OFF	ON	OFF	AUTO	OFF	OFF	OFF	< 10	< 24.7
F	OFF*	OFF*	ON	ON	ON	OPEN	ON	ON	OFF	21.6-23.4	33.3-35.7
H	ON	ON	ON	ON	ON	AUTO	OFF	ON	OFF	21.6-23.5	33.3-35.7
J	ON	OFF	ON	ON	ON	AUTO	OFF	ON	OFF	21.6-23.6	33.3-35.7
K	OFF	ON	ON	ON	ON	AUTO	OFF	ON	OFF	21.6-23.7	33.3-35.7

*BT4 must be performed after positioning switch

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

ullagePressCtrlVerifWithCkOutCmds2											
Test Conditions							Verification Requirements				
Test	E2 Hi Press Cmds		Sim Cmds			LH2 Hi Flow Switch	Hi Flow Sw Scn	E2 Flow Control Valves Close Solenoid		Ullage Pressure	
	hiFlowCmdA, V41K1483N	hiFlowCmdB, V41K1484N	fcvAllLoPressCheckoutCmd, V41K0050N	fcvAllCheckoutArmCmd, V41K0051N	E2FcvHiPressCheckoutCmd, V41K0053N			E2FCV.go2ClPwr, V41X1598E	E2FCV.gh2ClPwr, V41X1662E	lo2UllagePress2 (psig), T41P1751C	lh2UllagePress2 (psia), T41P1701C
A	OFF	OFF	ON	OFF	OFF	AUTO	OFF	OFF	OFF	< 10	< 24.7
C	OFF	OFF	ON	ON	ON	AUTO	OFF	ON	ON	21.6-23.4	33.3-35.7
D	OFF	OFF	OFF	OFF	OFF	AUTO	OFF	OFF	OFF	< 10	< 24.7
B	OFF	OFF	ON	ON	OFF	AUTO	OFF	OFF	OFF	18.6-20.4	30.3-32.7
E	OFF	OFF	OFF	ON	OFF	AUTO	OFF	OFF	OFF	< 10	< 24.7
F	OFF*	OFF*	ON	ON	ON	OPEN	ON	ON	OFF	21.6-23.4	33.3-35.7
H	ON	ON	ON	ON	ON	AUTO	OFF	ON	OFF	21.6-23.5	33.3-35.7
J	ON	OFF	ON	ON	ON	AUTO	OFF	ON	OFF	21.6-23.6	33.3-35.7
K	OFF	ON	ON	ON	ON	AUTO	OFF	ON	OFF	21.6-23.7	33.3-35.7

*BT4 must be performed after positioning switch

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

ullagePressCtrlVerifWithCkOutCmds3											
Test Conditions							Verification Requirements				
Test	E3 Hi Press Cmds		Sim Cmds			LH2 Hi Flow Switch	Hi Flow Sw Scn	E3 Flow Control Valves Close Solenoid		Ullage Pressure	
	hiFlowCmdA, V41K1485N	hiFlowCmdB, V41K1486N	fcvAllLoPressCheckoutCmd, V41K0050N	fcvAllCheckoutArmCmd, V41K0051N	E2FcvHiPressCheckoutCmd, V41K0054N			E3FCV.go2ClPwr, V41X1603E	E1FCV.gh2ClPwr, V41X1663E	lo2UllagePress2 (psig), T41P1752C	lh2UllagePress2 (psia), T41P1702C
A	OFF	OFF	ON	OFF	OFF	AUTO	OFF	OFF	OFF	< 10	< 24.7
C	OFF	OFF	ON	ON	ON	AUTO	OFF	ON	ON	21.6-23.4	33.3-35.7
D	OFF	OFF	OFF	OFF	OFF	AUTO	OFF	OFF	OFF	< 10	< 24.7
B	OFF	OFF	ON	ON	OFF	AUTO	OFF	OFF	OFF	18.6-20.4	30.3-32.7
E	OFF	OFF	OFF	ON	OFF	AUTO	OFF	OFF	OFF	< 10	< 24.7
F	OFF*	OFF*	ON	ON	ON	OPEN	ON	ON	OFF	21.6-23.4	33.3-35.7
H	ON	ON	ON	ON	ON	AUTO	OFF	ON	OFF	21.6-23.5	33.3-35.7
J	ON	OFF	ON	ON	ON	AUTO	OFF	ON	OFF	21.6-23.6	33.3-35.7
K	OFF	ON	ON	ON	ON	AUTO	OFF	ON	OFF	21.6-23.7	33.3-35.7

*BT4 must be performed after positioning switch

3.2.5.3.17. llsOpenDryWetSimMated()//Phase3

3.2.5.3.17.1. Descriptive Text

This method performs liquid level sensor verification in the mated configuration, per OMRSD's V41AH0.020/.030/.040 & V41AI0.050/.060/.075

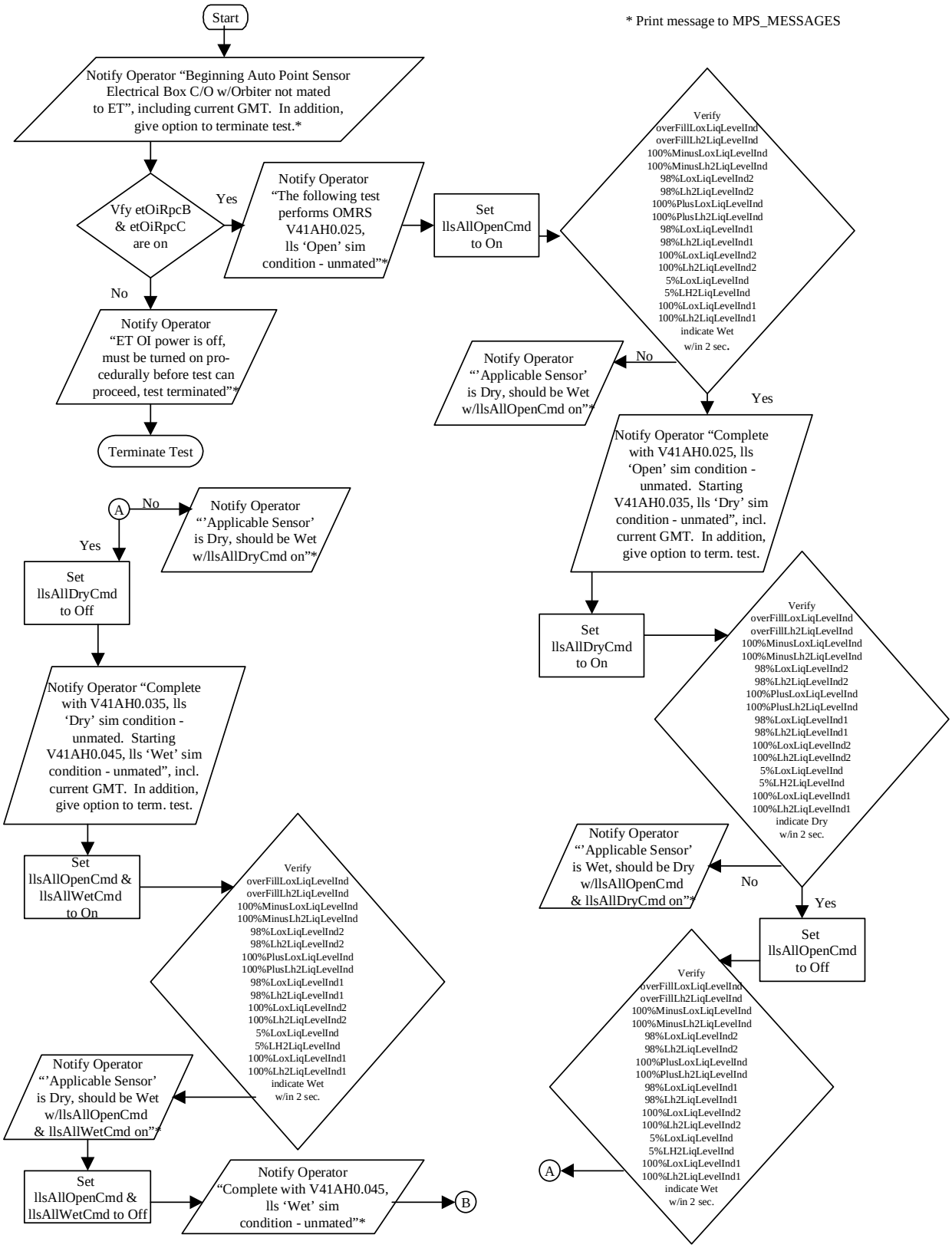
3.2.5.3.18. llsOpenDryWetSimUnmated()

3.2.5.3.18.1. Descriptive Text

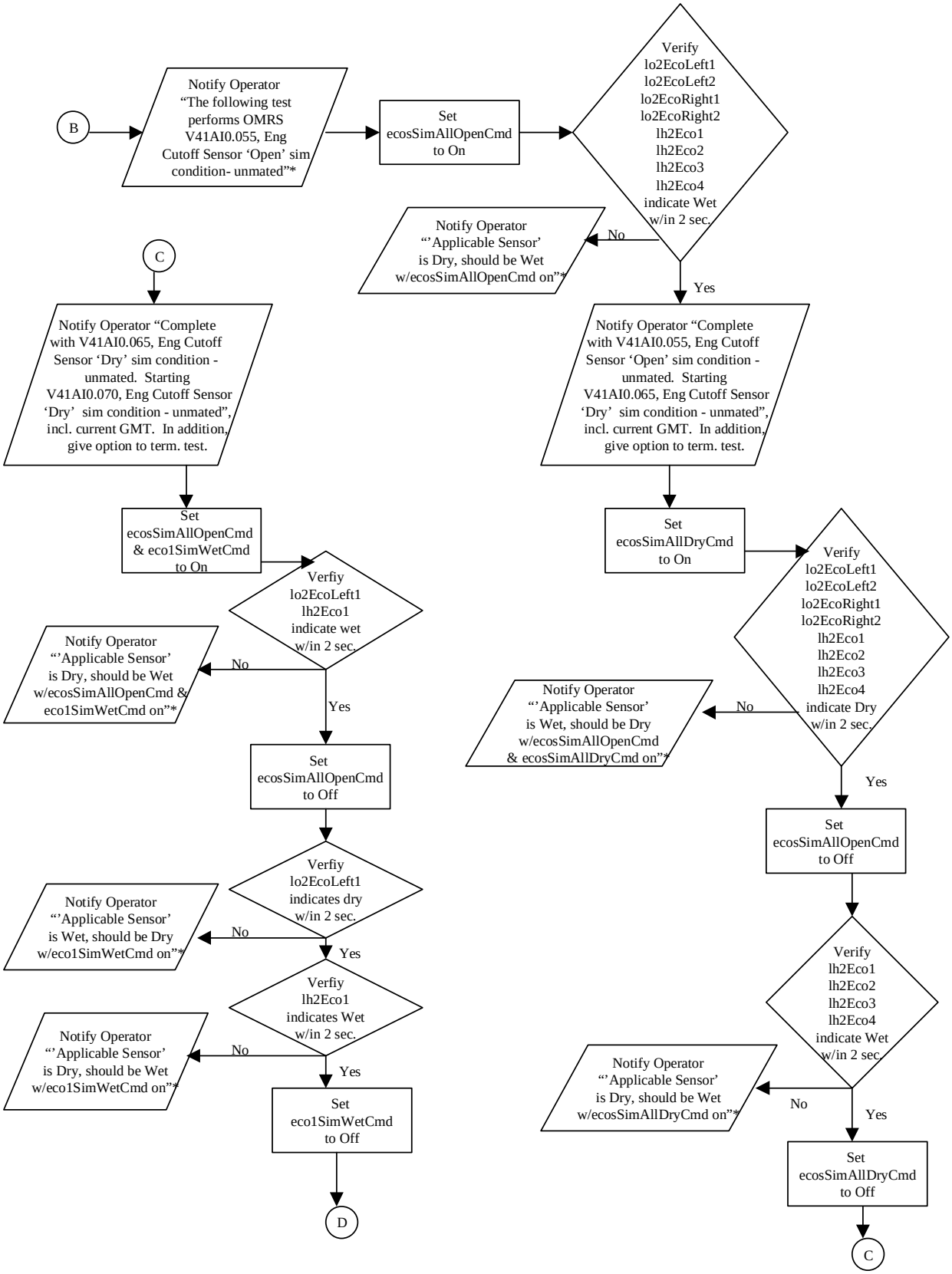
This method performs liquid level sensor verification in the mated configuration, per OMRSD's V41AH0.025/.035/.045 & V41AI0.055/.065/.070. All informational message text should be cyan, warnings in yellow and failures in red.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

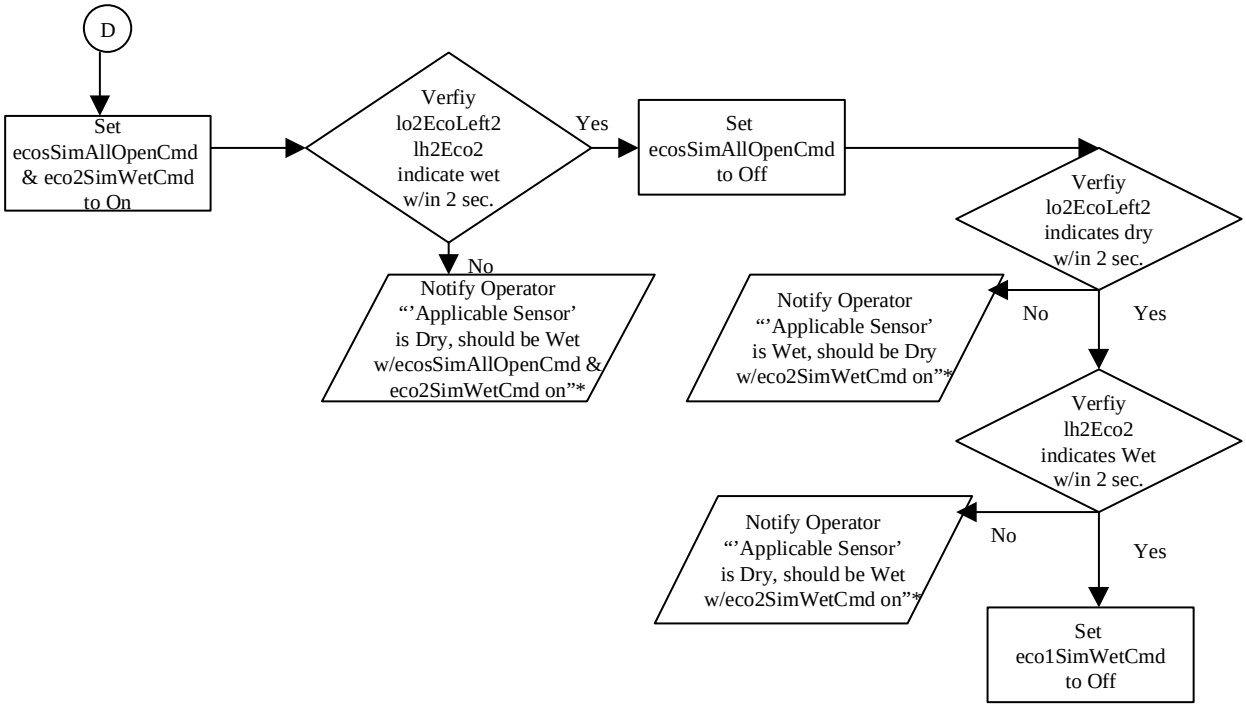
3.2.5.3.18.2. Sequence



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



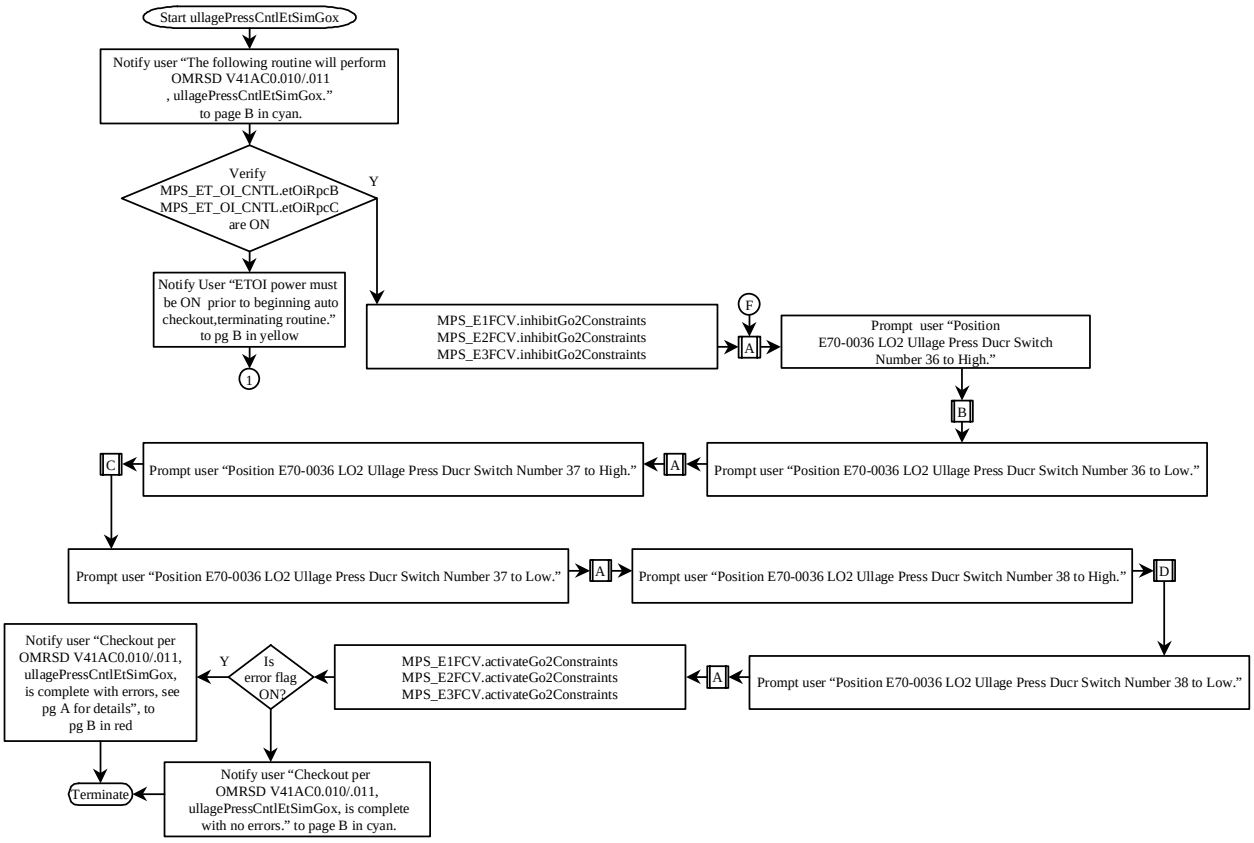
3.2.5.3.19. ullagePressCntlEtSimGox()

3.2.5.3.19.1. Descriptive Text

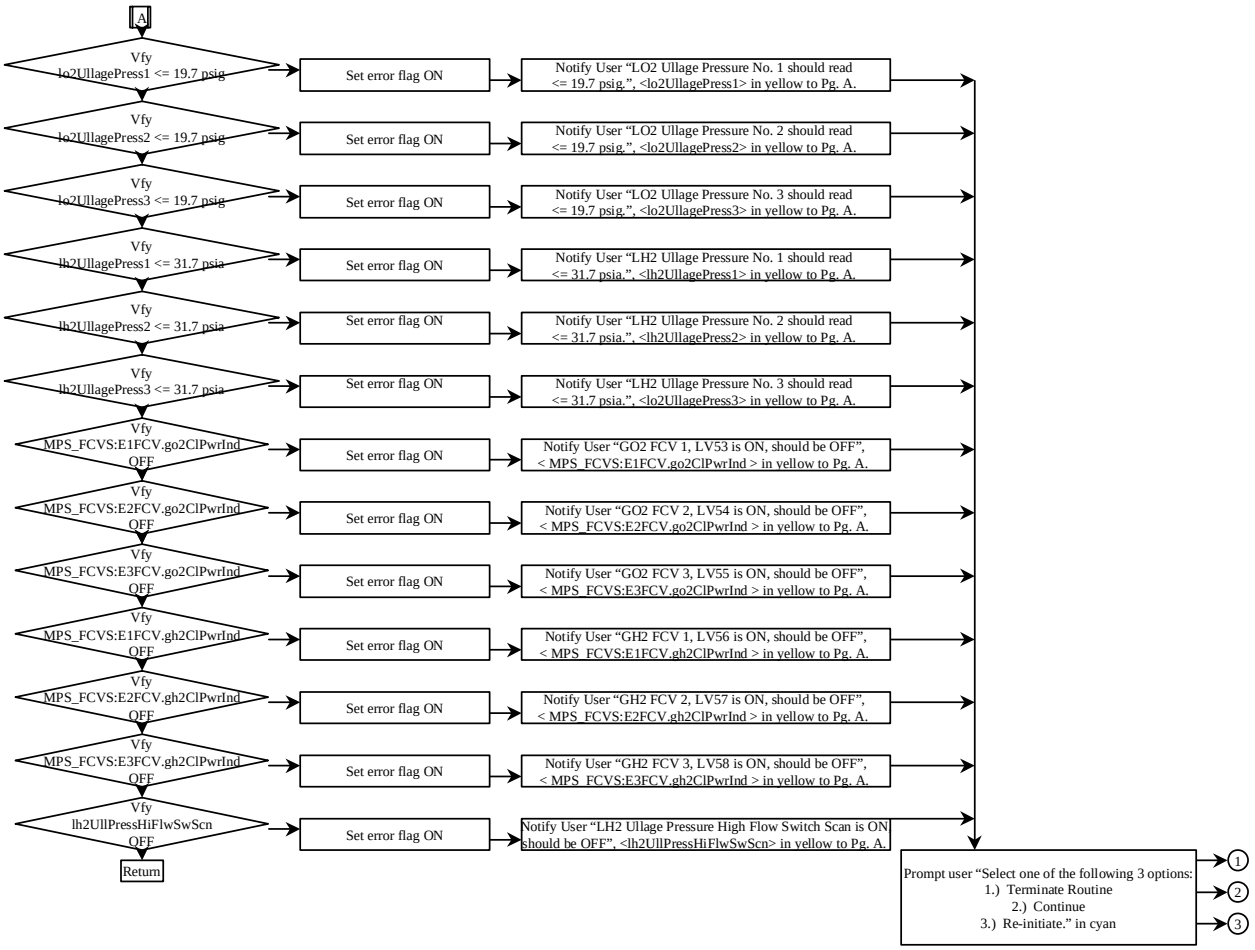
This method performs FCV ullage pressure control verification of the GO2 system per OMRSD V41AC0.010 and V41AC0.011. All prompts should be in cyan.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

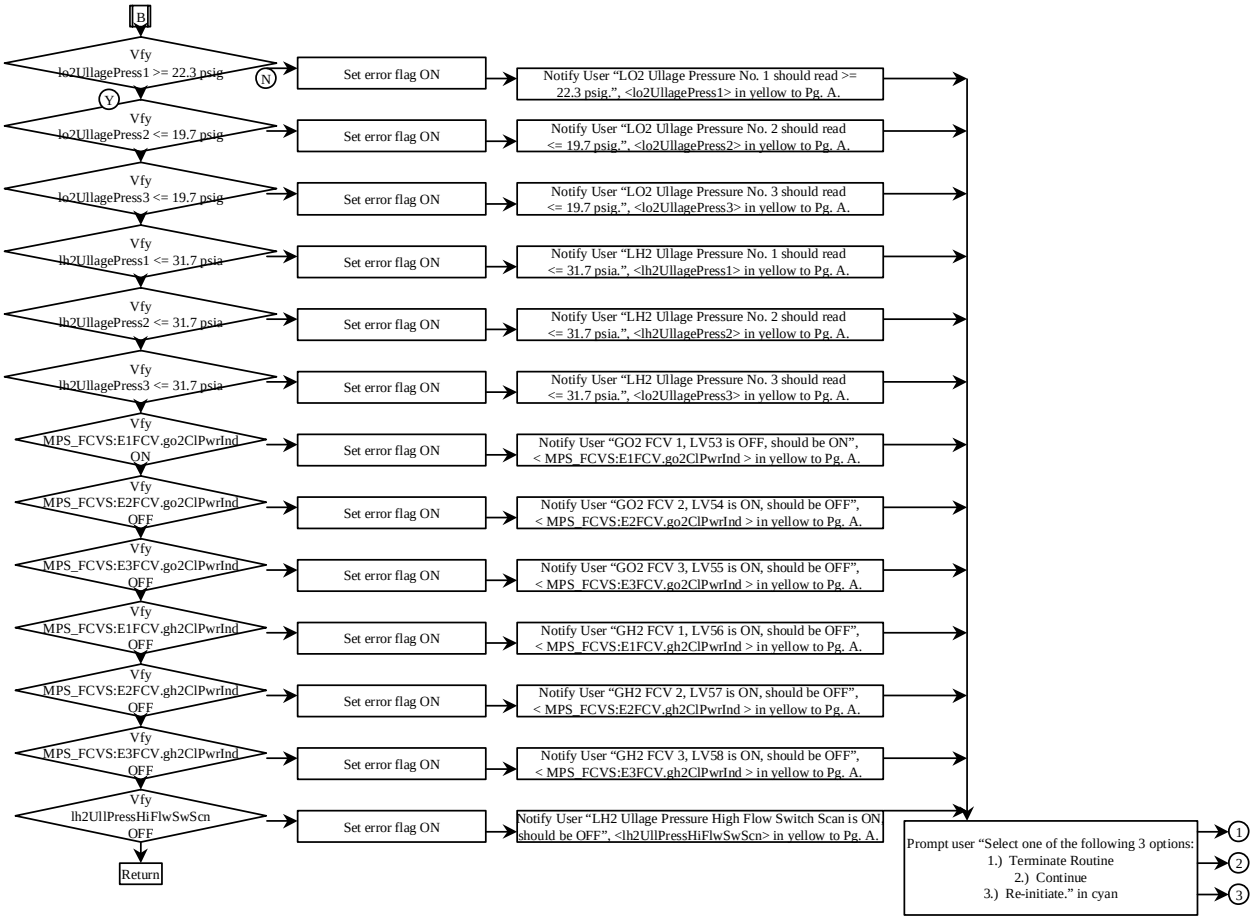
3.2.5.3.19.2. Sequence



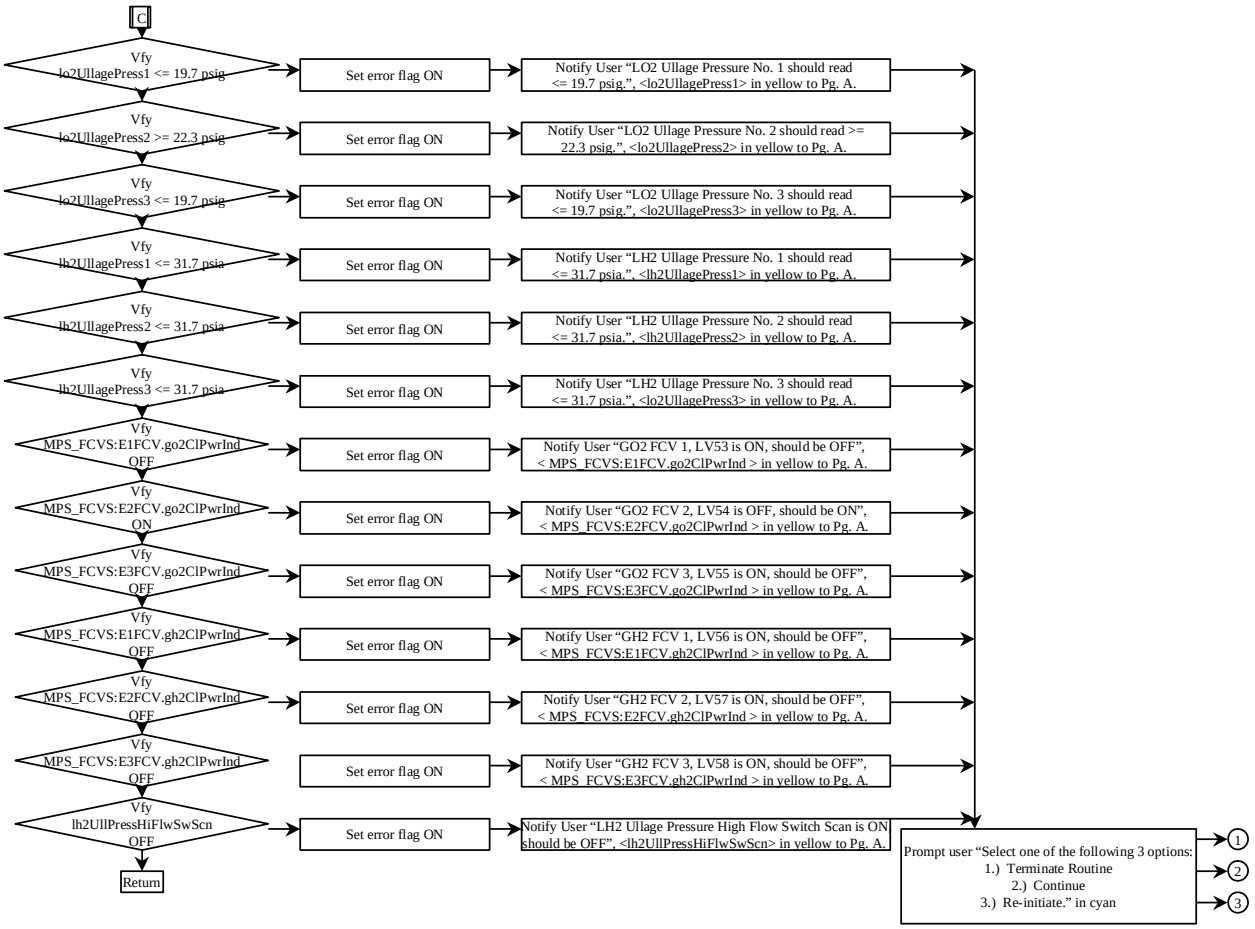
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



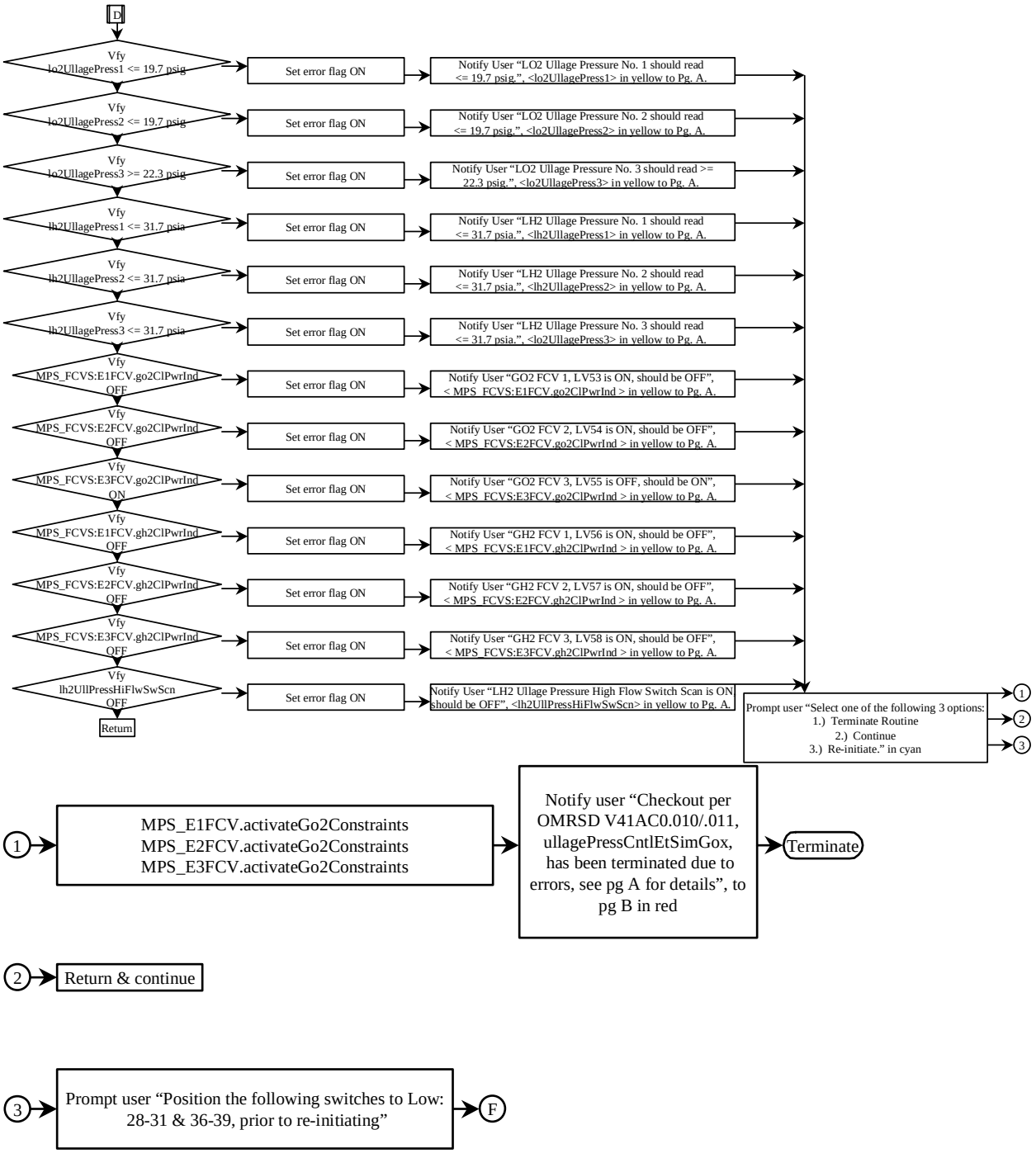
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



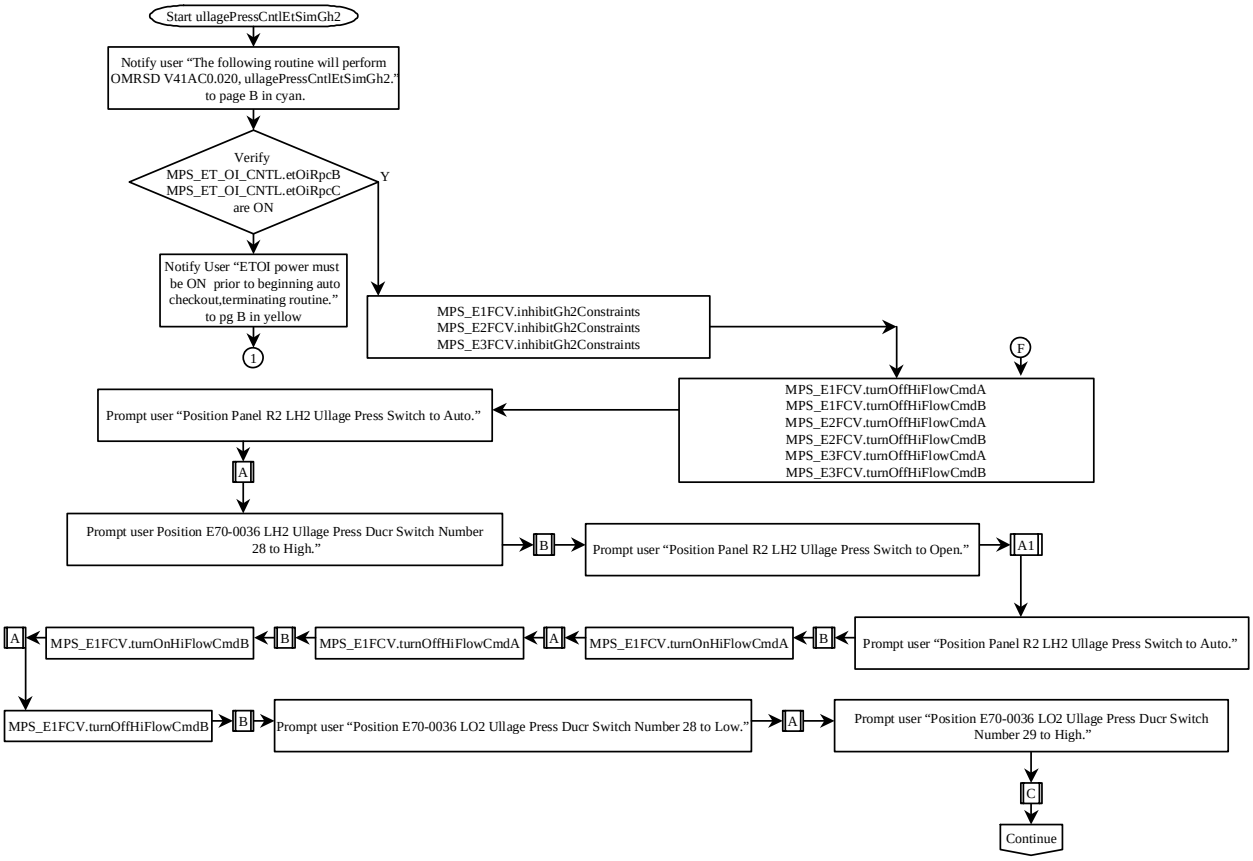
3.2.5.3.20. ullagePressCtrlEtSimGh2()

3.2.5.3.20.1. Descriptive Text

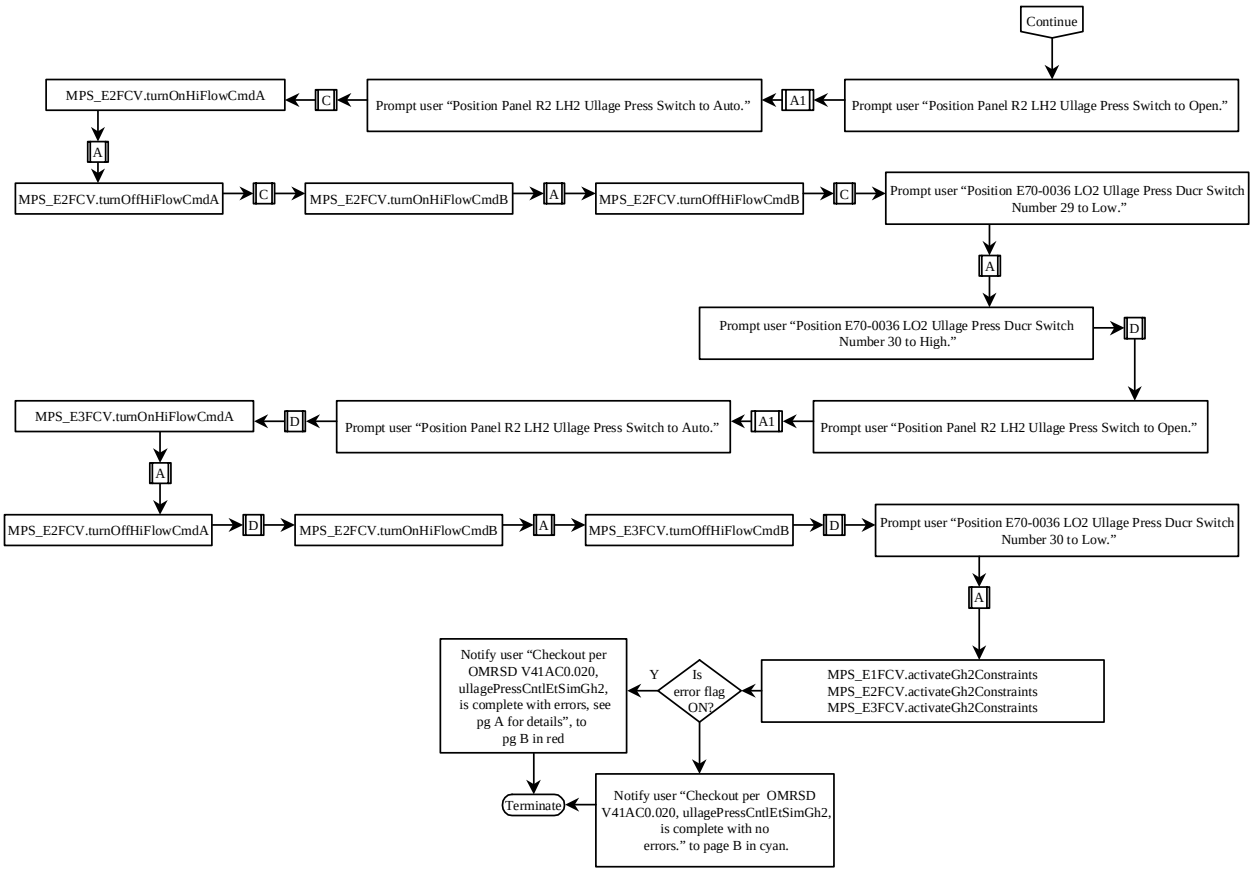
This method performs FCV ullage pressure control verification of the GH2 system per OMRSD V41AC0.020. All prompts should be in cyan

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

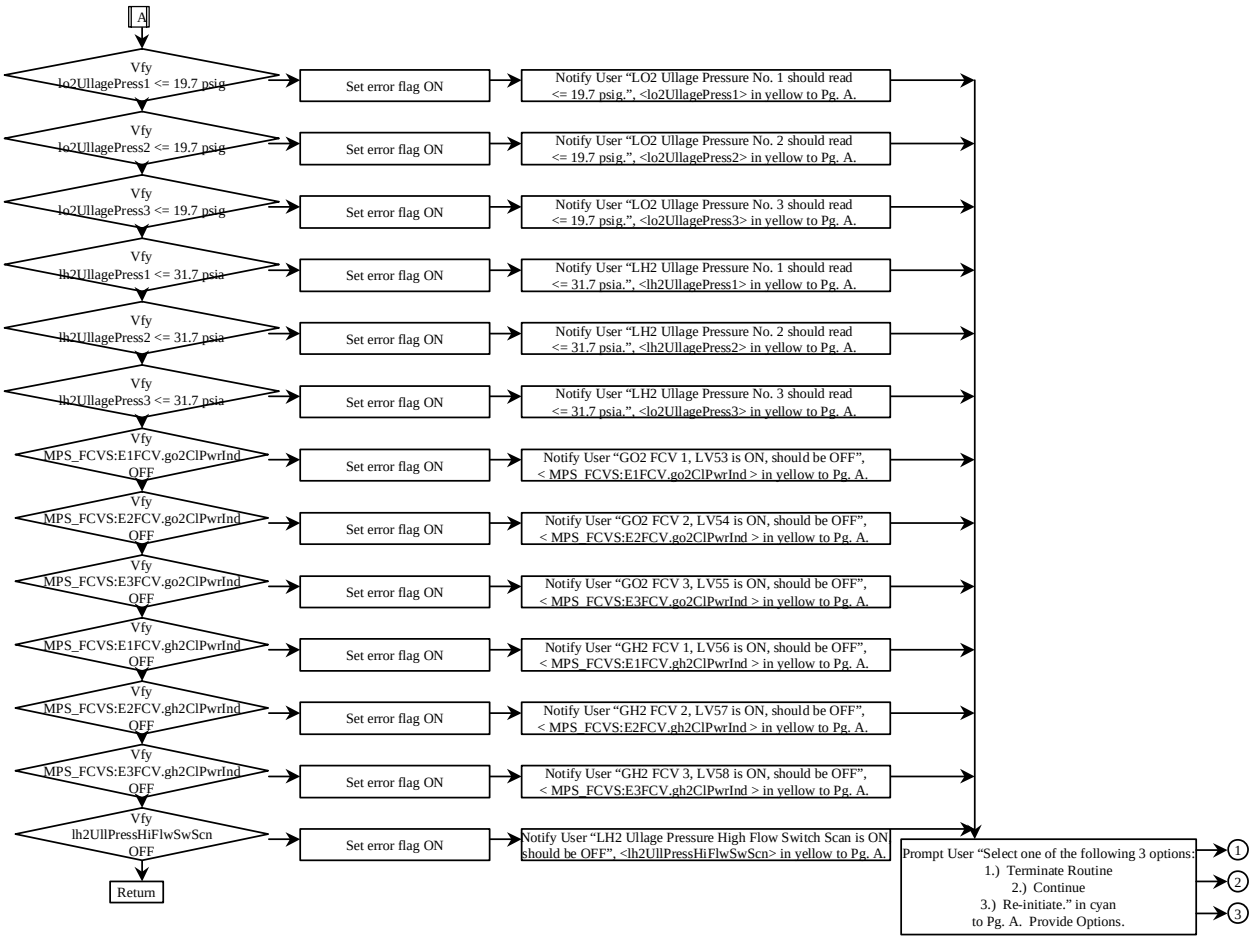
3.2.5.3.20.2. Sequence



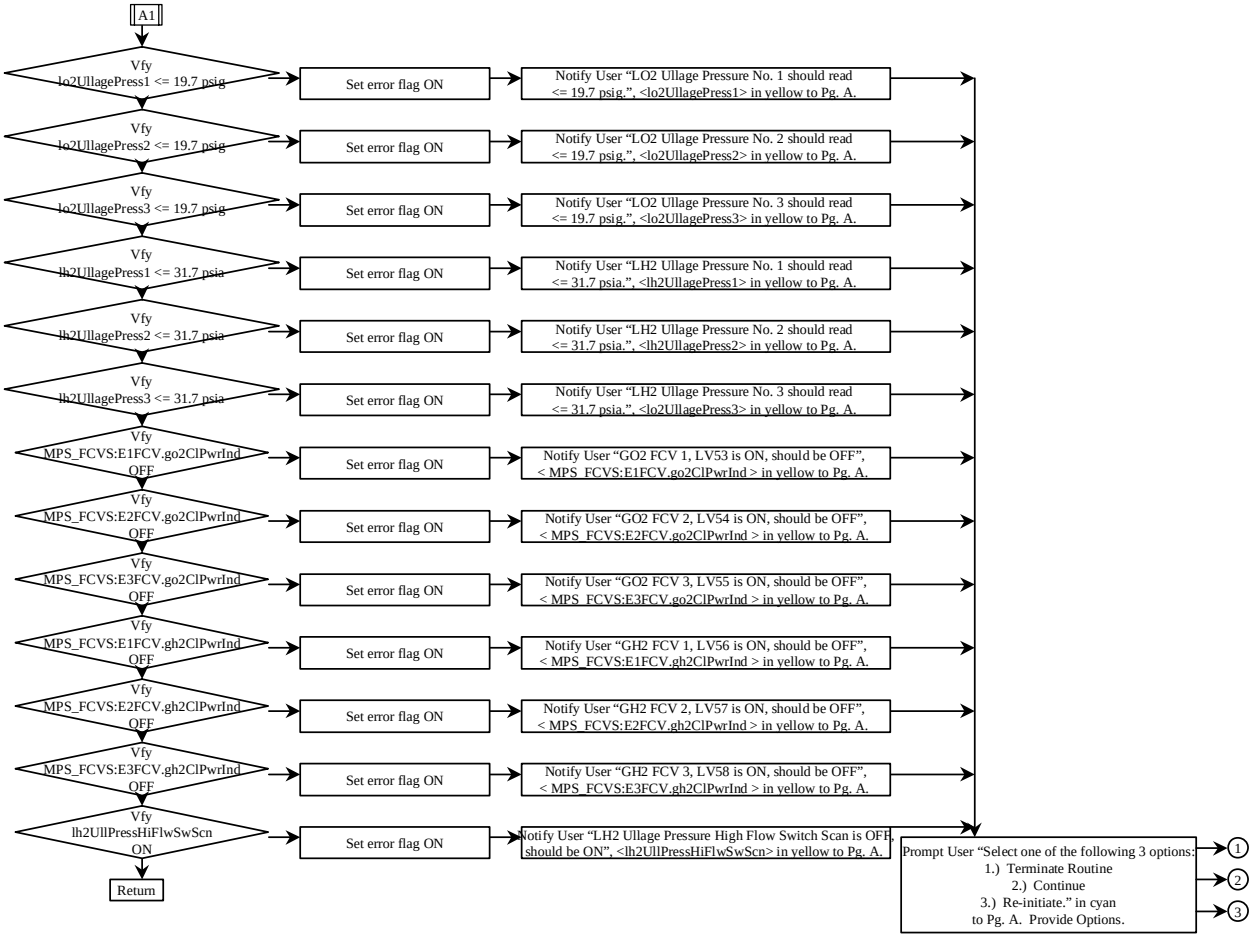
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



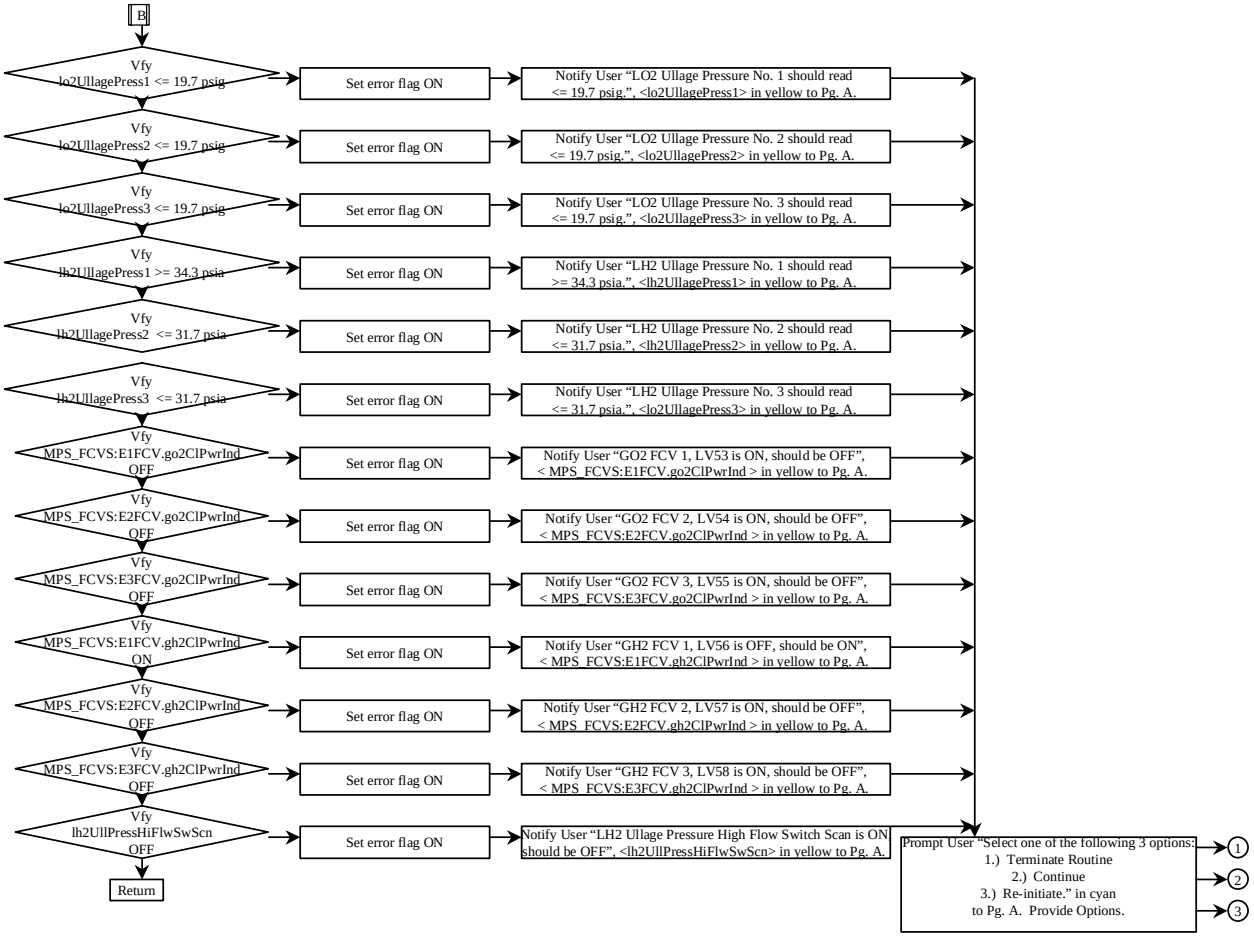
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



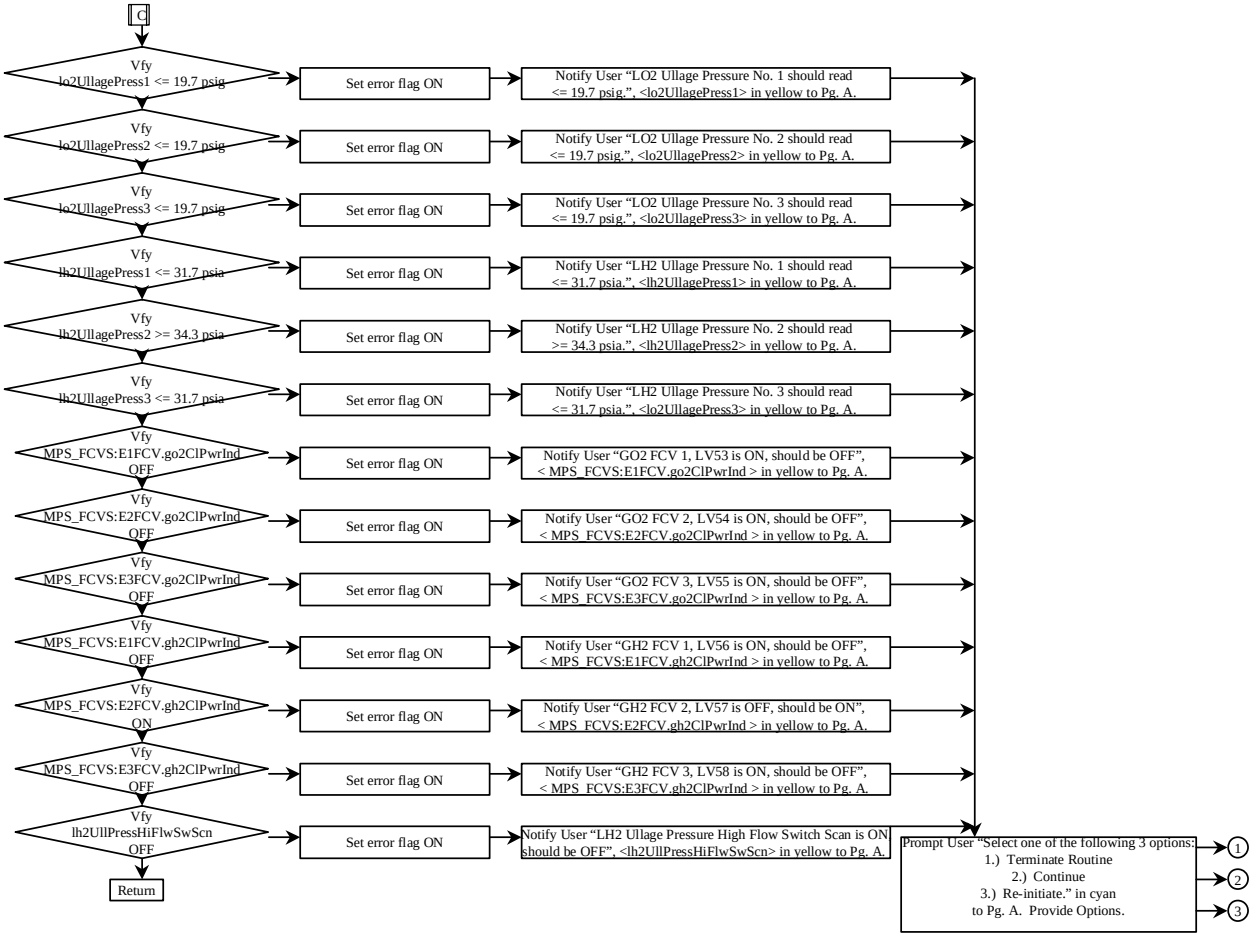
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



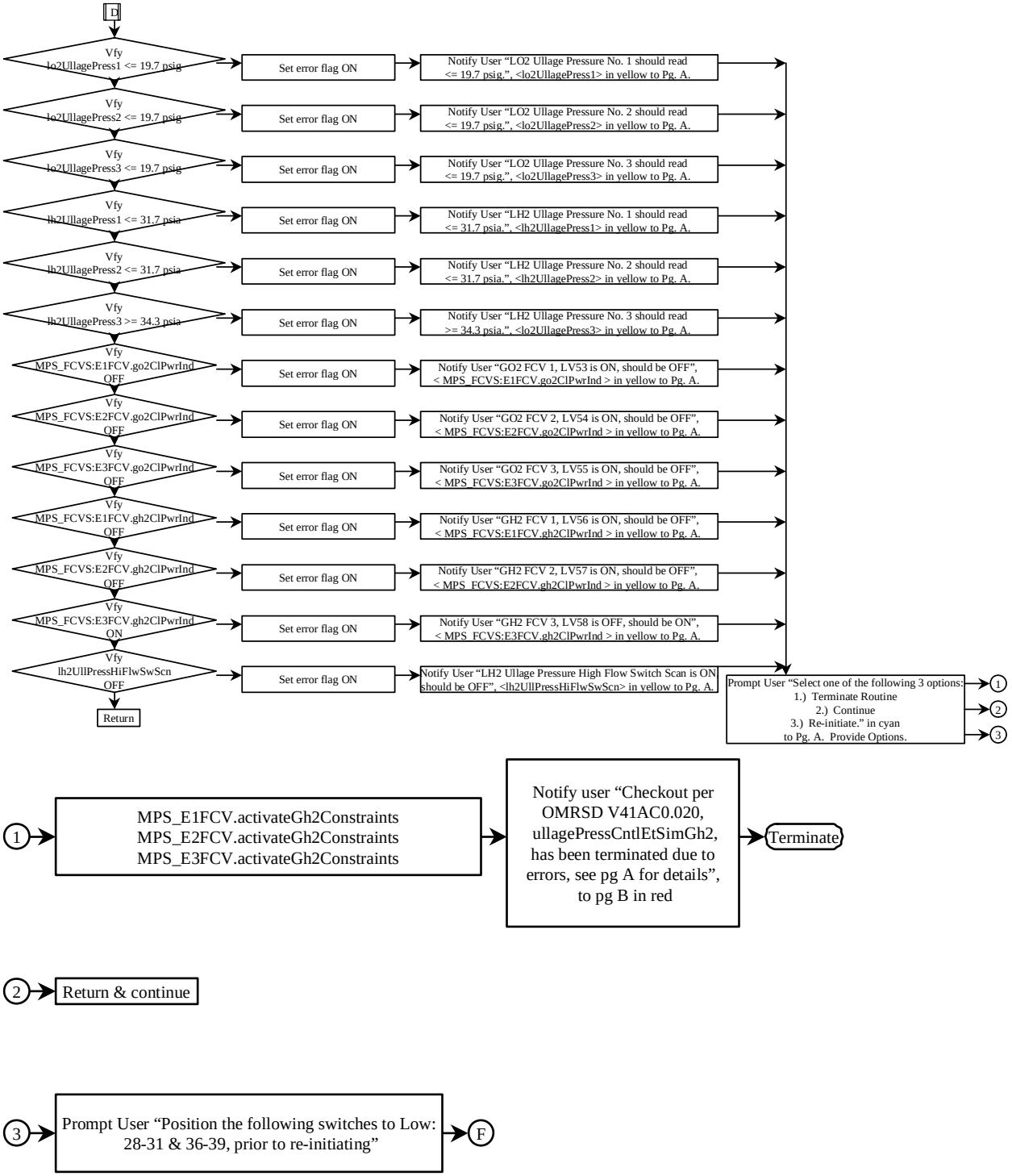
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.2.5.3.21. llsVerifEtSim()

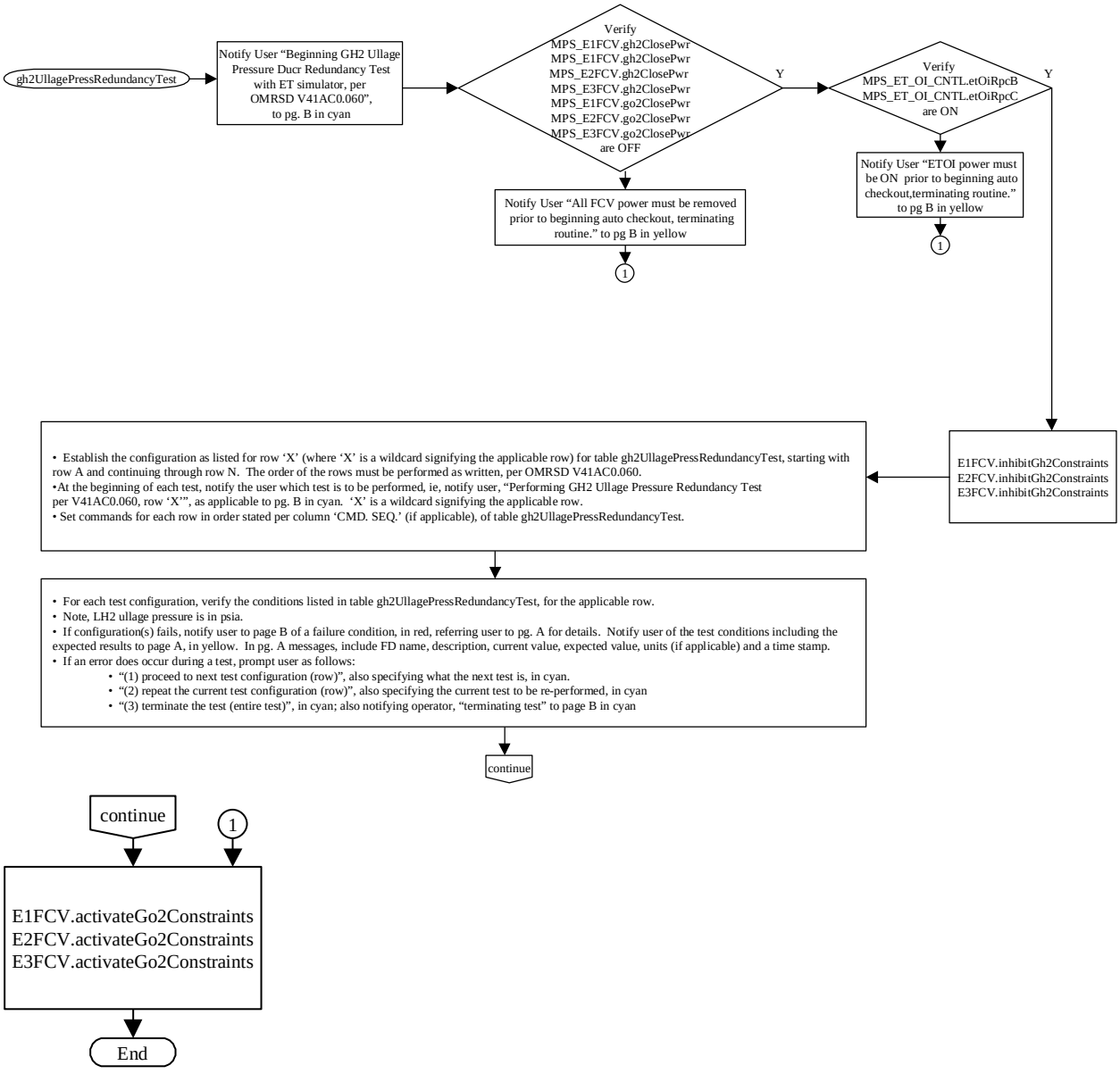
3.2.5.3.21.1. Descriptive Text

3.2.5.3.22. gh2UllagePressRedundancyWithEtSim()

3.2.5.3.22.1. Descriptive Text

This method performs GH2 ullage pressure transducer redundancy verificaton per OMRSD V41AC0.060. All prompts should be in cyan text.

3.2.5.3.22.2. Sequence



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

gh2UllagePressRedundancyTest, V41AC0.060										
	Test Conditions				Verification Requirements					
	Replace Ullage Ducr Cmds				GH2 Solenoid Vlv Close Pwr.			LOH Ullage Pressure		
	e1Lh2UllageDucrSwapCmd, V41K1700X	e2Lh2UllageDucrSwapCmd, V41K1701X	e3Lh2UllageDucrSwapCmd, V41K1702X	CMD. SEQ., Set V41K' ' ON first	E1FCV.gh2ClosePwrInd, V41X1661E	E2FCV.gh2ClosePwrInd, V41X1662E	E3FCV.gh2ClosePwrInd, V41X1663E	lh2UllagePress1, T41P1700C psia	lh2UllagePress2, T41P1701C psia	lh2UllagePress3, T41P1702C psia
A	OFF	OFF	OFF	n/a	OFF	OFF	OFF	<31.7	<31.7	<31.7
B	ON	OFF	OFF	n/a	ON	OFF	OFF	<34.3	<31.7	<31.7
C	ON	ON	OFF	'1700'	ON	OFF	OFF	<34.3	<31.7	<31.7
D	OFF	ON	OFF	n/a	OFF	ON	OFF	<31.7	<34.3	<31.7
E	OFF	ON	ON	'1701'	OFF	ON	OFF	<31.7	<34.3	<31.7
F	OFF	OFF	ON	n/a	OFF	OFF	ON	<31.7	<31.7	<34.3
G	ON	OFF	ON	'1702'	OFF	OFF	ON	<31.7	<31.7	<34.3
H	ON	OFF	OFF	n/a	ON	OFF	OFF	<34.3	<31.7	<31.7
I	ON	OFF	ON	'1700'	ON	OFF	OFF	<34.3	<31.7	<31.7
J	OFF	OFF	ON	n/a	OFF	OFF	ON	<31.7	<31.7	<34.3
K	OFF	ON	ON	'1702'	OFF	OFF	ON	<31.7	<31.7	<34.3
L	OFF	ON	OFF	n/a	OFF	ON	OFF	<31.7	<34.3	<31.7
M	ON	ON	OFF	'1701'	OFF	ON	OFF	<31.7	<34.3	<31.7
N	ON	OFF	OFF	n/a	ON	OFF	OFF	<34.3	<31.7	<31.7

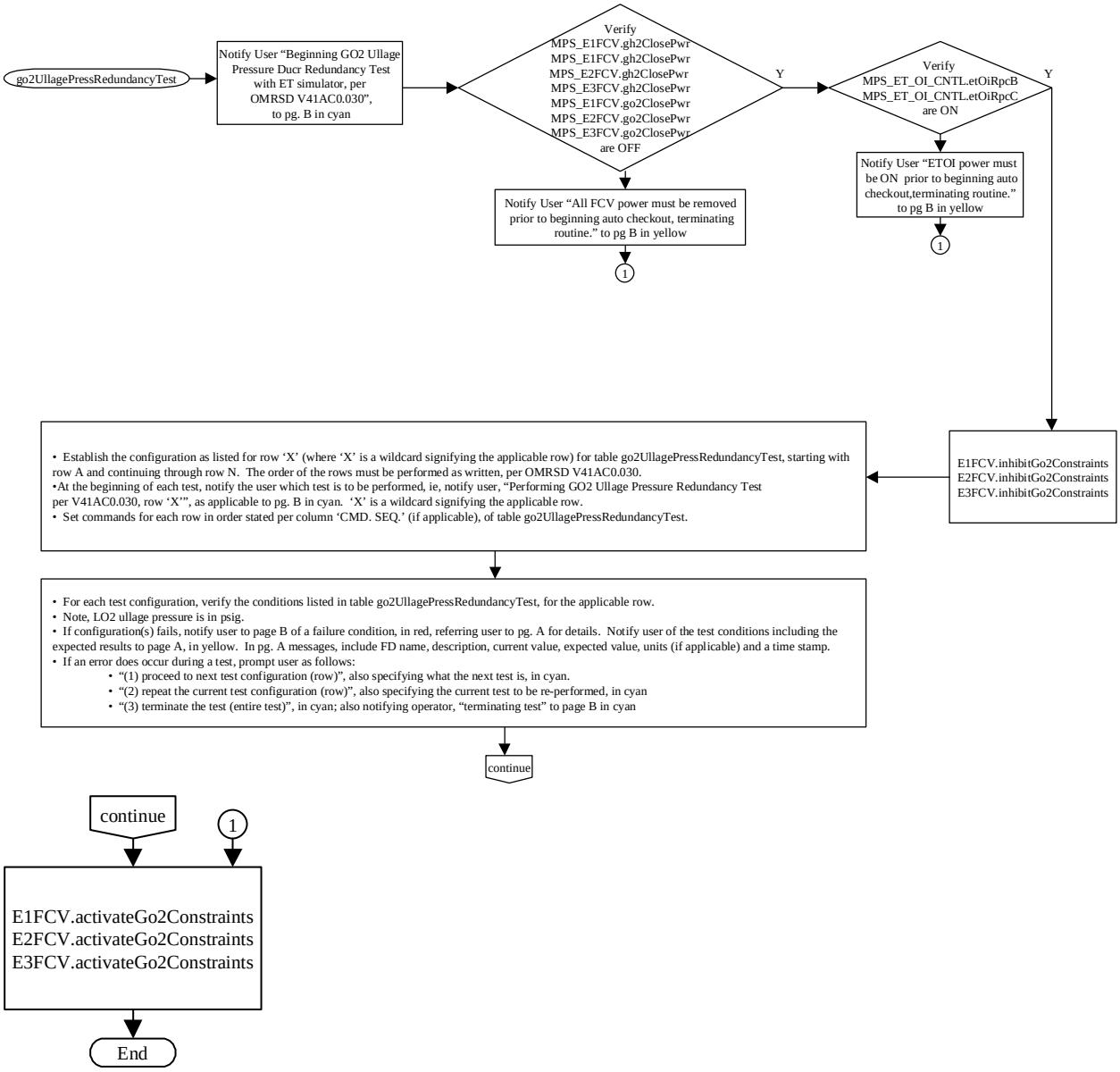
3.2.5.3.23. go2UllagePressRedundancyWithEtSim()

3.2.5.3.23.1. Descriptive Text

This method performs GO2 ullage pressure transducer redundancy verificaton per OMRSD V41AC0.030. All prompts should be in cyan text.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.2.5.3.23.2. Sequence



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

go2UllagePressRedundancyTest, V41AC0.030										
	Test Conditions				Verification Requirements					
	Replace Ullage Ducr Cmds				GO2 Solenoid Vlv Close Pwr.			LO2 Ullage Pressure		
	e1Lo2UllageDucrSwapCmd, V41K1750X	e2Lo2UllageDucrSwapCmd, V41K1751X	e3Lo2UllageDucrSwapCmd, V41K1752X	Cmd. Seq. Set V41K' ' ON first	E1FCV.go2ClosePwrInd, V41X1596E	E2FCV.go2ClosePwrInd, V41X1598E	E3FCV.go2ClosePwrInd, V41X1603E	lo2UllagePress1, T41P1750C psig	lo2UllagePress2, T41P1751C psig	lo2UllagePress3, T41P1752C psig
A	OFF	OFF	OFF	n/a	OFF	OFF	OFF	<19.7	<19.7	<19.7
B	ON	OFF	OFF	n/a	ON	OFF	OFF	<22.3	<19.7	<19.7
C	ON	ON	OFF	'1750'	ON	OFF	OFF	<22.3	<19.7	<19.7
D	OFF	ON	OFF	n/a	OFF	ON	OFF	<19.7	<22.3	<19.7
E	OFF	ON	ON	'1751'	OFF	ON	OFF	<19.7	<22.3	<19.7
F	OFF	OFF	ON	n/a	OFF	OFF	ON	<19.7	<19.7	<22.3
G	ON	OFF	ON	'1752'	OFF	OFF	ON	<19.7	<19.7	<22.2
H	ON	OFF	OFF	n/a	ON	OFF	OFF	<22.3	<19.7	<19.7
I	ON	OFF	ON	'1750'	ON	OFF	OFF	<22.3	<19.7	<19.7
J	OFF	OFF	ON	n/a	OFF	OFF	ON	<19.7	<19.7	<22.3
K	OFF	ON	ON	'1752'	OFF	OFF	ON	<19.7	<19.7	<22.3
L	OFF	ON	OFF	n/a	OFF	ON	OFF	<19.7	<22.3	<19.7
M	ON	ON	OFF	'1751'	OFF	ON	OFF	<19.7	<22.3	<19.7
N	ON	OFF	OFF	n/a	ON	OFF	OFF	<22.3	<19.7	<19.7

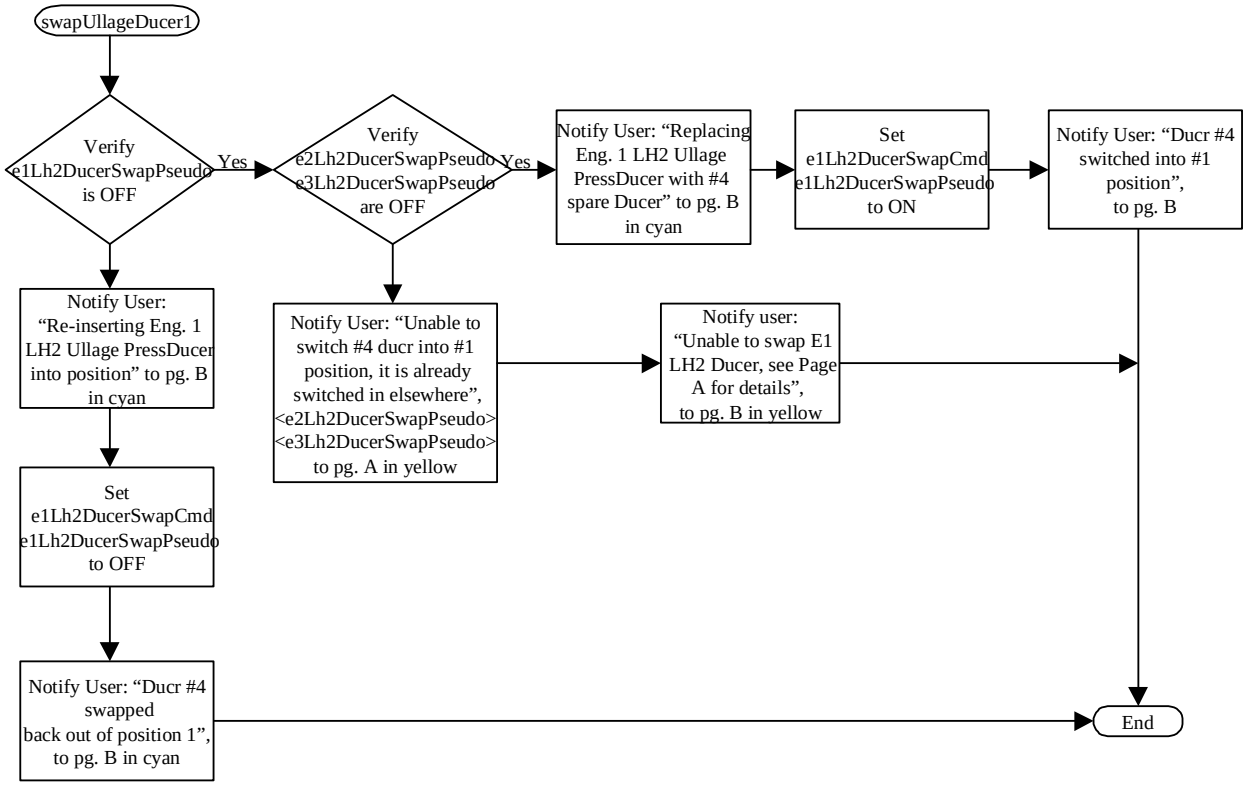
3.2.5.3.24. swapUllageDucer1

3.2.5.3.24.1. Descriptive Text

This method will swap in the spare GH2 ullage pressure transducer number 4 into the engine number 1 location and set its associated pseudo accordingly.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.2.5.3.24.2. Sequence



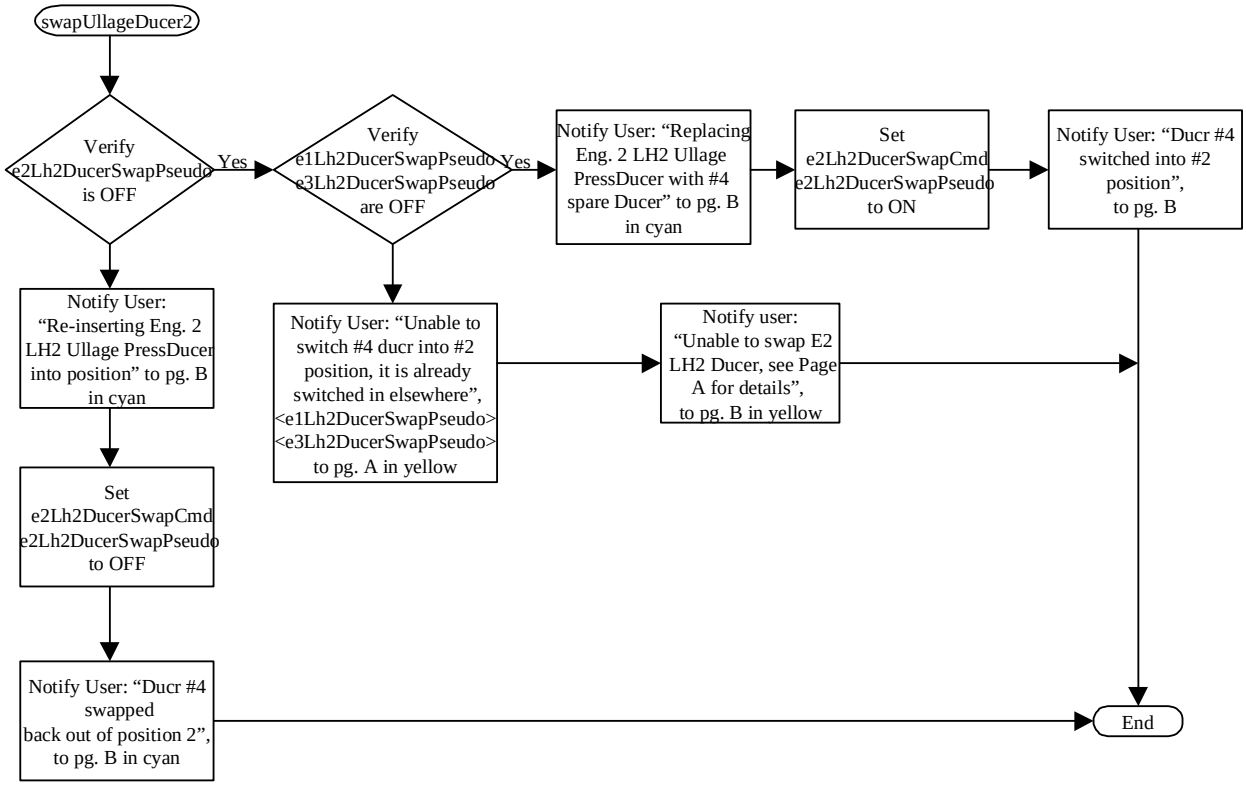
3.2.5.3.25. swapUllageDucer2

3.2.5.3.25.1. Descriptive Text

This method will swap in the spare GH2 ullage pressure transducer number 4 into the engine number 2 location and set its associated pseudo accordingly.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.2.5.3.25.2. Sequence



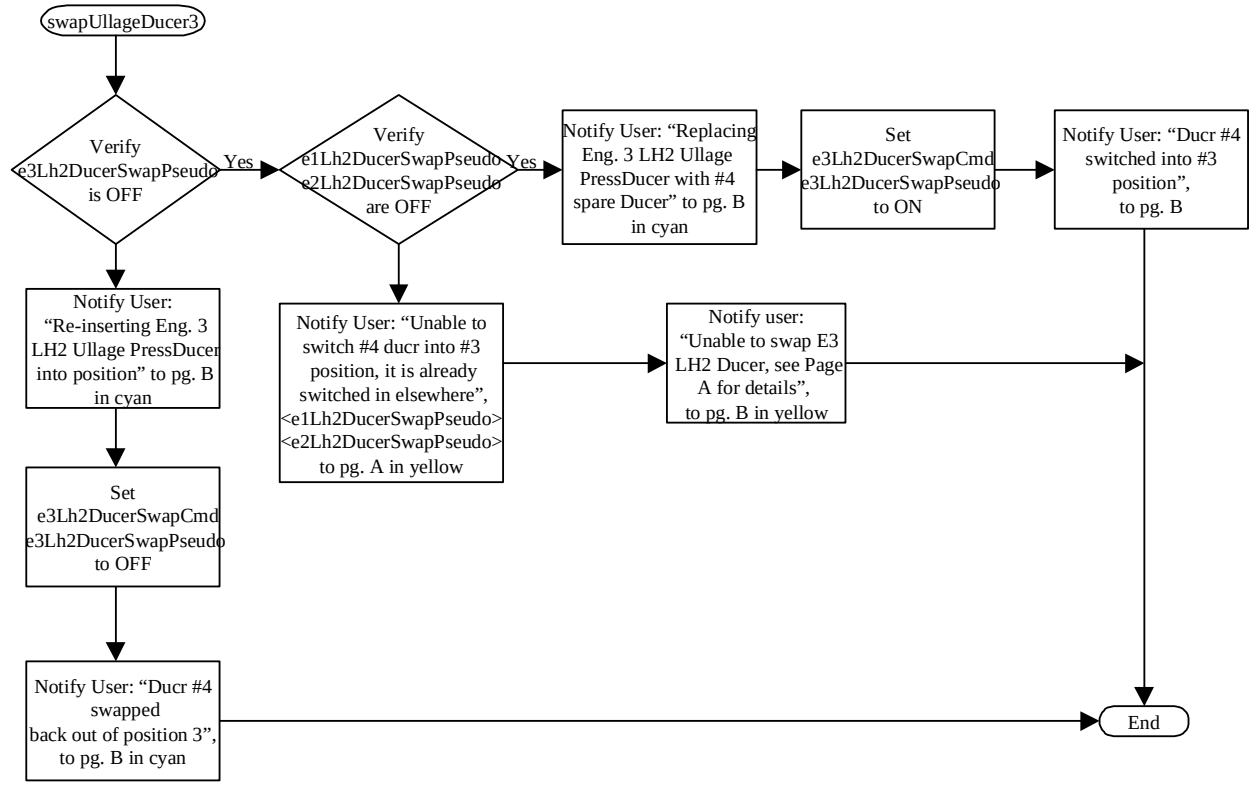
3.2.5.3.26. swapUllageDucer3

3.2.5.3.26.1. Descriptive Text

This method will swap in the spare GH2 ullage pressure transducer number 4 into the engine number 3 location and set its associated pseudo accordingly.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.2.5.3.26.2. Sequence



3.2.5.3.27. verifyCommandsOff()

3.2.5.3.27.1. Descriptive Text

This function will verify that ETOI power commands are OFF

3.2.5.3.27.2. Sequence

```
verifyCommandsOff()
{
        etoi command status1 = byteTest4 <etOiPwrCmd1>
                verify  etoi command status1 == OFF
                else send message "etoi pwr command1;<etOiPwrCmd1>; " is ON, should be OFF" to
                        Page A

        etoi command status2 = byteTest4 <etOiPwrCmd2>
        verify  etoi command status2 == OFF
                else send message "etoi pwr command2;<etOiPwrCmd2>; " is ON, should be OFF" to
                        Page A

        etoi command status3 = byteTest4 <etOiPwrCmd3>
        verify  etoi command status3 == OFF
                else send message "etoi pwr command3;<etOiPwrCmd3>; " is ON, should be OFF" to
                        Page A
        {
        verify  etoi command status1 && etoi command status2 && etoi command status3 == OFF

        return TRUE
        }
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
        ELSE return FALSE
}
```

3.2.5.3.28. verifyTminus6BT4()//Phase3

3.2.5.3.29. verifyTminus3BT4()//Phase3

3.2.5.3.30. issueE1FcvCmds()

3.2.5.3.30.1. Descriptive Text

This method will set the fcvAllCheckoutArmCmd and the e1FcvHiPressCheckoutCmd to ON to allow energization of a FCV via the MPS_GH2_FCVS class

3.2.5.3.30.2. Sequence

```
issueE1Gh2FcvCmds()
{
        set:

        fcvAllCheckoutArmCmd
        e1FcvHiPressCheckoutCmd

        to ON
}
```

3.2.5.3.31. issueE2FcvCmds()

3.2.5.3.31.1. Descriptive Text

This method will set the fcvAllCheckoutArmCmd and the e2FcvHiPressCheckoutCmd to ON to allow energization of a FCV via the MPS_GH2_FCVS class

3.2.5.3.31.2. Sequence

```
issueE2Gh2FcvCmds()
{
        set:

        fcvAllCheckoutArmCmd
        e2FcvHiPressCheckoutCmd

        to ON
}
```

3.2.5.3.32. issueE3FcvCmds()

3.2.5.3.32.1. Descriptive Text

This method will set the fcvAllCheckoutArmCmd and the e3FcvHiPressCheckoutCmd to ON to allow energization of a FCV via the MPS_GH2_FCVS class

3.2.5.3.32.2. Sequence

```
issueE3Gh2FcvCmds()
{
        set:

        fcvAllCheckoutArmCmd
        e3FcvHiPressCheckoutCmd

        to ON
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

}

3.2.5.3.33. removeE1FcvCmds()

3.2.5.3.33.1. Descriptive Text

This method will set the fcvAllCheckoutArmCmd and the e1FcvHiPressCheckoutCmd to OFF to allow deenergization of a FCV via the MPS_GH2_FCVS class

3.2.5.3.33.2. Seqence

```
removeE1Gh2FcvCmds()
{
        set:

        fcvAllCheckoutArmCmd
        e1FcvHiPressCheckoutCmd

        to OFF
}
```

3.2.5.3.34. removeE2FcvCmds()

3.2.5.3.34.1. Descriptive Text

This method will set the fcvAllCheckoutArmCmd and the e2FcvHiPressCheckoutCmd to OFF to allow deenergization of a FCV via the MPS_GH2_FCVS class

3.2.5.3.34.2. Sequence

```
removeE2Gh2FcvCmds()
{
        set:

        fcvAllCheckoutArmCmd
        e2FcvHiPressCheckoutCmd

        to OFF
}
```

3.2.5.3.35. removeE3FcvCmds()

3.2.5.3.35.1. Descriptive Text

This method will set the fcvAllCheckoutArmCmd and the e3FcvHiPressCheckoutCmd to OFF to allow deenergization of a FCV via the MPS_GH2_FCVS class

3.2.5.3.35.2. Sequence

```
removeE3Gh2FcvCmds()
{
        set:

        fcvAllCheckoutArmCmd
        e3FcvHiPressCheckoutCmd

        to OFF
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.2.5.4. User Interfaces

3.2.5.5. Safety Requirements

3.2.6. MPS_LH2_Control_Logic_Manager

3.2.6.1. Description

This Class is an aggregation of all the associated MPS_LH2 and MPS_GHe Prerequisite Control Logic and Reactive Control Logic. The class will contain the attributes and methods necessary to provide status and housekeeping of the Control Logic Subsystem.

3.2.6.1.1. Instance Index

Instance	Other Users
MPS_LH2_Control_Logic_Manager	

3.2.6.1.2. Class Diagram

<insert class diagram here>

3.2.6.1.3. Attributes and Aggregations

Attribute	Data Type	Initial Value	Publish / Persistence	Client Access	Description
CLNotificationFlag	PD_ON_OFF		PUB		
Control_Logic_Mode	MPS_CL_Mode		PUB		
MPS_RCL_GHe_Bottle_Press	MPS_RCL_GHe_Bottle_Press				
MPS_RCL_Pneu_Reg_Press	MPS_RCL_Pneu_Reg_Press				
MPS_RCL_E1_Reg_Press	MPS_RCL_E1_Reg_Press				
MPS_RCL_E2_Reg_Press	MPS_RCL_E2_Reg_Press				
MPS_RCL_E3_Reg_Press	MPS_RCL_E3_Reg_Press				
MPS_RCL_GHe_Bottle_Temp	MPS_RCL_GHe_Bottle_Temp				
MPS_RCL_LH2_Manifold_Press	MPS_RCL_LH2_Manifold_Press				
MPS_RCL_LH2_PV11_OP_Ind	MPS_RCL_LH2_PV11_OP_Ind				
MPS_RCL_LH2_PV12_OP_Ind	MPS_RCL_LH2_PV12_OP_Ind				
MPS_RCL_LH2_PV13_OP_Ind	MPS_RCL_LH2_PV13_OP_Ind				
MPS_RCL_LH2_PD3_OP_Ind	MPS_RCL_LH2_PD3_OP_Ind				

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

MPS_RCL_LH2_PD2_OP_Ind	MPS_RCL_LH2_PD2_OP_Ind				
MPS_PCL_PV11_CL_CMD	MPS_PCL_PV11_CL_CMD				
MPS_PCL_PV11_OP_CMD	MPS_PCL_PV11_OP_CMD				
MPS_PCL_PV12_CL_CMD	MPS_PCL_PV12_CL_CMD				
MPS_PCL_PV12_OP_CMD	MPS_PCL_PV12_OP_CMD				
MPS_PCL_PV13_OP_CMD	MPS_PCL_PV13_OP_CMD				
MPS_PCL_PD2_CL_CMD	MPS_PCL_PD2_CL_CMD				
MPS_PCL_PD2_OP_CMD	MPS_PCL_PD2_OP_CMD				
MPS_PCL_PD2_LATCH_LK_CMD	MPS_PCL_PD2_LATCH_LK_CMD				
MPS_PCL_PD2_LATCH_UL_CMD	MPS_PCL_PD2_LATCH_UL_CMD				
MPS_PCL_PD3_CL_CMD	MPS_PCL_PD3_CL_CMD				
MPS_PCL_PD3_OP_CMD	MPS_PCL_PD3_OP_CMD				
MPS_PCL_PV8_CL_CMD	MPS_PCL_PV8_CL_CMD				
MPS_PCL_LV74_OP_CMD	MPS_PCL_LV74_OP_CMD				
MPS_PCL_LV75_OP_CMD	MPS_PCL_LV75_OP_CMD				
MPS_PCL_RECIRC_OP_CMD	MPS_PCL_RECIRC_OP_CMD				
MPS_PCL_RTLS_DUMP_OP_CMD	MPS_PCL_RTLS_DUMP_OP_CMD				
MPS_PCL_PT_SENS_PWR_CMD	MPS_PCL_PT_SENS_PWR_CMD				
MPS_PCL_FCV1_CO_CMD	MPS_PCL_FCV1_CO_CMD				
MPS_PCL_FCV2_CO_CMD	MPS_PCL_FCV2_CO_CMD				
MPS_PCL_FCV3_CO_CMD	MPS_PCL_FCV3_CO_CMD				
MPS_PCL_PNEU_OP_CMD	MPS_PCL_PNEU_OP_CMD				
MPS_PCL_LV10_OP_CMD	MPS_PCL_LV10_OP_CMD				
MPS_PCL_BLOWDOWN_OP_CMD	MPS_PCL_BLOWDOWN_OP_CMD				

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.2.6.1.4. Operations Index

Operation	Description
setCLModeReadyForLoad()	This method will set up the Control Logic to support Propellant Loading Operations.
setCLModeTest()	This method will set up the Control Logic to support Test Operations.
setCLModeTermCount()	This method will set up the Control Logic to support Terminal Count Operations.
getState()	This method will return the Control_Logic_Mode
determineState()	This method will determine the Control_Logic_Mode
determineCLNotification()	This method will set the CLNotificationFlag to ON if any of the Aggregated CLNotificationFlags are ON Else it will be set to OFF.
getCLNotification()	This method will return the CLNotificationFlag.
activateLoadingRCL()	This method will activate the LH2 Loading Reactive Control Logic.
inhibitLoadingRCL()	This method will inhibit the LH2 Loading Reactive Control Logic.
resetCLNotification()	This method will set all CL notification flags off by running the resetCLNotification() method for all associated PCL/RCL sequences.
ReportPCLBlock()	This method will report a PCL Block message based on the associated CLNotificationFlag

3.2.6.1.5. Associations

Association	Type	Class	Description

3.2.6.2. Operations

3.2.6.2.1. setCLModeReadyForLoad()

3.2.6.2.1.1. Descriptive Text

This method will set up the Control Logic to support Propellant Loading Operations.

3.2.6.2.1.2. Sequence

```
// setCLModeReadyForLoad()


MPS_RCL_GHe_Bottle_Press.inhibit();
MPS_RCL_GHe_Bottle_Press.alter(LO,4550);
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

MPS_RCL_GHe_Bottle_Press.activate();
MPS_RCL_Pneu_Reg_Press.inhibit();
MPS_RCL_E1_Reg_Press.inhibit();
MPS_RCL_E2_Reg_Press.inhibit();
MPS_RCL_E3_Reg_Press.inhibit();
MPS_RCL_GHe_Bottle_Temp.inhibit();
MPS_RCL_LH2_Manifold_Press.inhibit();
MPS_RCL_LH2_PV11_OP_Ind.inhibit();
MPS_RCL_LH2_PV12_OP_Ind.activate();
MPS_RCL_LH2_PV13_OP_Ind.inhibit();
MPS_RCL_LH2_PD3_OP_Ind.activate();
MPS_RCL_LH2_PD2_OP_Ind.activate();
MPS_PCL_PV11_CL_CMD.activate();
MPS_PCL_PV11_OP_CMD.activate();
MPS_PCL_PV12_CL_CMD.activate();
MPS_PCL_PV12_OP_CMD.activate();
MPS_PCL_PV13_OP_CMD.activate();
MPS_PCL_PD2_CL_CMD.activate();
MPS_PCL_PD2_OP_CMD.activate();
MPS_PCL_PD2_LATCH_LK_CMD.activate();
MPS_PCL_PD2_LATCH_UL_CMD.activate();
MPS_PCL_PD3_CL_CMD.activate();
MPS_PCL_PD3_OP_CMD.activate();
MPS_PCL_PV8_CL_CMD.inhibit();
MPS_PCL_LV74_OP_CMD.activate();
MPS_PCL_LV75_OP_CMD.actiavte();
MPS_PCL_RECIRC_OP_CMD.inhibit();
MPS_PCL_RTLS_DUMP_OP_CMD.activate();
MPS_PCL_PT_SENS_PWR_CMD.activate();
MPS_PCL_FCV1_CO_CMD.activate();
MPS_PCL_FCV2_CO_CMD.activate();
MPS_PCL_FCV3_CO_CMD.activate();
MPS_PCL_PNEU_OP_CMD.inhibit();
MPS_PCL_LV10_OP_CMD.inhibit();
MPS_PCL_BLOWDOWN_OP_CMD.activate();

Record Message 'MPS LH2 CL initialized for Load' to <PAGE-A>;

return;

3.2.6.2.2. setCLModeGo()

3.2.6.2.2.1. Descriptive Text

This method will set up the Control Logic to support Terminal Count Operations.

3.2.6.2.2.2. Sequence

// setCLModeGo()

MPS_RCL_GHe_Bottle_Press.setCLOverrideOff();
MPS_RCL_GHe_Bottle_Press.inhibit();
MPS_RCL_GHe_Bottle_Press.alter(LO,4550);
MPS_RCL_GHe_Bottle_Press.activate();

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

MPS_RCL_Pneu_Reg_Press.setCLOverrideOn();
MPS_RCL_Pneu_Reg_Press.inhibit();
MPS_RCL_E1_Reg_Press.setCLOverrideOn();
MPS_RCL_E1_Reg_Press.inhibit();
MPS_RCL_E2_Reg_Press.setCLOverrideOn();
MPS_RCL_E2_Reg_Press.inhibit();
MPS_RCL_E3_Reg_Press.setCLOverrideOn();
MPS_RCL_E3_Reg_Press.inhibit();
MPS_RCL_GHe_Bottle_Temp.setCLOverrideOn();
MPS_RCL_GHe_Bottle_Temp.inhibit();
MPS_RCL_LH2_Manifold_Press.setCLOverrideOn();
MPS_RCL_LH2_Manifold_Press.inhibit();
MPS_RCL_LH2_PV11_OP_Ind.setCLOverrideOn();
MPS_RCL_LH2_PV11_OP_Ind.inhibit();
MPS_RCL_LH2_PV12_OP_Ind.setCLOverrideOn();
MPS_RCL_LH2_PV12_OP_Ind.inhibit();
MPS_RCL_LH2_PV13_OP_Ind.setCLOverrideOn();
MPS_RCL_LH2_PV13_OP_Ind.inhibit();
MPS_RCL_LH2_PD3_OP_Ind.setCLOverrideOn();
MPS_RCL_LH2_PD3_OP_Ind.inhibit();
MPS_RCL_LH2_PD2_OP_Ind.setCLOverrideOn();
MPS_RCL_LH2_PD2_OP_Ind.inhibit();
MPS_PCL_PV11_CL_CMD.setCLOverrideOff();
MPS_PCL_PV11_OP_CMD.setCLOverrideOff();
MPS_PCL_PV12_CL_CMD.setCLOverrideOff();
MPS_PCL_PV12_OP_CMD.setCLOverrideOff();
MPS_PCL_PV13_OP_CMD.setCLOverrideOff();
MPS_PCL_PD2_CL_CMD.setCLOverrideOff();
MPS_PCL_PD2_OP_CMD.setCLOverrideOff();
MPS_PCL_PD2_LATCH_LK_CMD.setCLOverrideOff();
MPS_PCL_PD2_LATCH_UL_CMD.setCLOverrideOff();
MPS_PCL_PD3_CL_CMD.setCLOverrideOff();
MPS_PCL_PD3_OP_CMD.setCLOverrideOff();
MPS_PCL_PV8_CL_CMD.setCLOverrideOn();
MPS_PCL_LV74_OP_CMD.setCLOverrideOff();
MPS_PCL_LV75_OP_CMD.setCLOverrideOff();
MPS_PCL_RECIRC_OP_CMD.setCLOverrideOn();
MPS_PCL_RTLS_DUMP_OP_CMD.setCLOverrideOff();
MPS_PCL_PT_SENS_PWR_CMD.setCLOverrideOff();
MPS_PCL_FCV1_CO_CMD.setCLOverrideOff();
MPS_PCL_FCV2_CO_CMD.setCLOverrideOff();
MPS_PCL_FCV3_CO_CMD.setCLOverrideOff();
MPS_PCL_PNEU_OP_CMD.setCLOverrideOn();
MPS_PCL_LV10_OP_CMD.setCLOverrideOn();
MPS_PCL_BLOWDOWN_OP_CMD.setCLOverrideOff();

Record Message 'MPS LH2 CL initialized for Terminal Count' to <PAGE-A>;

return;

3.2.6.2.3. setCLModeTest()

3.2.6.2.3.1. Descriptive Text

This method will set up the Control Logic to support Systems Test Operations.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.2.6.2.3.2. Sequence

```
// setCLModeTest()


MPS_RCL_GHe_Bottle_Press.setCLOverrideOff();
MPS_RCL_GHe_Bottle_Press.inhibit();
MPS_RCL_GHe_Bottle_Press.alter(LO,4500);
MPS_RCL_GHe_Bottle_Press.activate();
MPS_RCL_Pneu_Reg_Press.setCLOverrideOff();
MPS_RCL_E1_Reg_Press.setCLOverrideOff();
MPS_RCL_E2_Reg_Press.setCLOverrideOff();
MPS_RCL_E3_Reg_Press.setCLOverrideOff();
MPS_RCL_GHe_Bottle_Temp.setCLOverrideOff();
MPS_RCL_GHe_Bottle_Temp.inhibit();
MPS_RCL_GHe_Bottle_Temp.alter(LO,190);
MPS_RCL_GHe_Bottle_Temp.activate();
MPS_RCL_LH2_Manifold_Press.setCLOverrideOff();
MPS_RCL_LH2_Manifold_Press.inhibit();
MPS_RCL_LH2_Manifold_Press.alter(LO,55);
MPS_RCL_LH2_Manifold_Press.activate();
MPS_RCL_LH2_PV11_OP_Ind.setCLOverrideOn();
MPS_RCL_LH2_PV11_OP_Ind.inhibit();
MPS_RCL_LH2_PV12_OP_Ind.setCLOverrideOn();
MPS_RCL_LH2_PV12_OP_Ind.inhibit();
MPS_RCL_LH2_PV13_OP_Ind.setCLOverrideOn();
MPS_RCL_LH2_PV13_OP_Ind.inhibit();
MPS_RCL_LH2_PD3_OP_Ind.setCLOverrideOn();
MPS_RCL_LH2_PD3_OP_Ind.inhibit();
MPS_RCL_LH2_PD2_OP_Ind.setCLOverrideOn();
MPS_RCL_LH2_PD2_OP_Ind.inhibit();
MPS_PCL_PV11_CL_CMD.setCLOverrideOn();
MPS_PCL_PV11_OP_CMD.setCLOverrideOn();
MPS_PCL_PV12_CL_CMD.setCLOverrideOn();
MPS_PCL_PV12_OP_CMD.setCLOverrideOn();
MPS_PCL_PV13_OP_CMD.setCLOverrideOn();
MPS_PCL_PD2_CL_CMD.setCLOverrideOff();
MPS_PCL_PD2_OP_CMD.setCLOverrideOff();
MPS_PCL_PD2_LATCH_LK_CMD.setCLOverrideOff();
MPS_PCL_PD2_LATCH_UL_CMD.setCLOverrideOff();
MPS_PCL_PD3_CL_CMD.setCLOverrideOn();
MPS_PCL_PD3_OP_CMD.setCLOverrideOn();
MPS_PCL_PV8_CL_CMD.setCLOverrideOff();
MPS_PCL_LV74_OP_CMD.setCLOverrideOff();
MPS_PCL_LV75_OP_CMD.setCLOverrideOff();
MPS_PCL_RECIRC_OP_CMD.setCLOverrideOff();
MPS_PCL_RTLS_DUMP_OP_CMD.setCLOverrideOn();
MPS_PCL_PT_SENS_PWR_CMD.setCLOverrideOn();
MPS_PCL_FCV1_CO_CMD.setCLOverrideOff();
MPS_PCL_FCV2_CO_CMD.setCLOverrideOff();
MPS_PCL_FCV3_CO_CMD.setCLOverrideOff();
MPS_PCL_PNEU_OP_CMD.setCLOverrideOff();
MPS_PCL_LV10_OP_CMD.setCLOverrideOff();
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
MPS_PCL_BLOWDOWN_OP_CMD.setCLOverrideOff();

Record Message 'MPS LH2 CL initialized for Test' to <PAGE-A>;

return;
```

3.2.6.2.4. determineState()

3.2.6.2.4.1. Descriptive Text

This method will determine the Control_Logic_Mode

3.2.6.2.4.2. Sequence

```
// determineState()



if ((NMPS_RCLO_GHe_TNK_P == OFF) &&
   (NMPS_RCLO_PNEU_REG_P == ON) &&
   (NMPS_RCLO_E1_REG_P == ON) &&
   (NMPS_RCLO_E2_REG_P == ON) &&
   (NMPS_RCLO_E3_REG_P == ON) &&
   (NMPS_RCLO_GHe_TNK_T == ON) &&
   (NMPS_RCLO_LH2_MAN_P == ON) &&
   (NMPS_RCLO_PV11_OP == OFF) &&
   (NMPS_RCLO_PV12_OP == OFF) &&
   (NMPS_RCLO_PV13_OP  == OFF) &&
   (NMPS_RCLO_PD3_OP == OFF) &&
   (NMPS_RCLO_PD2_OP == OFF) &&
   (NMPS_PCLO_PV11_CL == OFF) &&
   (NMPS_PCLO_PV11_OP == OFF) &&
   (NMPS_PCLO_PV12_CL == OFF) &&
   (NMPS_PCLO_PV12_OP == OFF) &&
   (NMPS_PCLO_PV13_OP == OFF) &&
   (NMPS_PCLO_PD2_CL == OFF) &&
   (NMPS_PCLO_PD2_OP == OFF) &&
   (NMPS_PCLO_PD2_LTC_LK == OFF) &&
   (NMPS_PCLO_PD2_LTC_UL == OFF) &&
   (NMPS_PCLO_PD3_CL == OFF) &&
   (NMPS_PCLO_PD3_OP == OFF) &&
   (NMPS_PCLO_PV8_CL == ON) &&
   (NMPS_PCLO_LV74_OP == OFF) &&
   (NMPS_PCLO_LV75_OP == OFF) &&
   (NMPS_PCLO_RECIRC_OP == ON) &&
   (NMPS_PCLO_RTLS_DP_OP == OFF) &&
   (NMPS_PCLO_PT_SNS_PWR == OFF) &&
   (NMPS_PCLO_FCV1_CO == OFF) &&
   (NMPS_PCLO_FCV2_CO == OFF) &&
   (NMPS_PCLO_FCV3_CO == OFF) &&
   (NMPS_PCLO_PNEU_OP == ON) &&
   (NMPS_PCLO_LV10_OP  == ON) &&
   (NMPS_PCLO_BLOWDN_OP == OFF))
    {
    Control_Logic_Mode = 'LOAD';
    }
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
else
   {
    if ((NMPS_RCLO_GHe_TNK_P == OFF) &&
      (NMPS_RCLO_PNEU_REG_P == ON) &&
      (NMPS_RCLO_E1_REG_P == ON) &&
      (NMPS_RCLO_E2_REG_P == ON) &&
      (NMPS_RCLO_E3_REG_P == ON) &&
      (NMPS_RCLO_GHe_TNK_T == ON) &&
      (NMPS_RCLO_LH2_MAN_P == ON) &&
      (NMPS_RCLO_PV11_OP == ON) &&
      (NMPS_RCLO_PV12_OP == ON) &&
      (NMPS_RCLO_PV13_OP == ON) &&
      (NMPS_RCLO_PD3_OP == ON) &&
      (NMPS_RCLO_PD2_OP == ON) &&
      (NMPS_PCLO_PV11_CL == OFF) &&
      (NMPS_PCLO_PV11_OP == OFF) &&
      (NMPS_PCLO_PV12_CL == OFF) &&
      (NMPS_PCLO_PV12_OP == OFF) &&
      (NMPS_PCLO_PV13_OP == OFF) &&
      (NMPS_PCLO_PD2_CL == OFF) &&
      (NMPS_PCLO_PD2_OP == OFF) &&
      (NMPS_PCLO_PD2_LTC_LK == OFF) &&
      (NMPS_PCLO_PD2_LTC_UL == OFF) &&
      (NMPS_PCLO_PD3_CL == OFF) &&
      (NMPS_PCLO_PD3_OP == OFF) &&
      (NMPS_PCLO_PV8_CL == ON) &&
      (NMPS_PCLO_LV74_OP == OFF) &&
      (NMPS_PCLO_LV75_OP == OFF) &&
      (NMPS_PCLO_RECIRC_OP == ON) &&
      (NMPS_PCLO_RTLS_DMP_OP == OFF) &&
      (NMPS_PCLO_PT_SNS_PWR == OFF) &&
      (NMPS_PCLO_FCV1_CO == OFF) &&
      (NMPS_PCLO_FCV2_CO == OFF) &&
      (NMPS_PCLO_FCV3_CO == OFF) &&
      (NMPS_PCLO_PNEU_OP == ON) &&
      (NMPS_PCLO_LV10_OP == ON) &&
      (NMPS_PCLO_BLOWDN_OP == OFF))
       {
       Control_Logic_Mode = "GO";
       }
     else
       {
       if ((NMPS_RCLO_GHe_TNK_P == OFF) &&
         (NMPS_RCLO_PNEU_REG_P == OFF) &&
         (NMPS_RCLO_E1_REG_P == OFF) &&
         (NMPS_RCLO_E2_REG_P == OFF) &&
         (NMPS_RCLO_E3_REG_P == OFF) &&
         (NMPS_RCLO_GHe_TNK_T == OFF) &&
         (NMPS_RCLO_LH2_MAN_P == OFF) &&
         (NMPS_RCLO_PV11_OP == ON) &&
         (NMPS_RCLO_PV12_OP == ON) &&
         (NMPS_RCLO_PV13_OP == ON) &&
         (NMPS_RCLO_PD3 == ON) &&
         (NMPS_RCLO_PD2 == ON) &&
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
        (NMPS_PCLO_PV11_CL == ON) &&
        (NMPS_PCLO_PV11_OP == ON) &&
        (NMPS_PCLO_PV12_CL == ON) &&
        (NMPS_PCLO_PV12_OP == ON) &&
        (NMPS_PCLO_PV13_OP == ON) &&
        (NMPS_PCLO_PD2_CL == OFF) &&
        (NMPS_PCLO_PD2_OP == OFF) &&
        (NMPS_PCLO_PD2_LTC_LK == OFF) &&
        (NMPS_PCLO_PD2_LTC_UL == OFF) &&
        (NMPS_PCLO_PD3_CL == ON) &&
        (NMPS_PCLO_PD3_OP == ON) &&
        (NMPS_PCLO_PV8_CL == OFF) &&
        (NMPS_PCLO_LV74_OP == OFF) &&
        (NMPS_PCLO_LV75_OP == OFF) &&
        (NMPS_PCLO_RECIRC_OP == OFF) &&
        (NMPS_PCLO_RTLS_DMP_OP == ON) &&
        (NMPS_PCLO_PT_SNS_PWR == ON) &&
        (NMPS_PCLO_FCV1_CO == OFF) &&
        (NMPS_PCLO_FCV2_CO == OFF) &&
        (NMPS_PCLO_FCV3_CO == OFF) &&
        (NMPS_PCLO_PNEU_OP == OFF) &&
        (NMPS_PCLO_LV10_OP == OFF) &&
        (NMPS_PCLO_BLOWDN_OP == OFF))
         {
         Control_Logic_Mode = 'TEST';
         }
      else
         {
         Control_Logic_Mode = 'OTHER';
         }
      }
  }
```

return;

3.2.6.2.5. getState()

3.2.6.2.5.1. Descriptive Text

This method will return the Control_Logic_Mode

3.2.6.2.5.2. Sequence

```
// getState()

return Control_Logic_Mode;
```

3.2.6.2.6. getCLNotification()

3.2.6.2.6.1. Descriptive Text

This method will return the CLNotificationFlag

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.2.6.2.6.2. Sequence

```
// getCLNotification()

return CLNotificationFlag;
```

3.2.6.2.7. determineCLNotification()

3.2.6.2.7.1. Descriptive Text

This method will set the CLNotificationFlag to ON if any of the Aggregated CLNotificationFlags are ON Else it will be set to OFF.

3.2.6.2.7.2. Sequence

```
// determineNotification()



if ((NMPS_RCLN_GHe_TNK_P == ON) OR
   (NMPS_RCLN_PNEU_REG_P == ON) OR
   (NMPS_RCLN_E1_REG_P == ON) OR
   (NMPS_RCLN_E2_REG_P == ON) OR
   (NMPS_RCLN_E3_REG_P == ON) OR
   (NMPS_RCLN_GHe_TNK_T == ON) OR
   (NMPS_RCLN_LH2_MAN_P == ON) OR
   (NMPS_RCLN_PV11_OP == ON) OR
   (NMPS_RCLN_PV12_OP == ON) OR
   (NMPS_RCLN_PV13_OP  == ON) OR
   (NMPS_RCLN_PD3_OP == ON) OR
   (NMPS_RCLN_PD2_OP == ON) OR
   (NMPS_PCLN_PV11_CL == ON) OR
   (NMPS_PCLN_PV11_OP == ON) OR
   (NMPS_PCLN_PV12_CL == ON) OR
   (NMPS_PCLN_PV12_OP == ON) OR
   (NMPS_PCLN_PV13_OP == ON) OR
   (NMPS_PCLN_PD2_CL == ON) OR
   (NMPS_PCLN_PD2_OP == ON) OR
   (NMPS_PCLN_PD2_LTC_LK == ON) OR
   (NMPS_PCLN_PD2_LTC_UL == ON) OR
   (NMPS_PCLN_PD3_CL == ON) OR
   (NMPS_PCLN_PD3_OP == ON) OR
   (NMPS_PCLN_PV8_CL == ON) OR
   (NMPS_PCLN_LV74_OP == ON) OR
   (NMPS_PCLN_LV75_OP == ON) OR
   (NMPS_PCLN_RECIRC_OP == ON) OR
   (NMPS_PCLN_RTLS_DP_OP == ON) OR
   (NMPS_PCLN_PT_SNS_PWR == ON) OR
   (NMPS_PCLN_FCV1_CO == ON) OR
   (NMPS_PCLN_FCV2_CO == ON) OR
   (NMPS_PCLN_FCV3_CO == ON) OR
   (NMPS_PCLN_PNEU_OP == ON) OR
   (NMPS_PCLN_LV10_OP  == ON) OR
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
  (NMPS_PCLN_BLOWDN_OP == ON))
   {
   CLNotification = ON;
   }
else
   {
   CLNotification = OFF;
   }
```

3.2.6.2.8. activateLoadingRCL()

3.2.6.2.8.1. Descriptive Text

This method will activate the LH2 Loading Reactive Control Logic.

3.2.6.2.8.2. Sequence

```
// activateLoadingRCL()

MPS_RCL_LH2_PV11_OP_Ind.activate();
MPS_RCL_LH2_PV12_OP_Ind.activate();
MPS_RCL_LH2_PV13_OP_Ind.activate();
MPS_RCL_LH2_PD3_OP_Ind.activate();
MPS_RCL_LH2_PD2_OP_Ind.activate();

Record Message 'MPS LH2 Loading RCL activated' to <PAGE-A>;

return;
```

3.2.6.2.9. inhibitLoadingRCL()

3.2.6.2.9.1. Descriptive Text

This method will inhibit the LH2 Loading Reactive Control Logic.

3.2.6.2.9.2. Sequence

```
// inhibitLoadingRCL()

MPS_RCL_LH2_PV11_OP_Ind.inhibit();
MPS_RCL_LH2_PV12_OP_Ind.inhibit();
MPS_RCL_LH2_PV13_OP_Ind.inhibit();
MPS_RCL_LH2_PD3_OP_Ind.inhibit();
MPS_RCL_LH2_PD2_OP_Ind.inhibit();

Record Message 'MPS LH2 Loading RCL inhibit' to <PAGE-A>;

return;
```

3.2.6.2.10. resetCLNotification()

3.2.6.2.10.1. Descriptive Text

This method will set all CL notification flags off by running the resetCLNotification() method for all associated PCL/RCL sequences.

3.2.6.2.10.2. Sequence

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

// resetCLNotification()

MPS_RCL_GHe_Bottle_Press.resetCLNotification();
MPS_RCL_Pneu_Reg_Press.resetCLNotification();
MPS_RCL_E1_Reg_Press.resetCLNotification();
MPS_RCL_E2_Reg_Press.resetCLNotification();
MPS_RCL_E3_Reg_Press.resetCLNotification();
MPS_RCL_GHe_Bottle_Temp.resetCLNotification();
MPS_RCL_LH2_Manifold_Press.resetCLNotification();
MPS_RCL_LH2_PV11_OP_Ind.resetCLNotification();
MPS_RCL_LH2_PV12_OP_Ind.resetCLNotification();
MPS_RCL_LH2_PV13_OP_Ind.resetCLNotification();
MPS_RCL_LH2_PD3_OP_Ind.resetCLNotification();
MPS_RCL_LH2_PD2_OP_Ind.resetCLNotification();
MPS_PCL_PV11_CL_CMD.resetCLNotification();
MPS_PCL_PV11_OP_CMD.resetCLNotification();
MPS_PCL_PV12_CL_CMD.resetCLNotification();
MPS_PCL_PV12_OP_CMD.resetCLNotification();
MPS_PCL_PV13_OP_CMD.resetCLNotification();
MPS_PCL_PD2_CL_CMD.resetCLNotification();
MPS_PCL_PD2_OP_CMD.resetCLNotification();
MPS_PCL_PD2_LATCH_LK_CMD.resetCLNotification();
MPS_PCL_PD2_LATCH_UL_CMD.resetCLNotification();
MPS_PCL_PD3_CL_CMD.resetCLNotification();
MPS_PCL_PD3_OP_CMD.resetCLNotification();
MPS_PCL_PV8_CL_CMD.resetCLNotification();
MPS_PCL_LV74_OP_CMD.resetCLNotification();
MPS_PCL_LV75_OP_CMD.resetCLNotification();
MPS_PCL_RECIRC_OP_CMD.resetCLNotification();
MPS_PCL_RTLS_DUMP_OP_CMD.resetCLNotification();
MPS_PCL_PT_SENS_PWR_CMD.resetCLNotification();
MPS_PCL_FCV1_CO_CMD.resetCLNotification();
MPS_PCL_FCV2_CO_CMD.resetCLNotification();
MPS_PCL_FCV3_CO_CMD.resetCLNotification();
MPS_PCL_PNEU_OP_CMD.resetCLNotification();
MPS_PCL_LV10_OP_CMD.resetCLNotification();
MPS_PCL_BLOWDOWN_OP_CMD.resetCLNotification();

Record Message 'MPS LH2 CL Notification Reset' to <PAGE-A>;

return;

3.2.6.2.11. reportPCLBlock()

3.2.6.2.11.1. Descriptive Text

This method will report a PCL Block message based on the associated CLNotificationFlag

3.2.6.2.11.2. Effectivity

Updates based on change data of the associated CLNotificationFlags

3.2.6.2.11.3. Sequence

// reportPCLBlock()

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
if  (NMPS_PCLN_PV11_CL == ON)
   {
   Record Message 'MPS_PCL_PV11_CL_CMD  - PCL BLOCK ' to MPS_RCL message destination;
   }

if  (NMPS_PCLN_PV11_OP == ON)
   {
   Record Message 'MPS_PCL_PV11_OP_CMD  - PCL BLOCK ' to MPS_RCL message destination;
   }

if  (NMPS_PCLN_PV12_CL == ON)
   {
   Record Message 'MPS_PCL_PV12_CL_CMD  - PCL BLOCK ' to MPS_RCL message destination;
   }

if  (NMPS_PCLN_PV12_OP == ON)
   {
   Record Message 'MPS_PCL_PV12_OP_CMD  - PCL BLOCK ' to MPS_RCL message destination;
   }

if  (NMPS_PCLN_PV13_OP == ON)
   {
   Record Message 'MPS_PCL_PV13_OP_CMD  - PCL BLOCK ' to MPS_RCL message destination;
   }

if  (NMPS_PCLN_PD2_CL == ON)
   {
   Record Message 'MPS_PCL_PD2_CL_CMD  - PCL BLOCK ' to MPS_RCL message destination;
   }

if  (NMPS_PCLN_PD2_OP == ON)
   {
   Record Message 'MPS_PCL_PD2_OP_CMD  - PCL BLOCK ' to MPS_RCL message destination;
   }

if  (NMPS_PCLN_PD2_LTC_LK == ON)
   {
   Record Message 'MPS_PCL_PD2_LATCH_LK_CMD  - PCL BLOCK ' to MPS_RCL message destination;
   }

if  (NMPS_PCLN_PD2_LTC_UL == ON)
   {
   Record Message 'MPS_PCL_PD2_LATCH_UL_CMD  - PCL BLOCK ' to MPS_RCL message destination;
   }

if  (NMPS_PCLN_PD3_CL == ON)
   {
   Record Message 'MPS_PCL_PD3_CL_CMD  - PCL BLOCK ' to MPS_RCL message destination;
   }

if  (NMPS_PCLN_PD3_OP == ON)
   {
   Record Message 'MPS_PCL_PD3_OP_CMD  - PCL BLOCK ' to MPS_RCL message destination;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
   }

if   (NMPS_PCLN_PV8_CL == ON)
   {
   Record Message 'MPS_PCL_PV8_CL_CMD  - PCL BLOCK ' to MPS_RCL message destination;
   }

if   (NMPS_PCLN_LV74_OP == ON)
   {
   Record Message 'MPS_PCL_LV74_OP_CMD  - PCL BLOCK ' to MPS_RCL message destination;
   }

if   (NMPS_PCLN_LV75_OP == ON)
   {
   Record Message 'MPS_PCL_LV75_OP_CMD  - PCL BLOCK ' to MPS_RCL message destination;
   }

if   (NMPS_PCLN_RECIRC_OP == ON)
   {
   Record Message 'MPS_PCL_RECIRC_OP_CMD  - PCL BLOCK ' to MPS_RCL message destination;
   }

if   (NMPS_PCLN_RTLS_DP_OP == ON)
   {
   Record Message 'MPS_PCL_RTLS_DUMP_OP_CMD  - PCL BLOCK ' to MPS_RCL message destination;
   }

if   (NMPS_PCLN_PT_SNS_PWR == ON)
   {
   Record Message 'MPS_PCL_PT_SENS_PWR_CMD  - PCL BLOCK ' to MPS_RCL message destination;
   }

if   (NMPS_PCLN_FCV1_CO == ON)
   {
   Record Message 'MPS_PCL_FCV1_CO_CMD  - PCL BLOCK ' to MPS_RCL message destination;
   }

if   (NMPS_PCLN_FCV2_CO == ON)
   {
   Record Message 'MPS_PCL_FCV2_CO_CMD  - PCL BLOCK ' to MPS_RCL message destination;
   }

if   (NMPS_PCLN_FCV3_CO == ON)
   {
   Record Message 'MPS_PCL_FCV3_OP_CMD  - PCL BLOCK ' to MPS_RCL message destination;
   }

if   (NMPS_PCLN_PNEU_OP == ON)
   {
   Record Message 'MPS_PCL_PNEU_OP_CMD  - PCL BLOCK ' to MPS_RCL message destination;
   }

if   (NMPS_PCLN_LV10_OP  == ON)
   {
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
   Record Message 'MPS_PCL_LV10_OP_CMD  - PCL BLOCK ' to MPS_RCL message destination;
   }

if   (NMPS_PCLN_BLOWDN_OP == ON)
   {
   Record Message 'MPS_PCL_BLOWDOWN_OP_CMD  - PCL BLOCK ' to MPS_RCL message destination;
   }

return;
```

3.2.7. MPS_LO2_Control_Logic_Manager

3.2.7.1. Description

This Class is an aggregation of all the associated MPS_LO2 Prerequisite Control Logic and Reactive Control Logic. The class will contain the attributes and methods necessary to provide status and housekeeping of the Control Logic Subsystem.

3.2.7.1.1. Instance Index

Instance	Other Users
MPS_LO2_Control_Logic_Manager	

3.2.7.1.2. Class Diagram

<insert class diagram here>

3.2.7.1.3. Attributes and Aggregations

Attribute	Data Type	Initial Value	Publish / Persistence	Client Access	Description
CLNotificationFlag	PD_ON_OFF		PUB		
Control_Logic_Mode	MPS_CL_Mode		PUB		
MPS_RCL_LO2_Manifold_Press	MPS_RCL_LO2_Manifold_Press				
MPS_RCL_LO2_PV9_OP_Ind	MPS_RCL_LO2_PV9_OP_Ind				
MPS_RCL_LO2_PV10_OP_Ind	MPS_RCL_LO2_PV10_OP_Ind				
MPS_RCL_LO2_PD1_OP_Ind	MPS_RCL_LO2_PD1_OP_Ind				
MPS_PCL_PV9_CL_CMD	MPS_PCL_PV9_CL_CMD				
MPS_PCL_PV9_OP_CMD	MPS_PCL_PV9_OP_CMD				
MPS_PCL_PV10_CL_CMD	MPS_PCL_PV10_CL_CMD				
MPS_PCL_PV10_OP_CMD	MPS_PCL_PV10_OP_CMD				
MPS_PCL_PD1_CL_CMD	MPS_PCL_PD1_CL_CMD				

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

MPS_PCL_PD1_OP_CMD	MPS_PCL_PD1_OP_CMD				
MPS_PCL_PD1_LATCH_LK_CMD	MPS_PCL_PD1_LATCH_LK_CMD				
MPS_PCL_PD1_LATCH_UL_CMD	MPS_PCL_PD1_LATCH_UL_CMD				
MPS_PCL_PV19_CL_CMD	MPS_PCL_Pv19_CL_CMD				
MPS_PCL_PV7_CL_CMD	MPS_PCL_PV7_CL_CMD				
MPS_PCL_LO2_PREVALVES_CL_CMD	MPS_PCL_LO2_PREVALVES_CL_CMD				
MPS_PCL_LO2_PREVALVES_OP_CMD	MPS_PCL_LO2_PREVALVES_OP_CMD				
MPS_PCL_LO2_POGO_CL_CMD	MPS_PCL_POGO_CL_CMD				

3.2.7.1.4. Operations Index

Operation	**Description**
setCLModeReadyForLoad()	This method will set up the Control Logic to support Propellant Loading Operations.
setCLModeTest()	This method will set up the Control Logic to support Test Operations.
setCLModeTermCount()	This method will set up the Control Logic to support Terminal Count Operations.
getState()	This method will return the Control_Logic_Mode
determineState()	This method will determine the Control_Logic_Mode.
getCLNotification()	This method will return the CLNotificationFlag
determineCLNotification()	This method will set the CLNotificationFlag to ON if any of the Aggregated CLNotificationFlags are ON Else it will be set to OFF.
activateLoadingRCL()	This method will activate the LO2 Loading Reactive Control Logic.
inhibitLoadingRCL()	This method will inhibit the LO2 Loading Reactive Control Logic.
resetCLNotification()	This method will set all CL notification flags off by running the resetCLNotification() method for all associated PCL/RCL sequences.
reportPCLBlock()	This method will report a PCL Block message based on the associated CLNotificationFlag

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.2.7.1.5. Associations

Association	Type	Class	Description

3.2.7.2. Operations

3.2.7.2.1. setCLModeReadyForLoad()

3.2.7.2.1.1. Descriptive Text

This method will set up the Control Logic to support Propellant Loading Operations.

3.2.7.2.1.2. Sequence

// setCLModeReadyForLoad()

MPS_RCL_LO2_Manifold_Press.inhibit();
MPS_RCL_LO2_PV9_OP_Ind.inhibit();
MPS_RCL_LO2_PV10_OP_Ind.activate();
MPS_RCL_LO2_PD1_OP_Ind.activate();
MPS_PCL_PV9_CL_CMD.activate();
MPS_PCL_PV9_OP_CMD.activate();
MPS_PCL_PV10_CL_CMD.activate();
MPS_PCL_PV10_OP_CMD.activate();
MPS_PCL_PD1_CL_CMD.activate();
MPS_PCL_PD1_OP_CMD.activate();
MPS_PCL_PD1_LATCH_LK_CMD.activate();
MPS_PCL_PD1_LATCH_UL_CMD.activate();
MPS_PCL_PV19_CL_CMD.activate();
MPS_PCL_PV7_CL_CMD.inhibit();
MPS_PCL_LO2_PREVALVES_CL_CMD.activate();
MPS_PCL_LO2_PREVALVES_OP_CMD.activate();
MPS_PCL_LO2_POGO_CL_CMD.activate();

resetCLNotification();

Record Message 'MPS LO2 CL initialized READY_FOR_LOAD' to <PAGE-A>;

return;

3.2.7.2.2. setCLModeTermCount()

3.2.7.2.2.1. Descriptive Text

This method will set up the Control Logic to support Terminal Count Operations.

3.2.7.2.2.2. Sequence

// setCLModeTermCount()

MPS_RCL_LO2_Manifold_Press.inhibit();
MPS_RCL_LO2_PV9_OP_Ind.inhibit();
MPS_RCL_LO2_PV10_OP_Ind.inhibit();

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
MPS_RCL_LO2_PD1_OP_Ind.inhibit();
MPS_PCL_PV9_CL_CMD.activate();
MPS_PCL_PV9_OP_CMD.activate();
MPS_PCL_PV10_CL_CMD.activate();
MPS_PCL_PV10_OP_CMD.activate();
MPS_PCL_PD1_CL_CMD.activate();
MPS_PCL_PD1_OP_CMD.activate();
MPS_PCL_PD1_LATCH_LK_CMD.activate();
MPS_PCL_PD1_LATCH_UL_CMD.activate();
MPS_PCL_PV19_CL_CMD.activate();
MPS_PCL_PV7_CL_CMD.inhibit();
MPS_PCL_LO2_PREVALVES_CL_CMD.inhibit();
MPS_PCL_LO2_PREVALVES_OP_CMD.inhibit();
MPS_PCL_LO2_POGO_CL_CMD.inhibit();

resetCLNotification();

Record Message 'MPS LO2 CL initialized for Terminal Count' to <PAGE-A>;

return;
```

3.2.7.2.3. setCLModeTest()

3.2.7.2.3.1. Descriptive Text

This method will set up the Control Logic to support Systems Test Operations.

3.2.7.2.3.2. Sequence

```
// setCLModeTest()



MPS_RCL_LO2_Manifold_Press.inhibit();
MPS_RCL_LO2_Manifold_Press.alter(LO,70);
MPS_RCL_LO2_Manifold_Press.activate();
MPS_RCL_LO2_PV9_OP_Ind.inhibit();
MPS_RCL_LO2_PV10_OP_Ind.inhibit();
MPS_RCL_LO2_PD1_OP_Ind.inhibit();
MPS_PCL_PV9_CL_CMD.inhibit();
MPS_PCL_PV9_OP_CMD.inhibit();
MPS_PCL_PV10_CL_CMD.inhibit();
MPS_PCL_PV10_OP_CMD.inhibit();
MPS_PCL_PD1_CL_CMD.activate();
MPS_PCL_PD1_OP_CMD.activate();
MPS_PCL_PD1_LATCH_LK_CMD.activate();
MPS_PCL_PD1_LATCH_UL_CMD.activate();
MPS_PCL_PV19_CL_CMD.inhibit();
MPS_PCL_PV7_CL_CMD.activate();
MPS_PCL_LO2_PREVALVES_CL_CMD.inhibit();
MPS_PCL_LO2_PREVALVES_OP_CMD.inhibit();
MPS_PCL_LO2_POGO_CL_CMD.inhibit();

resetCLNotification();
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

Record Message 'MPS LO2 CL initialized for Test' to <PAGE-A>;

return;

3.2.7.2.4. determineState()

3.2.7.2.4.1. Descriptive Text

This method will determine the Control_Logic_Mode

3.2.7.2.4.2. Sequence

```
// determineState()



if ((NMPS_RCLS_LO2_MAN_P == INHIBITED) &&
  (NMPS_RCLS_PV9_OP == INHIBITED) &&
  (NMPS_RCLS_PV10_OP == ACTIVE) &&
  (NMPS_RCLS_PD1_OP == ACTIVE) &&
  (NMPS_PCLS_PV9_CL == ACTIVE) &&
  (NMPS_PCLS_PV9_OP == ACTIVE) &&
  (NMPS_PCLS_PV10_CL == ACTIVE) &&
  (NMPS_PCLS_PV10_OP == ACTIVE) &&
  (NMPS_PCLS_PD1_CL == ACTIVE) &&
  (NMPS_PCLS_PD1_OP == ACTIVE) &&
  (NMPS_PCLS_PD1_LTC_LK == ACTIVE) &&
  (NMPS_PCLS_PD1_LTC_UL == ACTIVE) &&
  (NMPS_PCLS_PV19_CL == ACTIVE) &&
  (NMPS_PCLS_PV7_CL == INHIBITED) &&
  (NMPS_PCLS_LO2_PVL_CL == ACTIVE) &&
  (NMPS_PCLS_LO2_PVL_OP == ACTIVE) &&
  (NMPS_PCLS_POGO_CL == ACTIVE))
   {
   Control_Logic_Mode = 'READY_FOR_LOAD';
   }
else
   {
   if ((NMPS_RCLS_LO2_MAN_P == INHIBITED) &&
      (NMPS_RCLS_PV9_OP == ACTIVE) &&
      (NMPS_RCLS_PV10_OP == ACTIVE) &&
      (NMPS_RCLS_PD1_OP == ACTIVE) &&
      (NMPS_PCLS_PV9_CL == ACTIVE) &&
      (NMPS_PCLS_PV9_OP == ACTIVE) &&
      (NMPS_PCLS_PV10_CL == ACTIVE) &&
      (NMPS_PCLS_PV10_OP == ACTIVE) &&
      (NMPS_PCLS_PD1_CL == ACTIVE) &&
      (NMPS_PCLS_PD1_OP == ACTIVE) &&
      (NMPS_PCLS_PD1_LTC_LK == ACTIVE) &&
      (NMPS_PCLS_PD1_LTC_UL == ACTIVE) &&
      (NMPS_PCLS_PV19_CL == ACTIVE) &&
      (NMPS_PCLS_PV7_CL == INHIBITED) &&
      (NMPS_PCLS_LO2_PVL_CL == ACTIVE) &&
      (NMPS_PCLS_LO2_PVL_OP == ACTIVE) &&
      (NMPS_PCLS_POGO_CL == ACTIVE))
       {
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
    Control_Logic_Mode = 'LOAD';
    }
  else
    {
   if ((NMPS_RCLS_LO2_MAN_P == INHIBITED) &&
      (NMPS_RCLS_PV9_OP == INHIBITED) &&
      (NMPS_RCLS_PV10_OP == INHIBITED) &&
      (NMPS_RCLS_PD1_OP == INHIBITED) &&
      (NMPS_PCLS_PV9_CL == ACTIVE) &&
      (NMPS_PCLS_PV9_OP == ACTIVE) &&
      (NMPS_PCLS_PV10_CL == ACTIVE) &&
      (NMPS_PCLS_PV10_OP == ACTIVE) &&
      (NMPS_PCLS_PD1_CL == ACTIVE) &&
      (NMPS_PCLS_PD1_OP == ACTIVE) &&
      (NMPS_PCLS_PD1_LTC_LK == ACTIVE) &&
      (NMPS_PCLS_PD1_LTC_UL == ACTIVE) &&
      (NMPS_PCLS_PV19_CL == ACTIVE) &&
      (NMPS_PCLS_PV7_CL == INHIBITED) &&
      (NMPS_PCLS_LO2_PVL_CL == INHIBITED) &&
      (NMPS_PCLS_LO2_PVL_OP == INHIBITED) &&
      (NMPS_PCLS_POGO_CL == INHIBITED))
       {
       Control_Logic_Mode = 'TERM_COUNT';
       }
   else
       {
      if ((NMPS_RCLS_LO2_MAN_P == ACTIVE) &&
        (NMPS_RCLS_PV9_OP == INHIBITED) &&
        (NMPS_RCLS_PV10_OP == INHIBITED) &&
        (NMPS_RCLS_PD1_OP == INHIBITED) &&
        (NMPS_PCLS_PV9_CL == INHIBITED) &&
        (NMPS_PCLS_PV9_OP == INHIBITED) &&
        (NMPS_PCLS_PV10_CL == INHIBITED) &&
        (NMPS_PCLS_PV10_OP == INHIBITED) &&
        (NMPS_PCLS_PD1_CL == ACTIVE) &&
        (NMPS_PCLS_PD1_OP == ACTIVE) &&
        (NMPS_PCLS_PD1_LTC_LK == ACTIVE) &&
        (NMPS_PCLS_PD1_LTC_UL == ACTIVE) &&
        (NMPS_PCLS_PV19_CL == INHIBITED) &&
        (NMPS_PCLS_PV7_CL == ACTIVE) &&
        (NMPS_PCLS_LO2_PVL_CL == INHIBITED) &&
        (NMPS_PCLS_LO2_PVL_OP == INHIBITED) &&
        (NMPS_PCLS_POGO_CL == INHIBITED))
          {
          Control_Logic_Mode = 'TEST';
          }
       else
          {
          Control_Logic_Mode = 'OTHER';
          }
       }
    }
  }
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

return;

3.2.7.2.5. getState()

3.2.7.2.5.1. Descriptive Text

This method will return the Control_Logic_Mode

3.2.7.2.5.2. Sequence

```
// getState()

return Control_Logic_Mode;
```

3.2.7.2.6. getCLNotification()

3.2.7.2.6.1. Descriptive Text

This method will return the CLNotificationFlag

3.2.7.2.6.2. Sequence

```
// getCLNotification()

return CLNotificationFlag;
```

3.2.7.2.7. determineCLNotification()

3.2.7.2.7.1. Descriptive Text

This method will set the CLNotificationFlag to ON if any of the Aggregated CLNotificationFlags are ON Else it will be set to OFF.

3.2.7.2.7.2. Sequence

```
// determineCLNotification()



if ((NMPS_RCLN_LO2_MAN_P == ON) OR
  (NMPS_RCLN_PV9_OP == ON) OR
  (NMPS_RCLN_PV10_OP == ON) OR
  (NMPS_RCLN_PD1_OP == ON) OR
  (NMPS_PCLN_PV9_CL == ON) OR
  (NMPS_PCLN_PV9_OP == ON) OR
  (NMPS_PCLN_PV10_CL == ON) OR
  (NMPS_PCLN_PV10_OP == ON) OR
  (NMPS_PCLN_PD1_CL == ON) OR
  (NMPS_PCLN_PD1_OP == ON) OR
  (NMPS_PCLN_PD1_LTC_LK == ON) OR
  (NMPS_PCLN_PD1_LTC_UL == ON) OR
  (NMPS_PCLN_PV19_CL == ON) OR
  (NMPS_PCLN_PV7_CL == ON) OR
  (NMPS_PCLN_LO2_PVL_CL == ON) OR
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
  (NMPS_PCLN_LO2_PVL_OP == ON) OR
  (NMPS_PCLN_POGO_CL == ON))
   {
   CLNotificationFlag = ON;
   }
else
   {
   CLNotificationFlag = OFF;
   }



return;
```

3.2.7.2.8. activateLoadingRCL()

3.2.7.2.8.1. Descriptive Text

This method will activate the LO2 Loading Reactive Control Logic.

3.2.7.2.8.2. Sequence

```
// activateLoadingRCL()


MPS_RCL_LO2_PV9_OP_Ind.activate();
MPS_RCL_LO2_PV10_OP_Ind.activate();
MPS_RCL_LO2_PD1_OP_Ind.activate();

Record Message "MPS LO2 Loading RCL activated" to <PAGE-A>;

return;
```

3.2.7.2.9. inhibitLoadingRCL()

3.2.7.2.9.1. Descriptive Text

This method will inhibit the LO2 Loading Reactive Control Logic.

3.2.7.2.9.2. Sequence

```
// inhibitLoadingRCL()


MPS_RCL_LO2_PV9_OP_Ind.inhibit();
MPS_RCL_LO2_PV10_OP_Ind.inhibit();
MPS_RCL_LO2_PD1_OP_Ind.inhibit();

Record Message "MPS LO2 Loading RCL inhibited" to <PAGE-A>;

return;
```

3.2.7.2.10. resetCLNotification()

3.2.7.2.10.1. Descriptive Text

This method will set all CL notification flags off by running the resetCLNotification() method for all associated PCL/RCL sequences.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.2.7.2.10.2. Sequence

```
// resetCLNotification()



MPS_RCL_LO2_Manifold_Press.resetCLNotification();
MPS_RCL_LO2_PV9_OP_Ind.resetCLNotification();
MPS_RCL_LO2_PV10_OP_Ind.resetCLNotification();
MPS_RCL_LO2_PD1_OP_Ind.resetCLNotification();
MPS_PCL_PV9_CL_CMD.resetCLNotification();
MPS_PCL_PV9_OP_CMD.resetCLNotification();
MPS_PCL_PV10_CL_CMD.resetCLNotification();
MPS_PCL_PV10_OP_CMD.resetCLNotification();
MPS_PCL_PD1_CL_CMD.resetCLNotification();
MPS_PCL_PD1_OP_CMD.resetCLNotification();
MPS_PCL_PD1_LATCH_LK_CMD.resetCLNotification();
MPS_PCL_PD1_LATCH_UL_CMD.resetCLNotification();
MPS_PCL_PV19_CL_CMD.resetCLNotification();
MPS_PCL_PV7_CL_CMD.resetCLNotification();
MPS_PCL_LO2_PREVALVES_CL_CMD.resetCLNotification();
MPS_PCL_LO2_PREVALVES_OP_CMD.resetCLNotification();
MPS_PCL_LO2_POGO_CL_CMD.resetCLNotification();


Record Message 'MPS LO2 CL Notification Reset' to <PAGE-A>;

return;
```

3.2.7.2.11. reportPCLBlock()

3.2.7.2.11.1. Descriptive Text

This method will report a PCL Block message based on the associated CLNotificationFlag

3.2.7.2.11.2. Effectivity

Updates based on change data of the associated CLNotificationFlags

3.2.7.2.11.3. Sequence

```
// reportPCLBlock()



if  (NMPS_PCLN_PV9_CL == ON)
   {
   Record Message 'MPS_PCL_PV9_CL_CMD  - PCL BLOCK ' to MPS_RCL message destination;
   }

if   (NMPS_PCLN_PV9_OP == ON)
   {
   Record Message 'MPS_PCL_PV9_OP_CMD  - PCL BLOCK ' to MPS_RCL message destination;
   }

if   (NMPS_PCLN_PV10_CL == ON)
   {
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
   Record Message 'MPS_PCL_PV10_CL_CMD  - PCL BLOCK ' to MPS_RCL message destination;
   }

if   (NMPS_PCLN_PV10_OP == ON)
   {
   Record Message 'MPS_PCL_PV10_OP_CMD  - PCL BLOCK ' to MPS_RCL message destination;
   }

if   (NMPS_PCLN_PD1_CL == ON)
   {
   Record Message 'MPS_PCL_PD1_CL_CMD  - PCL BLOCK ' to MPS_RCL message destination;
   }

if   (NMPS_PCLN_PD1_OP == ON)
   {
   Record Message 'MPS_PCL_PD1_OP_CMD  - PCL BLOCK ' to MPS_RCL message destination;
   }

if   (NMPS_PCLN_PD1_LTC_LK == ON)
   {
   Record Message 'MPS_PCL_PD1_LATCH_LK_CMD  - PCL BLOCK ' to MPS_RCL message destination;
   }

if   (NMPS_PCLN_PD1_LTC_UL == ON)
   {
   Record Message 'MPS_PCL_PD1_LATCH_UL_CMD  - PCL BLOCK ' to MPS_RCL message destination;
   }

if   (NMPS_PCLN_PV19_CL == ON)
   {
   Record Message 'MPS_PCL_PV19_CL_CMD  - PCL BLOCK ' to MPS_RCL message destination;
   }

if   (NMPS_PCLN_PV7_CL == ON)
   {
   Record Message 'MPS_PCL_PV7_CL_CMD  - PCL BLOCK ' to MPS_RCL message destination;
   }

if   (NMPS_PCLN_LO2_PVL_CL == ON)
   {
   Record Message 'MPS_PCL_LO2_PREVALVES_CL_CMD  - PCL BLOCK ' to MPS_RCL message
destination;
   }

if   (NMPS_PCLN_LO2_PVL_OP == ON)
   {
   Record Message 'MPS_PCL_LO2_PREVALVES_OP_CMD  - PCL BLOCK ' to MPS_RCL message
destination;
   }

if   (NMPS_PCLN_POGO_CL == ON)
   {
   Record Message 'MPS_PCL_LO2_POGO_CL_CMD  - PCL BLOCK ' to MPS_RCL message destination;
   }
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

return;

3.2.8. MPS_System

3.2.8.1. Description

The MPS_System class is comprised of MPS hydrogen, oxygen, and helium subsystems. This class calls subsystem methods during power-up and power-down operations required to configure constraint monitoring, control logic methods for vehicle test and check-out in the horizontal as well as the vertical. Auto cycle methods are also initiated from this class.

3.2.8.1.1. Instance Index

Instance	Other Users
MPS_MPS	

3.2.8.1.2. Class Diagram

<insert class diagram here>

3.2.8.1.3. Attributes and Aggregations

Attribute	Data Type	Initial Value	Publish / Persistence	Client Access	Description
MPS_LH2	MPS_Hydrogen_Subsystem				
MPS_LO2	MPS_Oxygen_Subsystem				
MPS_GHe	MPS_Helium_Subsystem				
MPS_ETOI	MPS_ET_OI_CNTL				
MPSCONFIGUREDFORTEST	PD_ON_OFF				
MPSTRANSBIASFLAG	PD_ON_OFF				

3.2.8.1.4. Operations Index

Operation	Description
configureForTest()	This method performs MPS LH2, GHE, and LOX system initial test configuration set-up, including establishing constraint monitor criteria and RCL activation. This method also applies transducer bias correction for LO2/LH2 manifold and feedline pressure transducers.
secureFromTest()	This method calls setPowerDownLimits to inhibit constraint monitoring of MPS LH2, GHe and LOX system parameters and inhibits appropriate RCL methods.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

removeTransducerBiasCorrection()	This method allows operator to reomve bias correction from applicable transducers.
autoCycleMPSValves()	This method call MPS LH2 and LO2 Subsystems to perform auto cycle on MPS propellant valves.

3.2.8.1.5. Associations

Association	Type	Class	Description

3.2.8.2. State Description

3.2.8.2.1. State Diagram

<insert state diagram here>

3.2.8.2.2. State Table

3.2.8.3. Operations

3.2.8.3.1. configureForTest()

3.2.8.3.1.1. Description

This method initiates MPS LH2, GHe and LOX systems constraint monitoring and initiates system RCL.

3.2.8.3.1.2. Sequence

```
configureForTest();
begin sequence;
{
send message to MPS and MPL Page B:
        "Configuring MPS Systems constraint criteria and RCL for test and check-out."
                to Page B in cyan

MPS_LH2.inhibitLH2Constraints();
MPS_GHe.inhibitGHeConstraints();
MPS_LO2.inhibitLOXConstraints();
MPS_ETOI.setPwrDownConstraints();

//set standard config

MPS_LH2.setLH2PowerUpConstraints();
MPS_GHe.setGHePowerUpConstraints();
MPS_LO2.setLO2PowerUpConstraints();

{
IF LO2configerror || LH2configerror == ON
        send message to Page B:
                An error occurred in determining bistable valve configuration during system initialization.  See
MPS
                Message display for details."  (in red)
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
end IF
}

//activate constraints
MPS_LH2.activateLH2Constraints();
MPS_GHe.activateGHeConstraints();
MPS_LO2.activateLOXConstraints();

//apply Transducer Bias Correction
MPS_LH2.applyTransducerBiasCorrection();
MPS_LO2.applyTransducerBiasCorrection();
set <NMPSTRANSBIASFLAG> ON;

//activate RCL
//LH2 CL manager does both GHe and LH2//

MPS_LH2_Control_Logic_Manager.setCLModeTest();
MPS_LO2_Control_Logic_Manager.setCLModeTest();

}
send message to Page B:
        "MPS Systems initial configuration complete." to Page B in cyan

//set MPS Test configuration pseudo ON

set <NMPSCONFIGUREDFORTEST> ON

end sequence;
```

3.2.8.3.2. secureFromTest()

3.2.8.3.2.1. Description

This method verifies no stray commands issued, inhibits constraint monitoring, and inhibits RCL for post test securing.

3.2.8.3.2.2. Sequence

```
secureFromTest();
{
prompt operator:
        "MPS constraint monitoring and control logic will be inhibited.  Do you wish to continue?

        If response == YES, Continue;
                ELSE END;
}
send message to MPS and MPL Page B:
        "Terminating MPS Systems constraint monitoring and RCL for system post test securing."
                to Page B in cyan
{
MPS_LH2.verifyCmdsOff();
MPS_GHe.verifyCmdsOff();
MPS_LO2.verifyCmdsOff();
MPS_ETOI.verifyCmdsOff()

//remove pressure transducer bias correction
set <NMPSTRANSBIASFLAG> OFF;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
//terminate constraint monitoring
MPS_LH2.inhibitLH2Constraints();
MPS_GHe.inhibitGHeConstraints();
MPS_LO2.inhibitLOXConstraints();
MPS_ETOI.inhibitConstraints();

MPS_LH2_Control_Logic_Manager.inhibitCL();
MPS_LO2_Control_Logic_Manager.inhibitCL();


send message to MPS and MPL Page B:
        "MPS Systems post test securing is complete."
                to Page B in cyan

//set MPS Test configuration pseudo OFF
set <NMPSCONFIGUREDFORTEST> OFF
}
```

3.2.8.3.3. removeTransducerBiasCorrection()

3.2.8.3.3.1. Description

This method provides user interface function to set pseudo for MPS transducer bias correction to OFF. MPS Displays will publish uncorrected FD value.

3.2.8.3.3.2. Sequence

```
removeTransducerBiasCorrection();
//sets pseudo OFF
{
set <NMPSTRANSBIASFLAG> OFF;
}
```

3.2.8.3.4. autoCycleMPSValves()

3.2.8.3.4.1. Description

This method calls LO2 and LH2 subsystems to perform propellant valve auto cycle methods.

3.2.8.3.4.2. Sequence

```
autoCycleMPSValves();
{
send message to MPS and MPL Page B:
        "Beginning MPS LH2 and LO2 System propellant valve auto cycle."
                to Page B in cyan
}
MPS_LH2.autocycleLH2();

MPS_LO2.autocycleLO2();

{
send message to MPS and MPL Page B:
        "MPS LH2 and LO2 System propellant valve auto cycle complete. See MPS messages for cycle times and
         detailed messages."
                to Page B in cyan
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

}
end sequence;

3.2.8.4. User Interfaces

3.2.8.5. Safety Requirements

3.2.9. MPS_Hydrogen_Subsystem

3.2.9.1. Description

The MPS_Hydrogen_Subsystem class is comprised of all MPS propellant valves and instrumentation which make up the orbiter MPS liquid hydrogen system. Included in this class are LH2 fill and drain valves, prevalves, 17" disconnect and latch, high point bleed valve, recirc isolation valves and pump speed sensors, manifold repressurization valves, flow control valves, and manifold and feedline temperature and pressure transducers. This class provides LO2 subsystem methods required for vehicle test and check-out in the horizontal as well as the vertical.

3.2.9.1.1. Instance Index

Instance	Other Users
MPS_LH2	

3.2.9.1.2. Class Diagram

<insert class diagram here>

3.2.9.1.3. Attributes and Aggregations

Attribute	Data Type	Initial Value	Publish / Persistence	Client Access	Description
MPS_PV4	MPS_Valve_ORB_LH2_Prevalve				
MPS_PV5	MPS_Valve_ORB_LH2_Prevalve				
MPS_PV6	MPS_Valve_ORB_LH2_Prevalve				
MPS_PV8	MPS_Valve_ORB_Relief_Isolation				
MPS_PV11	MPS_Valve_ORB_OutBoard_F&D				
MPS_PV12	MPS_Valve_ORB_InBoard_F&D				
MPS_PV13	MPS_Valve_ORB_Topping				
MPS_LV36	MPS_Recirc				
MPS_PV14	MPS_Valve_ORB_Recirc_Isolation				

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

MPS_PV15	MPS_Valve_ORB_Recirc_Isolation				
MPS_PV16	MPS_Valve_ORB_Recirc_Isolation				
MPS_PV17	MPS_Valve_ORB_RTLS_Dump				
MPS_PV18	MPS_Valve_ORB_RTLS_Dump				
MPS_PV22	MPS_Valve_ORB_LH2HiPt_Bleed				
MPS_LV42	MPS_SolValve_ORB_2way				
MPS_LV43	MPS_SolValve_ORB_2way				
MPS_LV52	MPS_SolValve_ORB_2way				
MPS_LV74	MPS_SolValve_ORB_RTLSPress				
MPS_LV75	MPS_SolValve_ORB_RTLSPress				
MPS_PD2_Assembly	MPS_ORB_17inch_Disconnect_Assembly				
MPS_PD2	MPS_Valve_ORB_17inch_Disc				
MPS_PD2_Latch	MPS_Valve_ORB_Latch				
MPS_PD3	MPS_Valve_ORB_4inchDisc				
MPS_MT14	MPS_ORB_Pressure_Sensor				
MPS_MT27	MPS_ORB_Pressure_Sensor				
MPS_MT43	MPS_ORB_Pressure_Sensor				
MPS_MT44	MPS_ORB_Pressure_Sensor				
MPS_MT54	MPS_ORB_Pressure_Sensor				
MPS_MT11	MPS_ORB_Pressure_Sensor				
MPS_MT24	MPS_ORB_Pressure_Sensor				
MPS_MT37	MPS_ORB_Pressure_Sensor				
MPS_MT47	MPS_ORB_Pressure_Sensor				
MPS_MT42	MPS_ORB_Temperature_Sensor				
MPS_MT12	MPS_ORB_Temperature_Sensor				

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

MPS_MT25	MPS_ORB_Temperature_Sensor				
MPS_MT38	MPS_ORB_Temperature_Sensor				
MPS_PP1	MPS_ORB_Recirc_Speed_Sensor				
MPS_PP2	MPS_ORB_Recirc_Speed_Sensor				
MPS_PP3	MPS_ORB_Recirc_Speed_Sensor				
MPS_E1FCV	MPS_SolValve_ORB_FCV				
MPS_E2FCV	MPS_SolValve_ORB_FCV				
MPS_E3FCV	MPS_SolValve_ORB_FCV				

3.2.9.1.4. Operations Index

Operation	**Description**
setLH2BiStableVlvConstraints()	This method determine MPS LH2 system bistable valve configuration for constraint monitoring during test and check-out.
setLH2PowerUpConstraints()	This method is called by configureForTest() method to establish MPS LH2 constraint criteria for test and check-out.
activateLH2Constraints()	This method activates constraint monitoring of MPS LH2 system parameters.
inhibitLH2Constraints()	This method inhibits constraint monitoring of MPS LH2 system parameters.
applyTransducerBiasCorrection();	This method calls LH2 system manifold and feedline pressure transducers to apply bias correction.
verifyCmdsOff()	This method verifies all LH2 system valve commands are off.
lH2autoCycle()	This method calls each LH2 system valve auto cycle method.

3.2.9.1.5. Associations

Association	**Type**	**Class**	**Description**
MPS_LH2_Control_Logic_Manager			

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.2.9.2. State Description

3.2.9.2.1. State Diagram

<insert state diagram here>

3.2.9.2.2. State Table

3.2.9.3. Operations

3.2.9.3.1. setLH2BiStableVlvConstraints()

3.2.9.3.1.1. Description

This method establishes MPS LH2 system configuration for constraint monitoring during test and check-out.

3.2.9.3.1.2. Sequence

establishPowerUpConfig();

set LH2configerror = OFF

//check bistable valve positions
{
//LH2 Prevalves
If MPS_PV4.getIndState = CLOSED
    MPS_PV4.setClosePosConstraints()
If MPS_PV4.getIndState = OPEN
    MPS_PV4.setOpenPosConstraints()
Else send message to Page A:
    "Unable to establish constraint monitoring criteria for PV4" (red)
    set LH2configerror = ON

If MPS_PV5.getIndState = CLOSED
    MPS_PV5.setClosePosConstraints()
If MPS_PV5.getIndState = OPEN
    MPS_PV5.setOpenPosConstraints()
Else send message to Page A:
    "Unable to establish constraint monitoring criteria for PV5" (red)
    set LH2configerror = ON

If MPS_PV6.getIndState = CLOSED
    MPS_PV6.setClosePosConstraints()
If MPS_PV6.getIndState = OPEN
    MPS_PV6.setOpenPosConstraints()
Else send message to Page A:
    "Unable to establish constraint monitoring criteria for PV6" (red)
    set LH2configerror = ON

//LH2 Fill and Drains

If MPS_PV11.getIndState = CLOSED
    MPS_PV11.setClosePosConstraints()
If MPS_PV11.getIndState = OPEN
    MPS_PV11.setOpenPosConstraints()
Else send message to Page A:
    "Unable to establish constraint monitoring criteria for PV11" (red)

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
        set LH2configerror = ON

If MPS_PV12.getIndState = CLOSED
        MPS_PV12.setClosePosConstraints()
If MPS_PV12.getIndState = OPEN
        MPS_PV12.setOpenPosConstraints()
Else send message to Page A:
        "Unable to establish constraint monitoring criteria for PV12" (red)
        set LH2configerror = ON


return LH2configerror

end sequence
}
```

3.2.9.3.2. setLH2PowerUpConstraints()

3.2.9.3.2.1. Description

This sequnce calls each instance of the MPS LH2 System Classes to set their nominal power-up constraint criteria.

3.2.9.3.2.2. Sequence

```
//set standby configuration constraint criteria//
//bistable

setLH2BistableVlvConstraints();

//monostables

MPS_PV8.setOpenConstraints()
MPS_PV13.setCloseConstraints()
MPS_Recirc.setCloseConstraints()
MPS_PV14.setCloseConstraints()
MPS_PV15.setCloseConstraints()
MPS_PV16.setCloseConstraints()
MPS_PV17.setCloseConstraints()
MPS_PV18.setCloseConstraints()
MPS_PV22.setCloseConstraints()
MPS_LV41.setCloseConstraints()
MPS_LV42.setCloseConstraints()
MPS_LV52.setCloseConstraints()
MPS_LV74.setCloseConstraints()
MPS_LV75.setCloseConstraints()

//disconnect and latch
MPS_PD2.setCloseIndConstraints()
MPS_PD2_Latch.setUnlockIndConstraints()
MPS_PD3.setCloseIndConstraints()
```

3.2.9.3.3. activateLH2Constraints()

3.2.9.3.3.1. Description

This method initiates MPS LH2 system constraint monitoring based on initial system configuration.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.2.9.3.3.2. Sequence

begin sequence;

MPS_PD2.activateConstraints()

MPS_PD2_Latch.activateConstraints()

MPS_PD3.activateConstraints()

MPS_PV4.activateConstraints()

MPS_PV5.activateConstraints()

MPS_PV6.activateConstraints()

MPS_PV8.activateConstraints()

MPS_PV11.activateConstraints()

MPS_PV12.activateConstraints()

MPS_PV13.activateConstraints()

MPS_Recirc.activateConstraints()

MPS_PV14.activateConstraints()

MPS_PV15.activateConstraints()

MPS_PV16.activateConstraints()

MPS_PV17.activateConstraints()

MPS_PV18.activateConstraints()

MPS_PV22.activateConstraints()

MPS_LV42.activateConstraints()

MPS_LV43.activateConstraints()

MPS_LV52.activateConstraints()

MPS_LV74.activateConstraints()

MPS_LV75.activateConstraints()

MPS_E1FCV.activateConstraints()

MPS_E2FCV.activateConstraints()

MPS_E3FCV.activateConstraints()

MPS_MT42.activateConstraintss()

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

MPS_MT12.activateConstraintss()

MPS_MT25.activateConstraintss()

MPS_MT38.activateConstraintss()

MPS_MT14.activateConstraintss()

MPS_MT27.activateConstraintss()

MPS_MT43.activateConstraintss()

MPS_MT44.activateConstraintss()

MPS_MT54.activateConstraintss()

MPS_MT11.activateConstraintss()

MPS_MT24.activateConstraintss()

MPS_MT37.activateConstraintss()

MPS_MT47.activateConstraintss()

end sequence;

3.2.9.3.4. inhibitLH2Constraints()

3.2.9.3.4.1. Description

This method inhibits LH2 system constraint monitoring for system test securing.

3.2.9.3.4.2. Sequence

//Inhibit Constraint Monitoring

MPS_PV4.inhibitConstraints()

MPS_PV5.inhibitConstraints()

MPS_PV6.inhibitConstraints()

MPS_PV8.inhibitConstraints()

MPS_PV11.inhibitConstraints()

MPS_PV12.inhibitConstraints()

MPS_PV13.inhibitConstraints()

MPS_Recirc.inhibitConstraints()

MPS_PV14.inhibitConstraints()

MPS_PV15.inhibitConstraints()

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

MPS_PV16.inhibitConstraints()

MPS_PV17.inhibitConstraints()

MPS_PV18.inhibitConstraints()

MPS_PV22.inhibitConstraints()

MPS_PD2.inhibitConstraints()

MPS_PD2_Latch.inhibitConstraints()

MPS_PD3.inhibitConstraints()

MPS_LV42.inhibitConstraints()

MPS_LV43.inhibitConstraints()

MPS_LV52.inhibitConstraints()

MPS_LV74.inhibitConstraints()

MPS_LV75.inhibitConstraints()

MPS_E1FCV.inhibitConstraints()

MPS_E2FCV.inhibitConstraints()

MPS_E3FCV.inhibitConstraints()

MPS_MT42.inhibitConstraintss()

MPS_MT12.inhibitConstraintss()

MPS_MT25.inhibitConstraintss()

MPS_MT38.inhibitConstraintss()

MPS_MT14.inhibitConstraintss()

MPS_MT27.inhibitConstraintss()

MPS_MT43.inhibitConstraintss()

MPS_MT44.inhibitConstraintss()

MPS_MT54.inhibitConstraintss()

MPS_MT11.inhibitConstraintss()

MPS_MT24.inhibitConstraintss()

MPS_MT37.inhibitConstraintss()

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

MPS_MT47.inhibitConstraintss()

end sequence;

3.2.9.3.5. verifyCmdsOff()

3.2.9.3.5.1. Description

This operation calls each valve instance of the LH2 system to perform verifyCmdsOff() method.

3.2.9.3.5.2. Sequence

verifyCmdsOff();
{
MPS_PV4.verifyCmdsOff()

MPS_PV5.verifyCmdsOff()

MPS_PV6.verifyCmdsOff()

MPS_PV8.verifyCmdsOff()

MPS_PV11.verifyCmdsOff()

MPS_PV12.verifyCmdsOff()

MPS_PV13.verifyCmdsOff()

MPS_Recirc.verifyCmdsOff()

MPS_PV17.verifyCmdsOff()

MPS_PV18.verifyCmdsOff()

MPS_PV22.verifyCmdsOff()

MPS_PD2.verifyCmdsOff()

MPS_PD2_Latch.verifyCmdsOff()

MPS_PD3.verifyCmdsOff()

MPS_LV42.verifyCmdsOff()

MPS_LV43.verifyCmdsOff()

MPS_LV52.verifyCmdsOff()

MPS_LV74.verifyCmdsOff()

MPS_LV75.verifyCmdsOff()

MPS_E1FCV.verifyCmdsOff()

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

MPS_E2FCV.verifyCmdsOff()

MPS_E3FCV.verifyCmdsOff()
}
end sequence;

3.2.9.3.6. applyTransducerBiasCorrection();

3.2.9.3.6.1. Description

This method calls LH2 system pressure transducers to apply bias correction.

3.2.9.3.6.2. Sequence

begin sequence;

MPS_MT11.applyBiasCorrection();

MPS_MT14.applyBiasCorrection();

MPS_MT24.applyBiasCorrection();

MPS_MT27.applyBiasCorrection();

MPS_MT37.applyBiasCorrection();

MPS_MT43.applyBiasCorrection();

//MT44 intentionally not called

MPS_MT47.applyBiasCorrection();

MPS_MT54.applyBiasCorrection();

end sequence;

3.2.9.3.7. autoCycleLH2()

3.2.9.3.7.1. Description This method calls each LH2 system cryogenic valve to perform its auto cycle method.

3.2.9.3.7.2. Sequence

set LH2autocycleError == OFF
set autoValveCheckoutError == OFF

//Sequentially perform the following:

MPS_PV4.autocycle()
If autoValveCheckoutError == ON,

MPS_PV5.autocycle()

MPS_PV6.autocycle()

MPS_PV8.autocycle()

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

MPS_PV11.autocycle()

MPS_PV12.autocycle()

MPS_PV13.autocycle()

MPS_Recirc.autocycle()

MPS_PV17.autocycle()

MPS_PV18.autocycle()

MPS_PV22.autocycle()

3.2.9.4. User Interfaces

3.2.9.5. Safety Requirements

3.2.10. MPS_Oxygen_Subsystem

3.2.10.1. Description

The MPS_Oxygen_Subsystem class is comprised of all MPS propellant valves and instrumentation which make up the orbiter MPS liquid oxygen system. Included in this class are LO2 fill and drain valves, prevalves, 17" disconnect and latch, bleed valve, pogo valves, manifold repressurization valves, SSME LO2 cut-off sensors, and manifold and feedline temperature and pressure transducers. This class provides LO2 subsystem methods required for vehicle test and check-out in the horizontal as well as the vertical.

3.2.10.1.1. Instance Index

Instance	Other Users
MPS_LOX	

3.2.10.1.2. Class Diagram

<insert class diagram here>

3.2.10.1.3. Attributes and Aggregations

Attribute	Data Type	Initial Value	Publish / Persistence	Client Access	Description
MPS_PV1	MPS_Valve_ORB_LO2_Prevalve				
MPS_PV2	MPS_Valve_ORB_LO2_Prevalve				
MPS_PV3	MPS_Valve_ORB_LO2_Prevalve				
MPS_PV7	MPS_Valve_ORB_Relief_Isolation				
MPS_PV9	MPS_Valve_ORB_OutBoard_F&D				

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

MPS_PV10	MPS_Valve_ORB_InBoard_F&D				
MPS_PV19	MPS_Valve_ORB_OvBrd_Bleed				
MPS_PV20	MPS_Valve_ORB_Pogos				
MPS_PV20	MPS_Valve_ORB_Pogos				
MPS_LV40	MPS_SolValve_ORB_2way				
MPS_LV41	MPS_SolValve_ORB_2way				
MPS_PD1_Assembly	MPS_ORB_17inch_Disconnect_ Assembly				
MPS_PD1	MPS_Valve_ORB_17inch_Disc				
MPS_PD1_Latch	MPS_Valve_ORB_Latch				
MPS_MT5	MPS_ORB_Pressure_Sensor				
MPS_MT18	MPS_ORB_Pressure_Sensor				
MPS_MT31	MPS_ORB_Pressure_Sensor				
MPS_MT49	MPS_ORB_Pressure_Sensor				
MPS_MT50	MPS_ORB_Pressure_Sensor				
MPS_MT53	MPS_ORB_Pressure_Sensor				
MPS_MT6	MPS_ORB_Temperature_Sensor				
MPS_MT19	MPS_ORB_Temperature_Sensor				
MPS_MT32	MPS_ORB_Temperature_Sensor				
MPS_MT48	MPS_ORB_Temperature_Sensor				
MPS_MT512	MPS_ORB_Temperature_Sensor				
MPS_MT13	MPS_ORB_Temperature_Sensor				
MPS_MT26	MPS_ORB_Temperature_Sensor				
MPS_MT39	MPS_ORB_Temperature_Sensor				
MPS_LO2_Ec01	MPS_ORB_LOX_ECO_Sensor				
MPS_LO2_Ec01	MPS_ORB_LOX_ECO_Sensor				
MPS_LO2_Ec01	MPS_ORB_LOX_ECO_Sensor				

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.2.10.1.4. Operations Index

Operation	Description
setLO2BiStableVlvConstraints()	This method determines MPS LO2 system bistable valve configuration for constraint monitoring during test and check-out.
setLO2PowerUpConstraints()	This method is called by configureForTest() method to establish MPS LO2 constraint criteria for test and check-out.
activateLO2Constraints()	This method activates constraint monitoring of MPS LO2 system parameters.
inhibitLO2Constraints()	This method inhibits constraint monitoring of MPS LO2 system parameters.
verifyCmdsOff()	This method verifies LO2 system valve commands are off.
applyTransducerBiasCorrection();	This method applies ambient bias correction to LO2 feedline and manifold pressure transducers.
autoCycleLO2()	This method calls each LO2 system valve auto cyle method.

3.2.10.1.5. Associations

Association	Type	Class	Description
MPS_LO2_Control_Logic_Manager			

3.2.10.2. State Description

3.2.10.2.1. State Diagram

<insert state diagram here>

3.2.10.2.2. State Table

3.2.10.3. Operations

3.2.10.3.1. set LO2BiStableVlvConstraints()

3.2.10.3.1.1. Description

This method establishes MPS LO2 system configuration for constraint monitoring during test and check-out.

3.2.10.3.1.2. Sequence

establishPowerUpConfig();

set LO2configerror = OFF

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
//check bistable valve positions
{
//LO2 Prevalves
If MPS_PV1.getIndState = CLOSED
        MPS_PV1.setClosePosConstraints()
If MPS_PV1.getIndState = OPEN
        MPS_PV1.setOpenPosConstraints()
Else send message to Page A:
        "Unable to establish constraint monitoring criteria for PV1" (red)
        set LO2configerror = ON

If MPS_PV2.getIndState = CLOSED
        MPS_PV2.setClosePosConstraints()
If MPS_PV2.getIndState = OPEN
        MPS_PV2.setOpenPosConstraints()
Else send message to Page A:
        "Unable to establish constraint monitoring criteria for PV2" (red)
        set LO2configerror = ON

If MPS_PV3.getIndState = CLOSED
        MPS_PV3.setClosePosConstraints()
If MPS_PV3.getIndState = OPEN
        MPS_PV3.setOpenPosConstraints()
Else send message to Page A:
        "Unable to establish constraint monitoring criteria for PV3" (red)
        set LO2configerror = ON

//LO2 Fill and Drains

If MPS_PV9.getIndState = CLOSED
        MPS_PV9.setClosePosConstraints()
If MPS_PV9.getIndState = OPEN
        MPS_PV9.setOpenPosConstraints()
Else send message to Page A:
        "Unable to establish constraint monitoring criteria for PV9" (red)
        set LO2configerror = ON

If MPS_PV10.getIndState = CLOSED
        MPS_PV10.setClosePosConstraints()
If MPS_PV10.getIndState = OPEN
        MPS_PV10.setOpenPosConstraints()
Else send message to Page A:
        "Unable to establish constraint monitoring criteria for PV10" (red)
        set LO2configerror = ON


return LO2configerror

end sequence
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.2.10.3.2. setLO2PowerUpConstraints()

3.2.10.3.2.1. Description

This sequnce establishes nominal MPS LO2 system power-up constraint criteria.

3.2.10.3.2.2. Sequence

//set standby configuration constraint criteria//
//bistable

setLO2BistableVlvConstraints();

//monostables

MPS_PV7.setOpenConstraints()
MPS_PV19.setOpenConstraints()
MPS_PV20.setOpenConstraints()
MPS_PV21.setOpenConstraints()
MPS_LV40.setCloseConstraints()
MPS_LV41.setCloseConstraints()

//disconnect and latch
MPS_PD1.setCloseIndConstraints()
MPS_PD1_Latch.setUnlockIndConstraints()

3.2.10.3.3. activateLO2Constraints()

3.2.10.3.3.1. Description

This method initiates MPS LO2 system constraint monitoring based on initial system configuration.

3.2.10.3.3.2. Sequence

begin sequence;

MPS_PD1.activateConstraints()

MPS_PD1_Latch.activateConstraints()

MPS_PV1.activateConstraints()

MPS_PV2.activateConstraints()

MPS_PV3.activateConstraints()

MPS_PV7.activateConstraints()

MPS_PV9.activateConstraints()

MPS_PV10.activateConstraints()

MPS_PV19.activateConstraints()

MPS_PV20.activateConstraints()

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

MPS_PV21.activateConstraints()

MPS_LV40.activateConstraints()

MPS_LV41.activateConstraints()

MPS_MT6.activateConstraintss()

MPS_MT19.activateConstraintss()

MPS_MT32.activateConstraintss()

MPS_MT48.activateConstraintss()

MPS_MT512.activateConstraintss()

MPS_MT13.activateConstraintss()

MPS_MT26.activateConstraintss()

MPS_MT39.activateConstraintss()

MPS_MT5.activateConstraintss()

MPS_MT18.activateConstraintss()

MPS_MT31.activateConstraintss()

MPS_MT49.activateConstraintss()

MPS_MT50.activateConstraintss()

MPS_MT53.activateConstraintss()

end sequence;

3.2.10.3.4. inhibitLO2Constraints()

3.2.10.3.4.1. Description

This method inhibits LO2 system constraint monitoring for system test securing.

3.2.10.3.4.2. Sequence

//Inhibit Constraint Monitoring

MPS_PV1.inhibitConstraints()

MPS_PV2.inhibitConstraints()

MPS_PV3.inhibitConstraints()

MPS_PV7.inhibitConstraints()

MPS_PV9.inhibitConstraints()

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

MPS_PV10.inhibitConstraints()

MPS_PV19.inhibitConstraints()

MPS_PV20.inhibitConstraints()

MPS_PV21.inhibitConstraints()

MPS_PD1.inhibitConstraints()

MPS_PD1_Latch.inhibitConstraints()

MPS_LV40.inhibitConstraints()

MPS_LV41.inhibitConstraints()

MPS_MT6.inhibitConstraintss()

MPS_MT19.inhibitConstraintss()

MPS_MT32.inhibitConstraintss()

MPS_MT48.inhibitConstraintss()

MPS_MT512.inhibitConstraintss()

MPS_MT13.inhibitConstraintss()

MPS_MT26.inhibitConstraintss()

MPS_MT39.inhibitConstraintss()

MPS_MT5.inhibitConstraintss()

MPS_MT18.inhibitConstraintss()

MPS_MT31.inhibitConstraintss()

MPS_MT49.inhibitConstraintss()

MPS_MT50.inhibitConstraintss()

MPS_MT53.inhibitConstraintss()

end sequence;

3.2.10.3.5. verifyCmdsOff()

3.2.10.3.5.1. Description

This operation calls each valve instance of the LO2 system to perform verifyCmdsOff() method.

3.2.10.3.5.2. Sequence

verifyCmdsOff();

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

{
MPS_PV1.verifyCmdsOff()

MPS_PV2.verifyCmdsOff()

MPS_PV3.verifyCmdsOff()

MPS_PV7.verifyCmdsOff()

MPS_PV9.verifyCmdsOff()

MPS_PV10.verifyCmdsOff()

MPS_PV19.verifyCmdsOff()

MPS_PV20.verifyCmdsOff()

MPS_PV21.verifyCmdsOff()

MPS_PD1.verifyCmdsOff()

MPS_PD1_Latch.verifyCmdsOff()

MPS_LV40.verifyCmdsOff()

MPS_LV41.verifyCmdsOff()

}
end sequence;

3.2.10.3.6. applyTransducerBiasCorrection();

3.2.10.3.6.1. Description

This method calls LO2 system pressure transducers to apply bias correction.

3.2.10.3.6.2. Sequence

begin sequence;

MPS_MT5.applyBiasCorrection();

MPS_MT18.applyBiasCorrection();

MPS_MT31.applyBiasCorrection();

MPS_MT49.applyBiasCorrection();

//MT50 intentionally not called

MPS_MT53.applyBiasCorrection();

end sequence;

3.2.10.3.7. autoCycleLO2()

set LO2autocycleError == OFF

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

set autoValveCheckoutError == OFF

//Sequentially perform the following:

MPS_PV1.autocycle()
If autoValveCheckoutError == ON,

MPS_PV2.autocycle()

MPS_PV3.autocycle()

MPS_PV7.autocycle()

MPS_PV9.autocycle()

MPS_PV10.autocycle()

MPS_PV19.autocycle()

MPS_PV20.autocycle()

MPS_PV21.autocycle()

3.2.10.4. User Interfaces

3.2.10.5. Safety Requirements

3.2.11. MPS_Helium_Subsystem

3.2.11.1. Description

The MPS_Helium_Subsystem class is comprised of all MPS solenoid valves and instrumentation which make up the orbiter MPS/SSME high pressure and pneumatic helium system. Included in this class are regulator isolation valves, crossover valve, blowdown valves, interconnect valves, temperature and pressure transducers. This class provides GHe subsystem methods required for vehicle test and check-out in the horizontal as well as the vertical.

3.2.11.1.1. Instance Index

Instance	Other Users
MPS_GHe	

3.2.11.1.2. Class Diagram

<insert class diagram here>

3.2.11.1.3. Attributes and Aggregations

Attribute	Data Type	Initial Value	Publish / Persistence	Client Access	Description
MPS_LV1	MPS_SolValve_ORB_SMERegIsoA				

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

MPS_LV2	MPS_SolValve_ORB_SMERegIsoB				
MPS_LV3	MPS_SolValve_ORB_SMERegIsoA				
MPS_LV4	MPS_SolValve_ORB_SMERegIsoB				
MPS_LV5	MPS_SolValve_ORB_SMERegIsoA				
MPS_LV6	MPS_SolValve_ORB_SMERegIsoB				
MPS_LV7	MPS_SolValve_ORB_MPSRegIso				
MPS_LV8	MPS_SolValve_ORB_MPSRegIso				
MPS_LV10	MPS_SolValve_ORB_2way				
MPS_LV26	MPS_SolValve_ORB_2way				
MPS_LV27	MPS_SolValve_ORB_2way				
MPS_LV59	MPS_SolValve_ORB_Interconnect				
MPS_LV60	MPS_SolValve_ORB_2way_IntOut				
MPS_LV61	MPS_SolValve_ORB_Interconnect				
MPS_LV62	MPS_SolValve_ORB_2way_IntOut				
MPS_LV63	MPS_SolValve_ORB_Interconnect				
MPS_LV64	MPS_SolValve_ORB_2way_IntOut				
MPS_MT8	MPS_ORB_Pressure_Sensor				
MPS_MT10	MPS_ORB_Pressure_Sensor				
MPS_MT21	MPS_ORB_Pressure_Sensor				
MPS_MT34	MPS_ORB_Pressure_Sensor				

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

MPS_MT55	MPS_ORB_Pressure_Sensor				
MPS_MT74	MPS_ORB_Pressure_Sensor				
MPS_MT75	MPS_ORB_Pressure_Sensor				
MPS_MT76	MPS_ORB_Pressure_Sensor				
MPS_MT518	MPS_ORB_Pressure_Sensor				
MPS_MT9	MPS_ORB_Temperature_Sensor				
MPS_MT22	MPS_ORB_Temperature_Sensor				
MPS_MT35	MPS_ORB_Temperature_Sensor				
MPS_MT56	MPS_ORB_Temperature_Sensor				
MPS_MT71	MPS_ORB_Temperature_Sensor				
MPS_MT72	MPS_ORB_Temperature_Sensor				
MPS_MT73	MPS_ORB_Temperature_Sensor				

3.2.11.1.4. Operations Index

Operation	**Description**
setGHePowerUpConstraints	This method establishes MPS helium system standby configuration constraint criteria.
activateGHeConstraints	This method activates constraint monitoring of MPS GHe system parameters.
inhibitGHeConstraints	This method inhibits constraint monitoring of MPS GHe system parameters.
verifyCmdsOff()	This method verifies GHe solenoid valve commands are off.

3.2.11.1.5. Associations

Association	**Type**	**Class**	**Description**

3.2.11.2. State Description

3.2.11.2.1. State Diagram

<insert state diagram here>

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.2.11.2.2. State Table

3.2.11.3. Operations

3.2.11.3.1. setGHePowerUpConstraints()

3.2.11.3.1.1. Description

This sequnce establishes nominal MPS GHe system power-up constraint criteria.

3.2.11.3.1.2. Sequence

setGHePowerUpConstraints();
{
MPS_LV1.setCloseConstraints()

MPS_LV2.setCloseConstraints()

MPS_LV3.setCloseConstraints()

MPS_LV4.setCloseConstraints()

MPS_LV5.setCloseConstraints()

MPS_LV6.setCloseConstraints()

MPS_LV7.setCloseConstraints()

MPS_LV8.setCloseConstraints()

MPS_LV10.setCloseConstraints()

MPS_LV26.setCloseConstraints()

MPS_LV27.setCloseConstraints()

MPS_LV59.setCloseConstraints()

MPS_LV60.setCloseConstraints()

MPS_LV61.setCloseConstraints()

MPS_LV62.setCloseConstraints()

MPS_LV63.setCloseConstraints()

3.2.11.3.2. activateGHeConstraints()

3.2.11.3.2.1. Description

This method initiates MPS LH2 system constraint monitoring based on initial system configuration.

3.2.11.3.2.2. Sequence

Activate Constraint Monitoring on the following:

MPS_LV1.activateConstraints()

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

MPS_LV2.activateConstraints()

MPS_LV3.activateConstraints()

MPS_LV4.activateConstraints()

MPS_LV5.activateConstraints()

MPS_LV6.activateConstraints()

MPS_LV7.activateConstraints()

MPS_LV8.activateConstraints()

MPS_LV10.activateConstraints()

MPS_LV26.activateConstraints()

MPS_LV27.activateConstraints()

MPS_LV59.activateConstraints()

MPS_LV60.activateConstraints()

MPS_LV61.activateConstraints()

MPS_LV62.activateConstraints()

MPS_LV63.activateConstraints()

MPS_MT9.activateConstraintss()

MPS_MT22.activateConstraintss()

MPS_MT35.activateConstraintss()

MPS_MT56.activateConstraintss()

MPS_MT71.activateConstraintss()

MPS_MT72.activateConstraintss()

MPS_MT73.activateConstraintss()

MPS_MT8.activateConstraintss()

MPS_MT10.activateConstraintss()

MPS_MT21.activateConstraintss()

MPS_MT34.activateConstraintss()

MPS_MT55.activateConstraintss()

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

MPS_MT74.activateConstraintss()

MPS_MT75.activateConstraintss()

MPS_MT76.activateConstraintss()

MPS_MT518.activateConstraintss()

3.2.11.3.3. inhibitGHeConstraints()

3.2.11.3.3.1. Description

This method inhibits MPS GHe system constraint monitoring.

3.2.11.3.3.2. Sequence

Inhibit Constraint Monitoring on the following:

MPS_LV1.inhibitConstraints()

MPS_LV2.inhibitConstraints()

MPS_LV3.inhibitConstraints()

MPS_LV4.inhibitConstraints()

MPS_LV5.inhibitConstraints()

MPS_LV6.inhibitConstraints()

MPS_LV7.inhibitConstraints()

MPS_LV8.inhibitConstraints()

MPS_LV10.inhibitConstraints()

MPS_LV26.inhibitConstraints()

MPS_LV27.inhibitConstraints()

MPS_LV59.inhibitConstraints()

MPS_LV60.inhibitConstraints()

MPS_LV61.inhibitConstraints()

MPS_LV62.inhibitConstraints()

MPS_LV63.inhibitConstraints()

MPS_MT9.inhibitConstraints()

MPS_MT22.inhibitConstraints()

MPS_MT35.inhibitConstraints()

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

MPS_MT56.inhibitConstraints()

MPS_MT71.inhibitConstraints()

MPS_MT72.inhibitConstraints()

MPS_MT73.inhibitConstraints()

MPS_MT8.inhibitConstraints()

MPS_MT10.inhibitConstraints()

MPS_MT21.inhibitConstraints()

MPS_MT34.inhibitConstraints()

MPS_MT55.inhibitConstraints()

MPS_MT74.inhibitConstraints()

MPS_MT75.inhibitConstraints()

MPS_MT76.inhibitConstraints()

MPS_MT518.inhibitConstraints()

3.2.11.3.4. verifyCmdsOff()

3.2.11.3.4.1. Description

This method calls each instance of the MPS GHe system to perform verifyCmdsOff() method.

3.2.11.3.4.2. Sequence

MPS_LV1.verifyCmdsOff()

MPS_LV2.verifyCmdsOff()

MPS_LV3.verifyCmdsOff()

MPS_LV4.verifyCmdsOff()

MPS_LV5.verifyCmdsOff()

MPS_LV6.verifyCmdsOff()

MPS_LV7.verifyCmdsOff()

MPS_LV8.verifyCmdsOff()

MPS_LV10.verifyCmdsOff()

MPS_LV26.verifyCmdsOff()

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

MPS_LV27.verifyCmdsOff()

MPS_LV59.verifyCmdsOff()

MPS_LV60.verifyCmdsOff()

MPS_LV61.verifyCmdsOff()

MPS_LV62.verifyCmdsOff()

MPS_LV63.verifyCmdsOff()

3.2.11.4. User Interfaces

3.2.11.5. Safety Requirements

3.3. REACTIVE CONTROL LOGIC CLASSES

3.3.1. MPS_RCL_GHe_Bottle_Press

3.3.1.1. Description

This class Contains the Reactive Control Logic that provides protection for the Orbiter MPS and SSME 1/2/3 Pneumatic Helum Bottle Pressures. Overpressure protection (default set at 4500 psia, limit increased to 4550 psia for loading operations) will terminate the Helium supply to the Orbiter by closing:
In the OPF:
MPS_A83128
MPS_A83129
MPS_A83130
MPS_A83131
MPS_A83132
MPS_A83133
At the VAB/PAD:
MPS_A74936
MPS_A74935
MPS_A74937
MPS_A74983
This class inherits from the class CAS_Control_Logic. Reference - LPS Program VCE03/04

3.3.1.1.1. Instance Index

Instance	Other Users
MPS_RCL_GHe_Bottle_Press	

3.3.1.1.2. Initiating Measurements

Measurement
<V41P1600A1>

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

<V41P1150C1>
<V41P1250C1>
<V41P1350C1>

3.3.1.1.3. Class Diagram

<insert class diagram here>

3.3.1.1.4. Operations Index

Operation	Description
run()	This method will perform the execution of the RCL sequence. Notification of the sequence execution will be performed.
determineState()	This method will determine the enumerated state of the RCL sequence based on the associated initiating measurements and will return either: ACTIVE INHIBITED
getState()	This method is inherited from the base class - CAS_Control_Logic
activate()	This method will activate RCL Notification for all unbypassed initiating measurements.
inhibit()	This method will inhibit RCL Notification for all initiating measurements.
alter(lo,hi)	This method will change the RCL Limit for all initiating measurements
statusRCL()	This method will status all the RCL initiating measurements and output the status (active/inhibited), limits (LO,HI) and also the CLNotificationFlag.
resetCLNotification()	This method is inherited from the base class - CAS_Control_Logic
assertRCL()	This method will initialize the RCL monitoring for each initiating measurement by accessing the assert() function of the Constraint Manager.
release()	This method will terminate the RCL monitoring for each initiating measurement by accessing the release() function of the Constraint Manager.

3.3.1.1.5. Associations

Association	Type	Class	Description

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.3.1.2. Operations

3.3.1.2.1. getState()

3.3.1.2.1.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.3.1.2.2. resetCLNotification()

3.3.1.2.2.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.3.1.2.3. run()

3.3.1.2.3.1. Descriptive Text

This method will perform the execution of the RCL sequence. Notification of the sequence execution will be performed.

3.3.1.2.3.2. Sequence

```
// BEGIN MPS_RCL_GHe_Bottle_Press Reactive Control Logic


// VERIFY MEASUREMENTS ARE OUT OF LIMITS

IF <V41P1600A1>.getValue <= 4500 AND
   <V41P1150C1>.getValue <= 4500 AND
   <V41P1250C1>.getValue <= 4500 AND
   <V41P1350C1>.getValue <= 4500 THEN
  {
    Send Message "<clSequenceID> executed but pressure currently is less than 4500 psia.'
            to MPS_RCL message destination;
    Return;
  }


// CHECK IF ORB IS IN OPF

IF NINTORBITERLOCATION == OPF THEN

  {


// A83128 CLOSE
  SET <GHEK2000E> ON;

// A83129 CLOSE
  SET <GHEK2010E> OFF;

// A83130 CLOSE
  SET <GHEK2020E> OFF;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
// A83131 CLOSE
   SET <GHEK2040E> ON;

// A83132 CLOSE
   SET <GHEK2050E> OFF;

// A83133 CLOSE
   SET <GHEK2060E> OFF;

    Send Message "<clSequenceID> executed, OPF safed'
            to MPS_RCL message destination;
  }



// CHECK IF ORB IS AT VAB OR PAD

IF NINTORBITERLOCATION == VAB OR
   NINTORBITERLOCATION == PAD THEN
  {

// A74936 CLOSE
   SET <GHEK1000E> ON;
   SET <GHEK1008E> OFF;
   SET <GHEK1100E> ON;
   SET <GHEK1108E> OFF;

// A75935 CLOSE
   SET <GHEK1040E> OFF;
   SET <GHEK1047E> ON;

// A75937 CLOSE
   SET <GHEK1160E> OFF;
   SET <GHEK1167E> ON;

// A74983 CLOSE
   SET <GHEK1090E> OFF;
   SET <GHEK1097E> ON;
   SET <GHEK1190E> OFF;
   SET <GHEK1197E> ON;

    Send Message "<clSequenceID> executed, MLP safed'
            to MPS_RCL message destination;
  }


// SET/SEND NOTIFICATION


Set <CLNotificationFlag> to ON;

}

Return;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

// END REACTIVE SEQUENCE

3.3.1.2.4. determineState()

3.3.1.2.4.1. Descriptive Text

This method will determine the enumerated state of the RCL sequence based on the associated initiating measurements and will return either: ACTIVE OR INHIBITED.

3.3.1.2.4.2. Sequence

IF (getConstraintInfo<MPS_V41P1600A1_RCL> = 'ACTIVE' OR
getConstraintInfo<MPS_V41P1150C1_RCL> = 'ACTIVE' OR
getConstraintInfo<MPS_V41P1250C1_RCL> = 'ACTIVE' OR
getConstraintInfo<MPS_V41P1350C1_RCL> = 'ACTIVE') THEN CLState == 'ACTIVE';

IF (getConstraintInfo<MPS_V41P1600A1_RCL> = 'INHIBITED' AND
getConstraintInfo<MPS_V41P1150C1_RCL> = 'INHIBITED' AND
getConstraintInfo<MPS_V41P1250C1_RCL> = 'INHIBITED' AND
getConstraintInfo<MPS_V41P1350C1_RCL> = 'INHIBITED') THEN CLState == 'INHIBITED';

3.3.1.2.5. activate()

3.3.1.2.5.1. Descriptive Text

This method will activate RCL notification for all unbypassed initiating measurements.

3.3.1.2.5.2. Sequence

IF <V41P1600A1>.BYP is OFF, activate<MPS_V41P1600A1_RCL>;
IF <V41P1150C1>.BYP is OFF, activate<MPS_V41P1150C1_RCL>;
IF <V41P1250C1>.BYP is OFF, activate<MPS_V41P1250C1_RCL>;
IF <V41P1350C1>.BYP is OFF, activate<MPS_V41P1350C1_RCL>;

3.3.1.2.6. inhibit()

3.3.1.2.6.1. Descriptive Text

This method will inhibit RCL notification for all initiating measurements.

3.3.1.2.6.2. Sequence

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

inhibit<MPS_V41P1600C1_RCL>;
inhibit<MPS_V41P1150C1_RCL>;
inhibit<MPS_V41P1250C1_RCL>;
inhibit<MPS_V41P1350C1_RCL>;

3.3.1.2.7. alter(lo,hi)

3.3.1.2.7.1. Descriptive Text

This method will change the RCL Limit for all initiating measurements

3.3.1.2.7.2. Sequence

alter(lo,hi,)<MPS_V41P1600C1_RCL>;
alter(lo,hi,)<MPS_V41P1150C1_RCL>;
alter(lo,hi,)<MPS_V41P1250C1_RCL>;
alter(lo,hi,)<MPS_V41P1350C1_RCL>;

3.3.1.2.8. statusRCL()

3.3.1.2.8.1. Descriptive Text

This method will status all the RCL initiating measurements and output the status (active/inhibited), limits (LO,HI) and also the CLNotificationFlag.

3.3.1.2.8.2. Sequence

Record Message '<clSequenceID> status is:'
'initiating measurements: ' <V41P1600A1>.get.description, <V41P1600A1>.get.value'
<V41P1150C1>.getdescription, <V41P1150C1>.getvalue,
<V41P1250C1>.getdescription, <V41P1250C1>.getvalue,
<V41P1350C1>.getdescription, <V41P1350C1>.getvalue;
'RCL Limits: ' getConstraintInfo<MPS_V41P1600A1_RCL>'
getConstraintInfo<MPS_V41P1150C1_RCL>,
getConstraintInfo<MPS_V41P1250C1_RCL>,
getConstraintInfo<MPS_V41P1350C1_RCL>;
'RCL Notification: ' <CLNotificationFlag>.getValue();
'RCL State: ' MPS_RCL_GHe_Bottle_Press::getState();

3.3.1.2.9. assertRCL()

3.3.1.2.9.1. Descriptive Text

This method will initialize the RCL monitoring for each initiating measurement by accessing the assert() function of the Constraint Manager.

3.3.1.2.9.2. Sequence

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

// Initialize Constraints

{

assertAtomicConstraint<V41P1600A1>(LO,4500,MPS_V41P1600A1_RCL) with callback pointer to
MPS_RCL_GHe_Bottle_Press::run();
assertAtomicConstraint<V41P1150C1>(LO,4500,MPS_V41P1150C1_RCL) with callback pointer to
MPS_RCL_GHe_Bottle_Press::run();
assertAtomicConstraint<V41P1250C1>(LO,4500,MPS_V41P1250C1_RCL) with callback pointer to
MPS_RCL_GHe_Bottle_Press::run();
assertAtomicConstraint<V41P1350C1>(LO,4500,MPS_V41P1350C1_RCL) with callback pointer to
MPS_RCL_GHe_Bottle_Press::run();

Record Message '<clSequenceID> Monitoring has been initialized' to <PAGE-A>;

}

3.3.1.2.10. release()

3.3.1.2.10.1. Descriptive Text

This method will terminate the RCL monitoring for each initiating measurement by accessing the release() function of the Constraint Manager.

3.3.1.2.10.2. Sequence

releaseAtomicConstraint<MPS_V41P1600A1_RCL>;
releaseAtomicConstraint<MPS_V41P1150C1_RCL>;
releaseAtomicConstraint<MPS_V41P1250C1_RCL>;
releaseAtomicConstraint<MPS_V41P1350C1_RCL>;

Record Message '<clSequenceID> Monitoring has been terminated' to <PAGE-A>;

3.3.2. MPS_RCL_Pneu_Reg_Press

3.3.2.1. Description

This class Contains the Reactive Control Logic that provides protection for the Orbiter MPS Pneumatic Helum Reg Pressure. Overpressure protection (default set at 800 psia) will close both LV7/LV8.
This class inherits from the class CAS_Control_Logic. Reference - LPS Program VCE05

3.3.2.1.1. Instance Index

Instance	Other Users
MPS_RCL_Pneu_Reg_Press	

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.3.2.1.2. Initiating Measurements

Measurement
<V41P1605A1>
<V41P1650A1>

3.3.2.1.3. Class Diagram

<insert class diagram here>

3.3.2.1.4. Operations Index

Operation	**Description**
run()	This method will perform the execution of the RCL sequence. Notification of the sequence execution will be performed.
determineState()	This method will determine the enumerated state of the RCL sequence based on the associated initiating measurements and will return either: ACTIVE INHIBITED
getState()	This method is inherited from the base class - CAS_Control_Logic
activate()	This method will activate RCL Notification for all unbypassed initiating measurements.
inhibit()	This method will inhibit RCL Notification for all initiating measurements.
alter(lo,hi)	This method will change the RCL Limit for all initiating measurements.
statusRCL()	This method will status all the RCL initiating measurements and output the status (active/inhibited), limits (LO,HI) and also the CLNotificationFlag.
resetCLNotification()	This method is inherited from the base class - CAS_Control_Logic
assertRCL()	This method will initialize the RCL monitoring for each initiating measurement by accessing the assert() function of the Constraint Manager.
release()	This method will terminate the RCL monitoring for each initiating measurement by accessing the release() function of the Constraint Manager.

3.3.2.1.5. Associations

Association	**Type**	**Class**	**Description**

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.3.2.2. Operations

3.3.2.2.1. getState()

3.3.2.2.1.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.3.2.2.2. resetCLNotification()

3.3.2.2.2.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.3.2.2.3. run()

3.3.2.2.3.1. Descriptive Text

This method will perform the execution of the RCL sequence. Notification of the sequence execution will be performed.

3.3.2.2.3.2. Sequence

```
// BEGIN MPS_RCL_Pneu_Reg_Press Reactive Control Logic


// VERIFY MEASUREMENTS ARE OUT OF LIMITS

IF <V41P1605A1>.getValue <= 800 AND
   <V41P1650A1>.getValue <= 800 THEN
  {
    Send Message "<clSequenceID> executed but pressure is currently less than 800 psia.'
                to MPS_RCL message destination;
    Return;
  }



// LV7 CLOSE
  SET <V41K1607XL> OFF;

// LV8 CLOSE
  SET <V41K1608XL> OFF;

Send Message "<clSequenceID> executed, LV7/LV8 closed'
                to MPS_RCL message destination;

  }
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

// SET/SEND NOTIFICATION

Set <CLNotificationFlag> to ON;

Return;

// END REACTIVE SEQUENCE

3.3.2.2.4. determineState()

3.3.2.2.4.1. Descriptive Text

This method will determine the enumerated state of the RCL sequence based on the associated initiating measurements and will return either: ACTIVE OR INHIBITED.

3.3.2.2.4.2. Sequence

IF (getConstraintInfo<MPS_V41P1605A1_RCL> = 'ACTIVE' OR
getConstraintInfo<MPS_V41P1650A1_RCL> = 'ACTIVE') THEN CLState == 'ACTIVE';

IF (getConstraintInfo<MPS_V41P1605A1_RCL> = 'INHIBITED' AND
getConstraintInfo<MPS_V41P1650A1_RCL> = 'INHIBITED') THEN CLState == 'INHIBITED';

3.3.2.2.5. activate()

3.3.2.2.5.1. Descriptive Text

This method will activate RCL notification for all unbypassed initiating measurements.

3.3.2.2.5.2. Sequence

IF <V41P1605A1>.BYP is OFF, activate<MPS_V41P1605A1_RCL>;
IF <V41P1650A1>.BYP is OFF, activate<MPS_V41P1650A1_RCL>;

3.3.2.2.6. inhibit()

3.3.2.2.6.1. Descriptive Text

This method will inhibit RCL notification for all initiating measurements.

3.3.2.2.6.2. Sequence

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

inhibit<MPS_V41P1605A1_RCL>;
inhibit<MPS_V41P1650A1_RCL>;

3.3.2.2.7. alter(lo,hi)

3.3.2.2.7.1. Descriptive Text

This method will change the RCL Limit for all initiating measurements

3.3.2.2.7.2. Sequence

alter(lo,hi)<MPS_V41P1605A1_RCL>;
alter(lo,hi)<MPS_V41P1650A1_RCL>;

3.3.2.2.8. statusRCL()

3.3.2.2.8.1. Descriptive Text

This method will status all the RCL initiating measurements and output the status (active/inhibited), limits (LO,HI) and also the RCL Override.

3.3.2.2.8.2. Sequence

Record Message ' <clSequenceID> status is:'
'initiating measurements: ' <V41P1605A1>.get.description, <V41P1605A1>.get.value()';
<V41P1650A1>.get.description, <V41P1650A1>.get.value()';
'RCL Limits: ' getConstraintInfo<MPS_V41P1605A1_RCL>;
getConstraintInfo<MPS_V41P1650A1_RCL>;
'RCL Notification: ' <CLNotificationFlag>.getValue();
'RCL State: ' MPS_RCL_Pneu_Reg_Press::getState();

3.3.2.2.9. assertRCL()

3.3.2.2.9.1. Descriptive Text

This method will initialize the RCL monitoring for each initiating measurement by accessing the assert() function of the Constraint Manager.

3.3.2.2.9.2. Sequence

// Initialize Constraints

{

assertAtomicConstraint<V41P1605A1>(LO,800,MPS_V41P1605A1_RCL)with callback pointer to
MPS_RCL_Pneu_Reg_Press::run();
assertAtomicConstraint<V41P1650A1>(LO,800,MPS_V41P1650A1_RCL)with callback pointer to
MPS_RCL_Pneu_Reg_Press::run();

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

Record Message '<clSequenceID> Monitoring has been initialized' to <PAGE-A>;

}

3.3.2.2.10. release()

3.3.2.2.10.1. Descriptive Text

This method will terminate the RCL monitoring for each initiating measurement by accessing the release() function of the Constraint Manager.

3.3.2.2.10.2. Sequence

releaseAtomicConstraint<MPS_V41P1605A1_RCL>;
releaseAtomicConstraint<MPS_V41P1650A1_RCL>;

Record Message '<clSequenceID> Monitoring has been terminated' to <PAGE-A>;

3.3.3. MPS_RCL_E1_Reg_Press

3.3.3.1. Description

This class Contains the Reactive Control Logic that provides protection for the Orbiter SSME#1 Pneumatic Helum Reg Pressure. Overpressure protection (default set at 800 psia) will close both LV1/LV2.
This class inherits from the class CAS_Control_Logic. Reference - LPS Program VCE06

3.3.3.1.1. Instance Index

Instance	Other Users
MPS_RCL_E1_Reg_Press	

3.3.3.1.2. Initiating Measurements

Measurement
<V41P1154A1>
<V41P1153A1>

3.3.3.1.3. Class Diagram

<insert class diagram here>

3.3.3.1.4. Operations Index

Operation	Description

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

run()	This method will perform the execution of the RCL sequence. Notification of the sequence execution will be performed.
determineState()	This method will determine the enumerated state of the RCL sequence based on the associated initiating measurements and will return either: ACTIVE INHIBITED
getState()	This method is inherited from the base class - CAS_Control_Logic
activate()	This method will activate RCL Notification for all unbypassed initiating measurements.
inhibit()	This method will inhibit RCL Notification for all initiating measurements.
alter(lo,hi)	This method will change the RCL Limit for all initiating measurements.
statusRCL()	This method will status all the RCL initiating measurements and output the status (active/inhibited), limits (LO,HI) and also the CLNotificationFlag .
resetCLNotification()	This method is inherited from the base class - CAS_Control_Logic
assertRCL()	This method will initialize the RCL monitoring for each initiating measurement by accessing the assert() function of the Constraint Manager.
release()	This method will terminate the RCL monitoring for each initiating measurement by accessing the release() function of the Constraint Manager.

3.3.3.1.5. Associations

Association	Type	Class	Description

3.3.3.2. Operations

3.3.3.2.1. getState()

3.3.3.2.1.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.3.3.2.2. resetCLNotification()

3.3.3.2.2.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.3.3.2.3. run()

3.3.3.2.3.1. Descriptive Text

This method will perform the execution of the RCL sequence. Notification of the sequence execution will be performed.

3.3.3.2.3.2. Sequence

```
// BEGIN MPS_RCL_E1_Reg_Press Reactive Control Logic


// VERIFY MEASUREMENTS ARE OUT OF LIMITS

IF <V41P1154A1>.getValue <= 800 AND
   <V41P1153A1>.getValue <= 800 THEN
  {
    Send Message "<clSequenceID> executed but pressure is currently less than 800 psia.'
              to MPS_RCL message destination;
    Return;
  }

// LV1 CLOSE
  SET <V41K1155XL> OFF;

// LV2 CLOSE
  SET <V41K1156XL> OFF;
  SET <V41K1157XL> OFF;

 Send Message "<clSequenceID> executed, LV1/LV2 closed'
              to MPS_RCL message destination;

  }


// SET/SEND NOTIFICATION


Set <CLNotificationFlag> to ON;

Return;


// END REACTIVE SEQUENCE
```

3.3.3.2.4. determineState()

3.3.3.2.4.1. Descriptive Text

This method will determine the enumerated state of the RCL sequence based on the associated initiating measurements and will return either: ACTIVE OR INHIBITED.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.3.3.2.4.2. Sequence

IF (getConstraintInfo<MPS_V41P1154A1_RCL> = 'ACTIVE' OR
getConstraintInfo<MPS_V41P1153A1_RCL> = 'ACTIVE') THEN CLState == 'ACTIVE';

IF (getConstraintInfo<MPS_V41P1154A1_RCL> = 'INHIBITED' AND
getConstraintInfo<MPS_V41P1153A1_RCL> = 'INHIBITED') THEN CLState == 'INHIBITED';

3.3.3.2.5. activate()

3.3.3.2.5.1. Descriptive Text

This method will activate RCL notification for all unbypassed initiating measurements.

3.3.3.2.5.2. Sequence

IF <V41P1154A1>.BYP is OFF, activate<MPS_V41P1154A1_RCL>;
IF <V41P1153A1>.BYP is OFF, activate<MPS_V41P1153A1_RCL>;

3.3.3.2.6. inhibit()

3.3.3.2.6.1. Descriptive Text

This method will inhibit RCL notification for all initiating measurements.

3.3.3.2.6.2. Sequence

inhibit<MPS_V41P1154A1_RCL>;
inhibit<MPS_V41P1153A1_RCL>;

3.3.3.2.7. alter(lo,hi)

3.3.3.2.7.1. Descriptive Text

This method will change the RCL Limit for all initiating measurements

3.3.3.2.7.2. Sequence

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

alter(lo,hi)<MPS_V41P1154A1_RCL>;
alter(lo,hi)<MPS_V41P1153A1_RCL>;

3.3.3.2.8. statusRCL()

3.3.3.2.8.1. Descriptive Text

This method will status all the RCL initiating measurements and output the status (active/inhibited), limits (LO,HI) and also the RCL Override.

3.3.3.2.8.2. Sequence

Record Message '<clSequenceID> status is:'
'initiating measurements: '<V41P1154A1>.getdescription, <V41P1154A1>.getvalue(),
<V41P1153A1>.getdescription, <V41P1153A1>.getvalue();
'RCL Limits: ' getConstraintInfo<MPS_V41P1154A1_RCL>;
getConstraintInfo<MPS_V41P1153A1_RCL>;
'RCL Notification: ' <CLNotificationFlag>.getValue();
'RCL State: ' MPS_RCL_E1_Reg_Press::getState();

3.3.3.2.9. assertRCL()

3.3.3.2.9.1. Descriptive Text

This method will initialize the RCL monitoring for each initiating measurement by accessing the assert() function of the Constraint Manager.

3.3.3.2.9.2. Sequence

// Initialize Constraints

assertAtomicConstraint<V41P1154A1>(LO,800,MPS_V41P1154A1_RCL)with callback pointer to
MPS_RCL_E1_Reg_Press::run();
assertAtomicConstraint<V41P1153A1>(LO,800,MPS_V41P1153A1_RCL)with callback pointer to
MPS_RCL_E1_Reg_Press::run();

Record Message '<clSequenceID> Monitoring has been initialized' to <PAGE-A>;

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.3.3.2.10. release()

3.3.3.2.10.1. Descriptive Text

This method will terminate the RCL monitoring for each initiating measurement by accessing the release() function of the Constraint Manager.

3.3.3.2.10.2. Sequence

releaseAtomicConstraint<MPS_V41P1154A1_RCL>;
releaseAtomicConstraint<MPS_V41P1153A1_RCL>;

Record Message '<clSequenceID> Monitoring has been terminated' to <PAGE-A>;

3.3.4. MPS_RCL_E2_Reg_Press

3.3.4.1. Description

This class Contains the Reactive Control Logic that provides protection for the Orbiter SSME#2 Pneumatic Helum Reg Pressure. Overpressure protection (default set at 800 psia) will close both LV3/LV4.
This class inherits from the class CAS_Control_Logic. Reference - LPS Program VCE07

3.3.4.1.1. Instance Index

Instance	**Other Users**
MPS_RCL_E2_Reg_Press	

3.3.4.1.2. Initiating Measurements

Measurement
<V41P1254A1>
<V41P1253A1>

3.3.4.1.3. Class Diagram

<insert class diagram here>

3.3.4.1.4. Operations Index

Operation	**Description**
run()	This method will perform the execution of the RCL sequence. Notification of the sequence execution will be performed.
determineState()	This method will determine the enumerated state of the RCL sequence

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

	based on the associated initiating measurements and will return either: ACTIVE INHIBITED
getState()	This method is inherited from the base class - CAS_Control_Logic
activate()	This method will activate RCL Notification for all unbypassed initiating measurements.
inhibit()	This method will inhibit RCL Notification for all initiating measurements.
alter()	This method will change the RCL Limit for all initiating measurements.
statusRCL()	This method will status all the RCL initiating measurements and output the status (active/inhibited), limits (LO,HI) and also the CLNotificationFlag.
resetCLNotification()	This method is inherited from the base class - CAS_Control_Logic
assertRCL()	This method will initialize the RCL monitoring for each initiating measurement by accessing the assert() function of the Constraint Manager.
release()	This method will terminate the RCL monitoring for each initiating measurement by accessing the release() function of the Constraint Manager.

3.3.4.1.5. Associations

Association	Type	Class	Description

3.3.4.2. Operations

3.3.4.2.1. getState()

3.3.4.2.1.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.3.4.2.2. resetCLNotification()

3.3.4.2.2.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.3.4.2.3. run()

3.3.4.2.3.1. Descriptive Text

This method will perform the execution of the RCL sequence. Notification of the sequence execution will be performed.

3.3.4.2.3.2. Sequence

```
// BEGIN MPS_RCL_E2_Reg_Press Reactive Control Logic


// VERIFY MEASUREMENTS ARE OUT OF LIMITS

IF <V41P1254A1>.getValue <= 800 AND
   <V41P1253A1>.getValue <= 800 THEN
  {
    Send Message "<clSequenceID> executed but pressure is currently less than 800 psia.'
              to MPS_RCL message destination;
    Return;
  }


// LV3 CLOSE
  SET <V41K1255XL> OFF;

// LV4 CLOSE
  SET <V41K1256XL> OFF;
  SET <V41K1257XL> OFF;


  Send Message "<clSequenceID> executed, LV3/LV4 closed'
              to MPS_RCL message destination;
  }



// SET/SEND NOTIFICATION


Set <CLNotificationFlag> to ON;


Return;


// END REACTIVE SEQUENCE
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.3.4.2.4. determineState()

3.3.4.2.4.1. Descriptive Text

This method will determine the enumerated state of the RCL sequence based on the associated initiating measurements and will return either: ACTIVE OR INHIBITED.

3.3.4.2.4.2. Sequence

IF (getConstraintInfo<MPS_V41P1254A1_RCL> = 'ACTIVE' OR
getConstraintInfo<MPS_V41P1253A1_RCL> = 'ACTIVE') THEN CLState == 'ACTIVE';

IF (getConstraintInfo<MPS_V41P1254A1_RCL> = 'INHIBITED' AND
getConstraintInfo<MPS_V41P1253A1_RCL> = 'INHIBITED') THEN CLState == 'INHIBITED';

3.3.4.2.5. activate()

3.3.4.2.5.1. Descriptive Text

This method will activate RCL notification for all unbypassed initiating measurements.

3.3.4.2.5.2. Sequence

IF <V41P1254A1>.BYP is OFF, activate<MPS_V41P1254A1_RCL>;
IF <V41P1253A1>.BYP is OFF, activate<MPS_V41P1253A1_RCL>;

3.3.4.2.6. inhibit()

3.3.4.2.6.1. Descriptive Text

This method will inhibit RCL notification for all initiating measurements.

3.3.4.2.6.2. Sequence

inhibit<MPS_V41P1254A1_RCL>;
inhibit<MPS_V41P1253A1_RCL>;

3.3.4.2.7. alter(lo,hi)

3.3.4.2.7.1. Descriptive Text

This method will change the RCL Limit for all initiating measurements

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.3.4.2.7.2. Sequence

alter(lo,hi)<MPS_V41P1254A1_RCL>;
alter(lo,hi)<MPS_V41P1253A1_RCL>;

3.3.4.2.8. statusRCL()

3.3.4.2.8.1. Descriptive Text

This method will status all the RCL initiating measurements and output the status (active/inhibited), limits (LO,HI) and also the CLNotificationFlag.

3.3.4.2.8.2. Sequence

Record Message '<clSequenceID> status is:'
'initiating measurements: '<V41P1254A1>.getdescription, <V41P1254A1>.getvalue,
<V41P1253A1>.getdescription, <V41P1253A1>.getvalue();
'RCL Limits: ' getConstraintInfo<MPS_V41P1254A1_RCL>;
getConstraintInfo<MPS_V41P1253A1_RCL>;
'RCL Notification: ' <CLNotificationFlag>.getValue();
'RCL State: ' MPS_RCL_E2_Reg_Press::getState();

3.3.4.2.9. assertRCL()

3.3.4.2.9.1. Descriptive Text

This method will initialize the RCL monitoring for each initiating measurement by accessing the assert() function of the Constraint Manager.

3.3.4.2.9.2. Sequence

// Initialize Constraints

assertAtomicConstraint<V41P1254A1>(LO,800,MPS_V41P1254A1_RCL) with callback pointer to
MPS_RCL_E2_Reg_Press::run();
assertAtomicConstraint<V41P1253A1>(LO,800,MPS_V41P1253A1_RCL) with callback pointer to
MPS_RCL_E2_Reg_Press::run();

Record Message '<clSequenceID> Monitoring has been initialized' to <PAGE-A>;

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.3.4.2.10. release()

3.3.4.2.10.1. Descriptive Text

This method will terminate the RCL monitoring for each initiating measurement by accessing the release() function of the Constraint Manager.

3.3.4.2.10.2. Sequence

releaseAtomicConstraint<MPS_V41P1254A1_RCL>;
releaseAtomicConstraint<MPS_V41P1253A1_RCL>;

Record Message '<clSequenceID> Monitoring has been terminated' to <PAGE-A>;

3.3.5. MPS_RCL_E3_Reg_Press

3.3.5.1. Description

This class Contains the Reactive Control Logic that provides protection for the Orbiter SSME#3 Pneumatic Helum Reg Pressure. Overpressure protection (default set at 800 psia) will close both LV5/LV6.
This class inherits from the class CAS_Control_Logic. Reference - LPS Program VCE08

3.3.5.1.1. Instance Index

Instance	**Other Users**
MPS_RCL_E3_Reg_Press	

3.3.5.1.2. Initiating Measurements

Measurement
<V41P1354A1>
<V41P1353A1>

3.3.5.1.3. Class Diagram

<insert class diagram here>

3.3.5.1.4. Operations Index

Operation	**Description**
run()	This method will perform the execution of the RCL sequence. Notification of the sequence execution will be performed.
determineState()	This method will determine the enumerated state of the RCL sequence based on the associated initiating measurements and will return either:

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

	ACTIVE INHIBITED
getState()	This method is inherited from the base class - CAS_Control_Logic
activate()	This method will activate RCL Notification for all unbypassed initiating measurements.
inhibit()	This method will inhibit RCL Notification for all initiating measurements.
alter(lo,hi)	This method will change the RCL Limit for all initiating measurements.
statusRCL()	This method will status all the RCL initiating measurements and output the status (active/inhibited), limits (LO,HI) and also the CLNotificationFlag.
resetCLNotification()	This method is inherited from the base class - CAS_Control_Logic
assertRCL()	This method will initialize the RCL monitoring for each initiating measurement by accessing the assert() function of the Constraint Manager.
release()	This method will terminate the RCL monitoring for each initiating measurement by accessing the release() function of the Constraint Manager.

3.3.5.1.5. Associations

Association	Type	Class	Description

3.3.5.2. Operations

3.3.5.2.1. getState()

3.3.5.2.1.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.3.5.2.2. resetCLNotification()

3.3.5.2.2.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.3.5.2.3. run()

3.3.5.2.3.1. Descriptive Text

This method will perform the execution of the RCL sequence. Notification of the sequence execution will be performed.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.3.5.2.3.2. Sequence

```
// BEGIN MPS_RCL_E3_Reg_Press Reactive Control Logic

// VERIFY MEASUREMENTS ARE OUT OF LIMITS

IF <V41P1354A1>.getValue <= 800 AND
   <V41P1353A1>.getValue <= 800 THEN
  {
    Send Message "<clSequenceID> executed but pressure is currently less than 800 psia.'
                to MPS_RCL message destination;
    Return;
  }

// LV5 CLOSE
  SET <V41K1355XL> OFF;

// LV6 CLOSE
  SET <V41K1356XL> OFF;
  SET <V41K1357XL> OFF;

  Send Message "<clSequenceID> executed, LV5/LV6 closed'
                to MPS_RCL message destination;
  }

// SET/SEND NOTIFICATION

Set <CLNotificationFlag> to ON;

Return;

// END REACTIVE SEQUENCE
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.3.5.2.4. determineState()

3.3.5.2.4.1. Descriptive Text

This method will determine the enumerated state of the RCL sequence based on the associated initiating measurements and will return either: ACTIVE OR INHIBITED.

3.3.5.2.4.2. Sequence

IF (getConstraintInfo<MPS_V41P1354A1_RCL> = 'ACTIVE' OR
getConstraintInfo<MPS_V41P1353A1_RCL> = 'ACTIVE') THEN CLState == 'ACTIVE';

IF (getConstraintInfo<MPS_V41P1354A1_RCL> = 'INHIBITED' AND
getConstraintInfo<MPS_V41P1353A1_RCL> = 'INHIBITED') THEN CLState == 'INHIBITED';

3.3.5.2.5. activate()

3.3.5.2.5.1. Descriptive Text

This method will activate RCL notification for all unbypassed initiating measurements.

3.3.5.2.5.2. Sequence

IF <V41P1354A1>.BYP is OFF, activate<MPS_V41P1354A1_RCL>;
IF <V41P1353A1>.BYP is OFF, activate<MPS_V41P1353A1_RCL>;

3.3.5.2.6. inhibit()

3.3.5.2.6.1. Descriptive Text

This method will inhibit RCL notification for all initiating measurements.

3.3.5.2.6.2. Sequence

inhibit<MPS_V41P1354A1_RCL>;
inhibit<MPS_V41P1353A1_RCL>;

3.3.5.2.7. alter(lo,hi)

3.3.5.2.7.1. Descriptive Text

This method will change the RCL Limit for all initiating measurements

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.3.5.2.7.2. Sequence

alter(lo,hi)<MPS_V41P1354A1_RCL>;
alter(lo,hi)<MPS_V41P1353A1_RCL>;

3.3.5.2.8. statusRCL()

3.3.5.2.8.1. Descriptive Text

This method will status all the RCL initiating measurements and output the status (active/inhibited), limits (LO,HI) and also the RCL Override.

3.3.5.2.8.2. Sequence

Record Message '<clSequenceID> status is:'
'initiating measurements: ' <V41P1354A1>.getdescription, <V41P1354A1>.getvalue,
<V41P1353A1>.getdescription, <V41P1353A1>.getvalue();
'RCL Limits: ' getConstraintInfo<MPS_V41P1354A1_RCL>;
getConstraintInfo<MPS_V41P1353A1_RCL>;
'RCL Notification: ' <CLNotificationFlag>.getValue();
'RCL State: ' MPS_RCL_E3_Reg_Press::getState();

3.3.5.2.9. assertRCL()

3.3.5.2.9.1. Descriptive Text

This method will initialize the RCL monitoring for each initiating measurement by accessing the assert() function of the Constraint Manager.

3.3.5.2.9.2. Sequence

// Initialize Constraints

assertAtomicConstraint<V41P1354A1>(LO,800,MPS_V41P1354A1_RCL) with callback pointer to
MPS_RCL_E3_Reg_Press::run();
assertAtomicConstraint<V41P1353A1>(LO,800,MPS_V41P1353A1_RCL) with callback pointer to
MPS_RCL_E3_Reg_Press::run();

Record Message '<clSequenceID> Monitoring has been initialized' to <PAGE-A>;

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.3.5.2.10. release()

3.3.5.2.10.1. Descriptive Text

This method will terminate the RCL monitoring for each initiating measurement by accessing the release() function of the Constraint Manager.

3.3.5.2.10.2. Sequence

releaseAtomicConstraint<MPS_V41P1354A1_RCL>;
releaseAtomicConstraint<MPS_V41P1353A1_RCL>;

Record Message '<clSequenceID> Monitoring has been terminated' to <PAGE-A>;

3.3.6. MPS_RCL_GHe_Bottle_Temp

3.3.6.1. Description

This class Contains the Reactive Control Logic that provides protection for the Orbiter MPS and SSME 1/2/3 Pneumatic Helum Bottle Temperatures. Overtemp protection (default set at 190 degf) will terminate the Helium supply to the Orbiter by: In the OPF:

MPS_A83128
MPS_A83129
MPS_A83130
MPS_A83131
MPS_A83132
MPS_A83133

At the VAB/PAD:

MPS_A74936
MPS_A74935
MPS_A74937
MPS_A74983

This class inherits from the class CAS_Control_Logic. Reference - LPS Program VCE16

3.3.6.1.1. Instance Index

Instance	**Other Users**
MPS_RCL_GHe_Bottle_Temp	

3.3.6.1.2. Initiating Measurements

Measurement
<V41T1601A1>
<V41T1151A1>
<V41T1152A1>
<V41T1251A1>
<V41T1252A1>

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

<V41T1351A1>
<V41T1352A1>

3.3.6.1.3. Class Diagram

<insert class diagram here>

3.3.6.1.4. Operations Index

Operation	**Description**
run()	This method will perform the execution of the RCL sequence. Notification of the sequence execution will be performed.
determineState()	This method will determine the enumerated state of the RCL sequence based on the associated initiating measurements and will return either: ACTIVE INHIBITED
getState()	This method is inherited from the base class - CAS_Control_Logic
activate()	This method will activate RCL Notification for all unbypassed initiating measurements.
inhibit()	This method will inhibit RCL Notification for all initiating measurements.
alter(lo,hi)	This method will change the RCL Limit for all initiating measurements.
statusRCL()	This method will status all the RCL initiating measurements and output the status (active/inhibited), limits (LO,HI) and also the CLNotificationFlag.
resetCLNotification()	This method is inherited from the base class - CAS_Control_Logic
assertRCL()	This method will initialize the RCL monitoring for each initiating measurement by accessing the assert() function of the Constraint Manager.
release()	This method will terminate the RCL monitoring for each initiating measurement by accessing the release() function of the Constraint Manager.

3.3.6.1.5. Associations

Association	**Type**	**Class**	**Description**

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.3.6.2. Operations

3.3.6.2.1. getState()

3.3.6.2.1.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.3.6.2.2. resetCLNotification()

3.3.6.2.2.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.3.6.2.3. run()

3.3.6.2.3.1. Descriptive Text

This method will perform the execution of the RCL sequence. Notification of the sequence execution will be performed.

3.3.6.2.3.2. Sequence

```
// BEGIN  MPS_RCL_GHe_Bottle_Temp  Reactive Control Logic


// VERIFY MEASUREMENTS ARE OUT OF LIMITS

IF  <V41T1601A1>.getValue <=  190 AND
    <V41T1151A1>.getValue <=  190 AND
    <V41T1152A1>.getValue <=  190 AND
    <V41T1251A1>.getValue <=  190 AND
    <V41T1252A1>.getValue <=  190 AND
    <V41T1351A1>.getValue <=  190 AND
    <V41T1352A1>.getValue <=  190 THEN
  {
    Send Message "<clSequenceID> executed but temp is currently less than 190 degf.'
              to MPS_RCL message destination;
    Return;
  }

// CHECK IF ORB IS IN OPF

IF  NINTORBITERLOCATION = OPF THEN
  {


// A83128  CLOSE
   SET <GHEK2000E>  ON

// A83129  CLOSE
   SET <GHEK2010E>  OFF

// A83130  CLOSE
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
   SET <GHEK2020E>  OFF

//  A83131  CLOSE
   SET <GHEK2040E>  ON

//  A83132  CLOSE
   SET <GHEK2050E>  OFF

//  A83133  CLOSE
   SET <GHEK2060E>  OFF

     Send Message "<clSequenceID> executed, OPF safed'
                to MPS_RCL message destination;
  }



//  CHECK IF ORB IS AT VAB OR PAD

IF  NINTORBITERLOCATION = VAB OR
    NINTORBITERLOCATION = PAD THEN
  {

//  A74936  CLOSE
   SET <GHEK1000E>  ON
   SET <GHEK1008E>  OFF
   SET <GHEK1100E>  ON
   SET <GHEK1108E>  OFF

//  A75935  CLOSE
   SET <GHEK1040E>  OFF
   SET <GHEK1047E>  ON

//  A75937  CLOSE
   SET <GHEK1160E>  OFF
   SET <GHEK1167E>  ON

//  A74983  CLOSE
   SET <GHEK1090E>  OFF
   SET <GHEK1097E>  ON
   SET <GHEK1190E>  OFF
   SET <GHEK1197E>  ON

     Send Message "<clSequenceID> executed, MLP safed'
                to MPS_RCL message destination;
  }


//  SET/SEND NOTIFICATION


Set <CLNotificationFlag> to ON;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

}

Return;

// END REACTIVE SEQUENCE

3.3.6.2.4. determineState()

3.3.6.2.4.1. Descriptive Text

This method will determine the enumerated state of the RCL sequence based on the associated initiating measurements and will return either: ACTIVE OR INHIBITED.

3.3.6.2.4.2. Sequence

```
IF (getConstraintInfo<MPS_V41T1601A1_RCL> = 'ACTIVE' OR
   getConstraintInfo<MPS_V41T1151A1_RCL> = 'ACTIVE' OR
   getConstraintInfo<MPS_V41T1152A1_RCL> = 'ACTIVE' OR
   getConstraintInfo<MPS_V41T1251A1_RCL> = 'ACTIVE' OR
   getConstraintInfo<MPS_V41T1252A1_RCL> = 'ACTIVE' OR
   getConstraintInfo<MPS_V41T1351A1_RCL> = 'ACTIVE' OR
   getConstraintInfo<MPS_V41T1352A1_RCL> = 'ACTIVE') THEN CLState == 'ACTIVE';


IF (getConstraintInfo<MPS_V41T1601A1_RCL> = 'INHIBITED' AND
   getConstraintInfo<MPS_V41T1151A1_RCL> = 'INHIBITED' AND
   getConstraintInfo<MPS_V41T1152A1_RCL> = 'INHIBITED' AND
   getConstraintInfo<MPS_V41T1251A1_RCL> = 'INHIBITED' AND
   getConstraintInfo<MPS_V41T1252A1_RCL> = 'INHIBITED' AND
   getConstraintInfo<MPS_V41T1351A1_RCL> = 'INHIBITED' AND
   getConstraintInfo<MPS_V41T1352A1_RCL> = 'INHIBITED') THEN CLState == 'INHIBITED';
```

3.3.6.2.5. activate()

3.3.6.2.5.1. Descriptive Text

This method will activate RCL notification for all unbypassed initiating measurements.

3.3.6.2.5.2. Sequence

```
IF <V41T1601A1>.BYP is OFF, activate<MPS_V41T1601A1_RCL>;
IF <V41T1151A1>.BYP is OFF, activate<MPS_V41T1151A1_RCL>;
IF <V41T1152A1>.BYP is OFF, activate<MPS_V41T1152A1_RCL>;
IF <V41T1251A1>.BYP is OFF, activate<MPS_V41T1251A1_RCL>;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

IF <V41T1252A1>.BYP is OFF, activate<MPS_V41T1252A1_RCL>;
IF <V41T1351A1>.BYP is OFF, activate<MPS_V41T1351A1_RCL>;
IF <V41T1352A1>.BYP is OFF, activate<MPS_V41T1352A1_RCL>;

3.3.6.2.6. inhibit()

3.3.6.2.6.1. Descriptive Text

This method will inhibit RCL notification for all initiating measurements.

3.3.6.2.6.2. Sequence

inhibit<MPS_V41T1601A1_RCL>;
inhibit<MPS_V41T1151A1_RCL>;
inhibit<MPS_V41T1152A1_RCL>;
inhibit<MPS_V41T1251A1_RCL>;
inhibit<MPS_V41T1252A1_RCL>;
inhibit<MPS_V41T1351A1_RCL>;
inhibit<MPS_V41T1352A1_RCL>;

3.3.6.2.7. alter(lo,hi)

3.3.6.2.7.1. Descriptive Text

This method will change the RCL Limit for all initiating measurements

3.3.6.2.7.2. Sequence

alter(lo,hi)<MPS_V41T1601A1_RCL>;
alter(lo,hi)<MPS_V41T1151A1_RCL>;
alter(lo,hi)<MPS_V41T1152A1_RCL>;
alter(lo,hi)<MPS_V41T1251A1_RCL>;
alter(lo,hi)<MPS_V41T1252A1_RCL>;
alter(lo,hi)<MPS_V41T1351A1_RCL>;
alter(lo,hi)<MPS_V41T1352A1_RCL>;

3.3.6.2.8. statusRCL()

3.3.6.2.8.1. Descriptive Text

This method will status all the RCL initiating measurements and output the status (active/inhibited), limits (LO,HI) and also the RCL Override.

3.3.6.2.8.2. Sequence

Record Message '<clSequenceID> status is:'
'initiating measurements: ' <V41T1601A1>.get.description, <V41T1601A1>.get.value'

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

<V41T1151A1>.getdescription, <V41T1151A1>.getvalue,
<V41T1152A1>.getdescription, <V41T1152A1>.getvalue,
<V41T1251A1>.getdescription, <V41T1251A1>.getvalue,
<V41T1252A1>.getdescription, <V41T1252A1>.getvalue,
<V41T1351A1>.getdescription, <V41T1351A1>.getvalue;
<V41T1352A1>.getdescription, <V41T1352A1>.getvalue,
'RCL Limits: ' getConstraintInfo<MPS_V41T1601A1_RCL>;
getConstraintInfo<MPS_V41T1151A1_RCL>;
getConstraintInfo<MPS_V41T1152A1_RCL>;
getConstraintInfo<MPS_V41T1251A1_RCL>;
getConstraintInfo<MPS_V41T1252A1_RCL>;
getConstraintInfo<MPS_V41T1351A1_RCL>;
getConstraintInfo<MPS_V41T1352A1_RCL>;
'RCL Notification: ' <CLNotificationFlag>.getValue();
'RCL State: ' MPS_RCL_GHe_Bottle_Temp::getState();

3.3.6.2.9. assertRCL()

3.3.6.2.9.1. Descriptive Text

This method will initialize the RCL monitoring for each initiating measurement by accessing the assert() function of the Constraint Manager.

3.3.6.2.9.2. Sequence

// Initialize Constraints

assertAtomicConstraint<V41T1601A1>(LO,190,MPS_V41T1600A1_RCL) with callback pointer to
MPS_RCL_GHe_Bottle_Temp::run();
assertAtomicConstraint<V41T1151A1>(LO,190,MPS_V41T1151A1_RCL) with callback pointer to
MPS_RCL_GHe_Bottle_Temp::run();
assertAtomicConstraint<V41T1152A1>(LO,190,MPS_V41T1152A1_RCL) with callback pointer to
MPS_RCL_GHe_Bottle_Temp::run();
assertAtomicConstraint<V41T1251A1>(LO,190,MPS_V41T1251A1_RCL) with callback pointer to
MPS_RCL_GHe_Bottle_Temp::run();
assertAtomicConstraint<V41T1252A1>(LO,190,MPS_V41T1252A1_RCL) with callback pointer to
MPS_RCL_GHe_Bottle_Temp::run();
assertAtomicConstraint<V41T1351A1>(LO,190,MPS_V41T1351A1_RCL) with callback pointer to
MPS_RCL_GHe_Bottle_Temp::run();
assertAtomicConstraint<V41T1352A1>(LO,190,MPS_V41T1352A1_RCL) with callback pointer to
MPS_RCL_GHe_Bottle_Temp::run();

Record Message '<clSequenceID> Monitoring has been initialized' to <PAGE-A>;

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.3.6.2.10. release()

3.3.6.2.10.1. Descriptive Text

This method will terminate the RCL monitoring for each initiating measurement by accessing the release() function of the Constraint Manager.

3.3.6.2.10.2. Sequence

releaseAtomicConstraint<MPS_V41T1601A1_RCL>;
releaseAtomicConstraint<MPS_V41T1151A1_RCL>;
releaseAtomicConstraint<MPS_V41T1152A1_RCL>;
releaseAtomicConstraint<MPS_V41T1251A1_RCL>;
releaseAtomicConstraint<MPS_V41T1252A1_RCL>;
releaseAtomicConstraint<MPS_V41T1351A1_RCL>;
releaseAtomicConstraint<MPS_V41T1352A1_RCL>;

Record Message '<clSequenceID> Monitoring has been terminated' to <PAGE-A>;

3.3.7. MPS_RCL_LO2_Manifold_Press

3.3.7.1. Description

This class Contains the Reactive Control Logic that provides protection for the Orbiter MPS LO2 Manifold Pressures. Overpressure protection (default set at 70 psia) will terminate the Manifold repress and open the Manifold relief isolation valve. This class inherits from the class CAS_Control_Logic. Reference - LPS Program VCE01

3.3.7.1.1. Instance Index

Instance	**Other Users**
MPS_RCL_LO2_Manifold_Press	

3.3.7.1.2. Initiating Measurements

Measurement
<V41P1533C1>

3.3.7.1.3. Class Diagram

<insert class diagram here>

3.3.7.1.4. Operations Index

Operation	**Description**
run()	This method will perform the execution of the RCL sequence. Notification of the sequence execution will be performed.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

determineState()	This method will determine the enumerated state of the RCL sequence based on the associated initiating measurements and will return either: ACTIVE INHIBITED
getState()	This method is inherited from the base class - CAS_Control_Logic
activate()	This method will activate RCL Notification for all unbypassed initiating measurements.
inhibit()	This method will inhibit RCL Notification for all initiating measurements.
alter(lo,hi)	This method will change the RCL Limit for all initiating measurements.
statusRCL()	This method will status all the RCL initiating measurements and output the status (active/inhibited), limits (LO,HI) and also the CLNotificationFlag.
resetCLNotification()	This method is inherited from the base class - CAS_Control_Logic
assertRCL()	This method will initialize the RCL monitoring for each initiating measurement by accessing the assert() function of the Constraint Manager.
release()	This method will terminate the RCL monitoring for each initiating measurement by accessing the release() function of the Constraint Manager.

3.3.7.1.5. Associations

Association	Type	Class	Description

3.3.7.2. Operations

3.3.7.2.1. getState()

3.3.7.2.1.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.3.7.2.2. resetCLNotification()

3.3.7.2.2.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.3.7.2.3. run()

3.3.7.2.3.1. Descriptive Text

This method will perform the execution of the RCL sequence. Notification of the sequence execution will be performed.

3.3.7.2.3.2. Sequence

```
// BEGIN MPS_RCL_LO2_Manifold_Press Reactive Control Logic


// VERIFY MEASUREMENTS ARE OUT OF LIMITS

IF <V41P1533C1>.getValue <= 70 THEN
  {
    Send Message "<clSequenceID> executed but pressure is currently less than 70 psia.'
            to MPS_RCL message destination;
    Return;
  }


// SAFE ORBITER


// LV40 CLOSE
 SET <V41K1535XL> OFF;

// LV41 CLOSE
 SET <V41K1537XL> OFF;


// PV7 CLOSE
 SET <V41K1547XL> OFF;
 SET <V41K1548XL> OFF;
 SET <V41K1550XL> OFF;


// CHECK IF ORB IS IN OPF


IF NINTORBITERLOCATION = OPF THEN
  {



// A82916 CLOSE
   SET <GHEK2120E> OFF;

// A82921 OPEN
   SET <GHEK2130E> ON;

// A82915 CLOSE
   SET <GHEK2110E> OFF;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

}

// SET/SEND NOTIFICATION

Set <CLNotificationFlag> to ON;

Send Message "<clSequenceID> Reactive Control Logic executed'
to MPS_RCL message destination;

Return;

// END REACTIVE SEQUENCE

3.3.7.2.4. determineState()

3.3.7.2.4.1. Descriptive Text

This method will determine the enumerated state of the RCL sequence based on the associated initiating measurements and will return either: ACTIVE OR INHIBITED.

3.3.7.2.4.2. Sequence

IF getConstraintInfo<MPS_V41P1533C1_RCL> = 'ACTIVE' THEN CLState == 'ACTIVE';

IF getConstraintInfo<MPS_V41P1533C1_RCL> = 'INHIBITED' THEN CLState == 'INHIBITED';

3.3.7.2.5. activate()

3.3.7.2.5.1. Descriptive Text

This method will activate RCL notification for all unbypassed initiating measurements.

3.3.7.2.5.2. Sequence

IF <V41P1533C1>.BYP is OFF, activate<MPS_V41P1533C1_RCL>;

3.3.7.2.6. inhibit()

3.3.7.2.6.1. Descriptive Text

This method will inhibit RCL notification for all initiating measurements.

3.3.7.2.6.2. Sequence

inhibit<MPS_V41P1533C1_RCL>;

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.3.7.2.7. alter(lo,hi)

3.3.7.2.7.1. Descriptive Text

This method will change the RCL Limit for all initiating measurements

3.3.7.2.7.2. Sequence

alter(lo,hi)<MPS_V41P1533C1_RCL>;

3.3.7.2.8. statusRCL()

3.3.7.2.8.1. Descriptive Text

This method will status all the RCL initiating measurements and output the status (active/inhibited), limits (LO,HI) and also the RCL Override.

3.3.7.2.8.2. Sequence

Record Message '<clSequenceID> status is:'
'initiating measurements: ' <V41P1533C1>.get.description, <V41P1533C1>.get.value()';
'RCL Limits: ' getConstraintInfo<MPS_V41P1533C1_RCL>;
'RCL Notification: ' <CLNotificationFlag>.getValue();
'RCL State: ' MPS_RCL_LO2_Manifold_Press::getState();

3.3.7.2.9. assertRCL()

3.3.7.2.9.1. Descriptive Text

This method will initialize the RCL monitoring for each initiating measurement by accessing the assert() function of the Constraint Manager.

3.3.7.2.9.2. Sequence

// Initialize Constraints

assertAtomicConstraint<V41P1533C1>(LO,70,MPS_V41P1533C1_RCL) with callback pointer to
MPS_RCL_LO2_Manifold_Press::run();

Record Message '<clSequenceID> Monitoring has been initialized' to <PAGE-A>;

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.3.7.2.10. release()

3.3.7.2.10.1. Descriptive Text

This method will terminate the RCL monitoring for each initiating measurement by accessing the release() function of the Constraint Manager.

3.3.7.2.10.2. Sequence

releaseAtomicConstraint<MPS_V41P1533C1_RCL>;

Record Message '<clSequenceID> Monitoring has been terminated' to <PAGE-A>;

3.3.8. MPS_RCL_LH2_Manifold_Press

3.3.8.1. Description

This class Contains the Reactive Control Logic that provides protection for the Orbiter MPS LH2 Manifold Pressures. Overpressure protection (default set at 55 psia) will terminate the Manifold repress and open the Manifold relief isolation valve. This class inherits from the class CAS_Control_Logic. Reference - LPS Program VCE02

3.3.8.1.1. Instance Index

Instance	Other Users
MPS_RCL_LH2_Manifold_Press	

3.3.8.1.2. Initiating Measurements

Measurement
<V41P1433C1>

3.3.8.1.3. Class Diagram

<insert class diagram here>

3.3.8.1.4. Operations Index

Operation	Description
run()	This method will perform the execution of the RCL sequence. Notification of the sequence execution will be performed.
determineState()	This method will determine the enumerated state of the RCL sequence based on the associated initiating measurements and will return either: ACTIVE INHIBITED

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

getState()	This method is inherited from the base class - CAS_Control_Logic
activate()	This method will activate RCL Notification for all unbypassed initiating measurements.
inhibit()	This method will inhibit RCL Notification for all initiating measurements.
alter(lo,hi)	This method will change the RCL Limit for all initiating measurements.
statusRCL()	This method will status all the RCL initiating measurements and output the status (active/inhibited), limits (LO,HI) and also the CLNotificationFlag.
resetCLNotification()	This method is inherited from the base class - CAS_Control_Logic
assertRCL()	This method will initialize the RCL monitoring for each initiating measurement by accessing the assert() function of the Constraint Manager.
release()	This method will terminate the RCL monitoring for each initiating measurement by accessing the release() function of the Constraint Manager.

3.3.8.1.5. Associations

Association	Type	Class	Description

3.3.8.2. Operations

3.3.8.2.1. getState()

3.3.8.2.1.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.3.8.2.2. resetCLNotification()

3.3.8.2.2.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.3.8.2.3. run()

3.3.8.2.3.1. Descriptive Text

This method will perform the execution of the RCL sequence. Notification of the sequence execution will be performed.

3.3.8.2.3.2. Sequence

// BEGIN MPS_RCL_LH2_Manifold_Press Reactive Control Logic

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
// VERIFY MEASUREMENTS ARE OUT OF LIMITS

IF <V41P1433C1>.getValue <= 55 THEN
  {
    Send Message "<clSequenceID> executed but pressure is currently less than 55 psia.'
          to MPS_RCL message destination;
    Return;
  }


// SAFE ORBITER

// LV42 CLOSE
 SET <V41K1435XL> OFF;

// LV43 CLOSE
 SET <V41K1437XL> OFF;

// LV74 CLOSE
 SET <V41K1905XL> OFF;
 SET <V41K1907XL> OFF;
 SET <V41K1909XL> OFF;

// LV75 CLOSE
 SET <V41K1906XL> OFF;
 SET <V41K1908XL> OFF;
 SET <V41K1910XL> OFF;


// PV8 OPEN
 SET <V41K1447XL> OFF;
 SET <V41K1448XL> OFF;
 SET <V41K1450XL> OFF;



// CHECK IF ORB IS IN OPF


IF NINTORBITERLOCATION = OPF THEN
  {

// A83135 CLOSE

  SET <GHEK2070E> OFF;

// A83041 OPEN

  SET <GHEK2080E> ON;

  }
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

// SET/SEND NOTIFICATION

Set <CLNotificationFlag> to ON;

Send Message "<clSequenceID> Reactive Control Logic executed'
to MPS_RCL message destination;

Return;

// END REACTIVE SEQUENCE

3.3.8.2.4. determineState()

3.3.8.2.4.1. Descriptive Text

This method will determine the enumerated state of the RCL sequence based on the associated initiating measurements and will return either: ACTIVE OR INHIBITED

3.3.8.2.4.2. Sequence

IF getConstraintInfo<MPS_V41P1433C1_RCL> = 'ACTIVE' THEN CLState == 'ACTIVE';

IF getConstraintInfo<MPS_V41P1433C1_RCL> = 'INHIBITED' THEN CLState == 'INHIBITED';

3.3.8.2.5. activate()

3.3.8.2.5.1. Descriptive Text

This method will activate RCL notification for all unbypassed initiating measurements.

3.3.8.2.5.2. Sequence

IF <V41P1433C1>.BYP is OFF, activate<MPS_V41P1433C1_RCL>;

3.3.8.2.6. inhibit()

3.3.8.2.6.1. Descriptive Text

This method will inhibit RCL notification for all initiating measurements.

3.3.8.2.6.2. Sequence

inhibit<MPS_V41P1433C1_RCL>;

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.3.8.2.7. alter(lo,hi)

3.3.8.2.7.1. Descriptive Text

This method will change the RCL Limit for all initiating measurements

3.3.8.2.7.2. Sequence

alter(lo,hi)<MPS_V41P1433C1_RCL>;

3.3.8.2.8. statusRCL()

3.3.8.2.8.1. Descriptive Text

This method will status all the RCL initiating measurements and output the status (active/inhibited), limits (LO,HI) and also the RCL Override.

3.3.8.2.8.2. Sequence

Record Message '<clSequenceID> status is:'
'initiating measurements: ' <V41P1433C1>.get.description, <V41P1433C1>.get.value()';
'RCL Limits: ' getConstraintInfo<MPS_V41P1433C1_RCL>;
'RCL Notification: ' <CLNotificationFlag>.getValue();
'RCL State: ' MPS_RCL_LH2_Manifold_Press::getState();

3.3.8.2.9. assertRCL()

3.3.8.2.9.1. Descriptive Text

This method will initialize the RCL monitoring for each initiating measurement by accessing the assert() function of the Constraint Manager.

3.3.8.2.9.2. Sequence

// Initialize Constraints

assertAtomicConstraint<V41P1433C1>(LO,55,MPS_V41P1433C1_RCL) with callback pointer to
MPS_RCL_LH2_Manifold_Press::run();

Record Message '<clSequenceID> Monitoring has been initialized' to <PAGE-A>;

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.3.8.2.10. release()

3.3.8.2.10.1. Descriptive Text

This method will terminate the RCL monitoring for each initiating measurement by accessing the release() function of the Constraint Manager.

3.3.8.2.10.2. Sequence

releaseAtomicConstraint<MPS_V41P1433C1_RCL>;

Record Message '<clSequenceID> Monitoring has been terminated' to <PAGE-A>;

3.3.9. MPS_RCL_LO2_PV9_OP_Ind

3.3.9.1. Description

This class Contains the Reactive Control Logic that provides protection for the Orbiter MPS LO2 PV9 Open Indicator. Protection will initiate an LO2 stopflow. This class inherits from the class CAS_Control_Logic. Reference - LPS Program GCL54

3.3.9.1.1. Instance Index

Instance	**Other Users**
MPS_RCL_LO2_PV9_OP_Ind	LO2

3.3.9.1.2. Initiating Measurements

Measurement
<V41X1513E1>

3.3.9.1.3. Class Diagram

<insert class diagram here>

3.3.9.1.4. Operations Index

Operation	**Description**
run()	This method will perform the execution of the RCL sequence. Notification of the sequence execution will be performed.
determineState()	This method will determine the enumerated state of the RCL sequence based on the associated initiating measurements and will return either:

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

	ACTIVE INHIBITED
getState()	This method is inherited from the base class - CAS_Control_Logic
activate()	This method will activate RCL Notification for all unbypassed initiating measurements.
inhibit()	This method will inhibit RCL Notification for all initiating measurements.
statusRCL()	This method will status all the RCL initiating measurements and output the status (active/inhibited), limits (OFF,ON) and also the CLNotificationFlag.
resetCLNotification()	This method is inherited from the base class - CAS_Control_Logic
assertRCL()	This method will initialize the RCL monitoring for each initiating measurement by accessing the assert() function of the Constraint Manager.
release()	This method will terminate the RCL monitoring for each initiating measurement by accessing the release() function of the Constraint Manager.

3.3.9.1.5. Associations

Association	Type	Class	Description
MPS_RCL_LO2_PV9_OP_Ind	Static	LO2_RCL_StopFlow	Allows: 1. MPS_RCL_LO2_PV9_OP_Ind.run() to call LO2_RCL_StopFlow.runStopFlo

3.3.9.2. Operations

3.3.9.2.1. getState()

3.3.9.2.1.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.3.9.2.2. resetCLNotification()

3.3.9.2.2.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.3.9.2.3. run()

3.3.9.2.3.1. Descriptive Text

This method will perform the execution of the RCL sequence. Notification of the sequence execution will be performed.

3.3.9.2.3.2. Sequence

```
// BEGIN MPS_RCL_LO2_PV9_OP_Ind Reactive Control Logic



// CHECK IF PV9 OPEN INDICATOR IS BYPASSED

   IF <V41X1513E1>.BYP = ON THEN
    {
     Send Message "<clSequenceId> executed but indicator was Bypassed'
           to MPS_RCL message destination;
     Return;
    }



// VERIFY PV9 OPEN INDICATOR IS NOT ON

   IF <V41X1513E1>.getValue = ON THEN
    {
     Send Message "<clSequenceId> executed but OP IND currently is ON'
           to MPS_RCL message destination;
     Return;
    }



// VERIFY LO2 LOADING SEQUENCER IS AUTOFILL OR REPLENISH OR DRAIN

IF <NLO2_SEQUENCER_STATE> = "AUTOFILL" OR
   <NLO2_OPERATION_STATE> = "REPLENISH" OR
   <NLO2_SEQUENCER_STATE> = "AUTODRAIN" THEN

  {

   // SAFE ORBITER

   // PV9 OPEN

   Set <V41K1518XL> to ON;
   Set <V41K1518NL> to ON;
   Set <V41K1515XL> to OFF;


    // CALL FOR LO2 STOPFLOW

   LO2_RCL_StopFlow.runStopFlow();
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

// SET/SEND NOTIFICATION

Set <CLNotificationFlag> to ON;

Send Message "<clSequenceId> Reactive Control Logic executed'
to MPS_RCL message destination;

}

Return;

// END REACTIVE SEQUENCE

3.3.9.2.4. determineState()

3.3.9.2.4.1. Descriptive Text

This method will determine the enumerated state of the RCL sequence based on the associated initiating measurements and will return either: ACTIVE OR INHIBITED.

3.3.9.2.4.2. Sequence

IF getConstraintInfo<MPS_V41X1513E1_RCL> = 'ACTIVE' THEN CLState == 'ACTIVE';

IF getConstraintInfo<MPS_V41X1513E1_RCL> = 'INHIBITED' THEN CLState == 'INHIBITED';

3.3.9.2.5. activate()

3.3.9.2.5.1. Descriptive Text

This method will activate RCL notification for all unbypassed initiating measurements.

3.3.9.2.5.2. Sequence

IF <V41X1513E1>.BYP is OFF, activate<MPS_V41X1513E1_RCL>;

3.3.9.2.6. inhibit()

3.3.9.2.6.1. Descriptive Text

This method will inhibit RCL notification for all initiating measurements.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.3.9.2.6.2. Sequence

inhibit<MPS_V41X1513E1_RCL>;

3.3.9.2.7. statusRCL()

3.3.9.2.7.1. Descriptive Text

This method will status all the RCL initiating measurements and output the status (active/inhibited), limit (ON/OFF) and also the CLNotificationFlag.

3.3.9.2.7.2. Sequence

Record Message '<clSequenceId> status is:'
'initiating measurements: ' <V41X1513E1>.getDescription, <V41X1513E1>.getValue()';
'RCL Limits: ' getConstraintInfo<MPS_V41X1513E1_RCL>;
'RCL Notification: ' <CLNotificationFlag>.getValue();
'RCL State: ' MPS_RCL_LO2_PV9_OP_Ind::getState();

3.3.9.2.8. assertRCL()

3.3.9.2.8.1. Descriptive Text

This method will initialize the RCL monitoring for each initiating measurement by accessing the assert() function of the Constraint Manager.

3.3.9.2.8.2. Sequence

// Initialize Constraints

assertAtomicConstraint<V41X1513E1>(ON,MPS_V41X1513E1_RCL) with callback pointer to
MPS_RCL_LO2_PV9_OP_Ind::run();

Record Message '<clSequenceId> Monitoring has been initialized' to <PAGE-A>;

3.3.9.2.9. release()

3.3.9.2.9.1. Descriptive Text

This method will terminate the RCL monitoring for each initiating measurement by accessing the release() function of the Constraint Manager.

3.3.9.2.9.2. Sequence

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

releaseAtomicConstraint<MPS_V41X1513E1_RCL>;

Record Message '<clSequenceId> Monitoring has been terminated' to <PAGE-A>;

3.3.10. MPS_RCL_LO2_PV10_OP_Ind

3.3.10.1. Description

This class Contains the Reactive Control Logic that provides protection for the Orbiter MPS LO2 PV10 Open Indicator. Protection will iniitiate an LO2 stopflow. This class inherits from the class CAS_Control_Logic. Reference - LPS Program GCL55

3.3.10.1.1. Instance Index

Instance	Other Users
MPS_RCL_LO2_PV10_OP_Ind	LO2

3.3.10.1.2. Initiating Measurements

Measurement
<V41X1510E1>

3.3.10.1.3. Class Diagram

<insert class diagram here>

3.3.10.1.4. Operations Index

Operation	Description
run()	This method will perform the execution of the RCL sequence. Notification of the sequence execution will be performed.
determineState()	This method will determine the enumerated state of the RCL sequence based on the associated initiating measurements and will return either: ACTIVE INHIBITED
getState()	This method is inherited from the base class - CAS_Control_Logic
activate()	This method will activate RCL Notification for all unbypassed initiating measurements.
inhibit()	This method will inhibit RCL Notification for all initiating measurements.
statusRCL()	This method will status all the RCL initiating measurements and output the status (active/inhibited), limits (OFF,ON) and also the

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

	CLNotificationFlag.
resetCLNotification()	This method is inherited from the base class - CAS_Control_Logic
assertRCL()	This method will initialize the RCL monitoring for each initiating measurement by accessing the assert() function of the Constraint Manager.
release()	This method will terminate the RCL monitoring for each initiating measurement by accessing the release() function of the Constraint Manager.

3.3.10.1.5. Associations

Association	Type	Class	Description
MPS_RCL_LO2_PV10_OP_Ind	Static	LO2_RCL_StopFlow	Allows: 1. MPS_RCL_LO2_PV10_OP_Ind.run() to call LO2_RCL_StopFlow.runStopFlo

3.3.10.2. Operations

3.3.10.2.1. getState()

3.3.10.2.1.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.3.10.2.2. resetCLNotification()

3.3.10.2.2.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.3.10.2.3. run()

3.3.10.2.3.1. Descriptive Text

This method will perform the execution of the RCL sequence. Notification of the sequence execution will be performed.

3.3.10.2.3.2. Sequence

```
// BEGIN  MPS_RCL_LO2_PV10_OP_Ind  Reactive Control Logic

// CHECK IF PV10 OPEN INDICATOR IS BYPASSED

   IF  <V41X1510E1>.BYP = ON  THEN
    {
      Send Message "<clSequenceId> executed but indicator was Bypassed'
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
                to MPS_RCL message destination;
        Return;
      }



//  VERIFY PV10 OPEN INDICATOR IS NOT ON

    IF  <V41X1510E1> = ON  THEN
      {
       Send Message "<clSequenceId> executed but OP IND currently is ON'
                to MPS_RCL message destination;
        Return;
      }




//  VERIFY LO2 LOADING SEQUENCER IS AUTOFILL OR REPLENISH OR DRAIN

IF  <NLO2_SEQUENCER_STATE>  =   "AUTOFILL"  OR
    <NLO2_OPERATION_STATE>   =   "REPLENISH"  OR
    <NLO2_SEQUENCER_STATE>  =   "AUTODRAIN" THEN


   {

    //  SAFE ORBITER

    //   PV10 OPEN

    Set <V41K1501XL> to ON;
    Set <V41K1502XL> to ON;
    Set <V41K1503NL> to ON;
    Set <V41K1512XL> to OFF;

     // CALL FOR LO2 STOPFLOW

    LO2_RCL_StopFlow.runStopFlow();

     //  SET/SEND NOTIFICATION


      Set <CLNotificationFlag> to ON;

     Send Message "<clSequenceId> Reactive Control Logic executed'
                to MPS_RCL message destination;

}

Return;


//  END REACTIVE SEQUENCE
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.3.10.2.4. determineState()

3.3.10.2.4.1. Descriptive Text

This method will determine the enumerated state of the RCL sequence based on the associated initiating measurements and will return either: ACTIVE OR INHIBITED.

3.3.10.2.4.2. Sequence

IF getConstraintInfo<MPS_V41X1510E1_RCL> = 'ACTIVE' THEN CLState == 'ACTIVE';

IF getConstraintInfo<MPS_V41X1510E1_RCL> = 'INHIBITED' THEN CLState == 'INHIBITED';

3.3.10.2.5. activate()

3.3.10.2.5.1. Descriptive Text

This method will activate RCL notification for all unbypassed initiating measurements.

3.3.10.2.5.2. Sequence

IF <V41X1510E1>.BYP is OFF, activate<MPS_V41X1510E1_RCL>;

3.3.10.2.6. inhibit()

3.3.10.2.6.1. Descriptive Text

This method will inhibit RCL notification for all initiating measurements.

3.3.10.2.6.2. Sequence

inhibit<MPS_V41X1510E1_RCL>;

3.3.10.2.7. statusRCL()

3.3.10.2.7.1. Descriptive Text

This method will status all the RCL initiating measurements and output the status (active/inhibited), limit (ON/OFF) and also the RCL Override.

3.3.10.2.7.2. Sequence

Record Message '<clSequenceId> status is:'
'initiating measurement: ' <V41X1510E1>.get.description, <V41X1510E1>.get.value()';
'RCL Limits: ' getConstraintInfo<MPS_V41X1510E1_RCL>;
'RCL Notification: ' <CLNotification>.getValue();

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

'RCL State: ' MPS_RCL_LO2_PV10_OP_Ind::getState();

3.3.10.2.8. assertRCL()

3.3.10.2.8.1. Descriptive Text

This method will initialize the RCL monitoring for each initiating measurement by accessing the assert() function of the Constraint Manager.

3.3.10.2.8.2. Sequence

// Initialize Constraints

assertAtomicConstraint<V41X1510E1>(ON,MPS_V41X1510E1_RCL) with callback pointer to MPS_RCL_LO2_PV10_OP_Ind::run();

Record Message '<clSequenceId> Monitoring has been initialized' to <PAGE-A>;

3.3.10.2.9. release()

3.3.10.2.9.1. Descriptive Text

This method will terminate the RCL monitoring for each initiating measurement by accessing the release() function of the Constraint Manager.

3.3.10.2.9.2. Sequence

releaseAtomicConstraint<MPS_V41X1510E1_RCL>;

Record Message '<clSequenceId> Monitoring has been terminated' to <PAGE-A>;

3.3.11. MPS_RCL_LO2_PD1_OP_Ind

3.3.11.1. Description

This class Contains the Reactive Control Logic that provides protection for the Orbiter MPS LO2 PD1 Open Indicators.. Protection will iniitiate an LO2 stopflow. This class inherits from the class CAS_Control_Logic. Reference - LPS Program GCL56

3.3.11.1.1. Instance Index

Instance	Other Users
MPS_RCL_LO2_PD1_OP_Ind	LO2

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.3.11.1.2. Initiating Measurements

Measurement
<V41X1529X1>
<V41X1545X1>

3.3.11.1.3. Class Diagram

<insert class diagram here>

3.3.11.1.4. Operations Index

Operation	**Description**
run()	This method will perform the execution of the RCL sequence. Notification of the sequence execution will be performed.
determineState()	This method will determine the enumerated state of the RCL sequence based on the associated initiating measurements and will return either: ACTIVE INHIBITED
getState()	This method is inherited from the base class - CAS_Control_Logic
activate()	This method will activate RCL Notification for all unbypassed initiating measurements.
inhibit()	This method will inhibit RCL Notification for all initiating measurements.
statusRCL()	This method will status all the RCL initiating measurements and output the status (active/inhibited), limits (OFF,ON) and also the CLNotification.
resetCLNotification()	This method is inherited from the base class - CAS_Control_Logic
assertRCL()	This method will initialize the RCL monitoring for each initiating measurement by accessing the assert() function of the Constraint Manager.
release()	This method will terminate the RCL monitoring for each initiating measurement by accessing the release() function of the Constraint Manager.

3.3.11.1.5. Associations

Association	**Type**	**Class**	**Description**
MPS_RCL_LO2_PD1_OP_I	Stati	LO2_RCL_StopFlow	Allows:

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

nd	c		1. MPS_RCL_LO2_PD1_OP_Ind.run() to call LO2_RCL_StopFlow.runStopFlo

3.3.11.2. Operations

3.3.11.2.1. getState()

3.3.11.2.1.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.3.11.2.2. resetCLNotification()

3.3.11.2.2.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.3.11.2.3. run()

3.3.11.2.3.1. Descriptive Text

This method will perform the execution of the RCL sequence. Notification of the sequence execution will be performed.

3.3.11.2.3.2. Sequence

```
// BEGIN  MPS_RCL_LO2_Manifold_Press  Reactive Control Logic


 // VERIFY BOTH PD1 OPEN INDICATORS ARE BYPASSED

   IF  <V41X1529X1>.BYP  = ON AND
       <V41X1545X1>.BYP  = ON  THEN
       {
          Send Message "<clSequenceId> executed but both indicators are Bypassed'
                  to MPS_RCL message destination;
          Return;
       }


 // VERIFY AT LEAST 1 UNBYPASSED PD1 OPEN INDICATORS IS OFF

   IF  <V41X1529X1>.BYP = OFF AND
       <V41X1529X1> = OFF
                    OR
   IF   <V41X1545X1>.BYP  = OFF  AND
        <V41X1545X1>  = OFF THEN

  }


// VERIFY LO2 LOADING SEQUENCER IS REPLENISH

   IF  <NLO2_SEQUENCER_STATE>  =  "AUTOFILL"  AND
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
      <NLO2_OPERATION_STATE>  =   "REPLENISH"  THEN
   {

   IF  <V41X1529X1>.BYP = OFF AND
       <V41X1529X1> = ON  AND
       <V41X1545X1>.BYP = OFF  AND
       <V41X1545X1> = ON
                    OR
       <V41X1529X1>.BYP  = ON AND
       <V41X1545X1>.BYP  = OFF  AND
       <V41X1545X1> = ON
                    OR
       <V41X1529X1>.BYP = OFF  AND
       <V41X1529X1> = ON AND
       <V41X1545X1>.BYP = ON THEN

       {

       // CALL FOR LO2 STOPFLOW

        LO2_RCL_StopFlow.runStopFlow();

       //  SET/SEND NOTIFICATION


       Set <CLNotificationFlag> to ON;

       Send Message "<clSequenceId> Reactive Control Logic executed'
                  to MPS_RCL message destination;
       Return;

       }

       ELSE
       {

       Send Message "<clSequenceId> executed in Replenish but unbypassed OP IND's are currently ON'
                  to MPS_RCL message destination;
          Return;

       }


    }

 ELSE

    {


// VERIFY LO2 LOADING SEQUENCER IS AUTOFILL OR DRAIN

IF  (<NLO2_SEQUENCER_STATE>  =   "AUTOFILL"  AND
   <NLO2_OPERATION_STATE>   !=   "REPLENISH")  OR
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
<NLO2_SEQUENCER_STATE> =  "AUTODRAIN"  THEN

 {

IF   <V41X1529X1>.BYP = OFF AND
     <V41X1529X1>  = OFF
                  OR
     <V41X1545X1>.BYP = OFF  AND
     <V41X1545X1> = OFF  THEN

   {

// CALL FOR LO2 STOPFLOW

 LO2_RCL_StopFlow.runStopFlow();

//  SET/SEND NOTIFICATION

 Set <CLNotificationFlag> to ON;

 Send Message "<clSequenceId> Reactive Control Logic executed'
          to MPS_RCL message destination;

  Return;

   }

 ELSE
   {
  Send Message "<clSequenceId> executed but unbypassed OP IND's are currently ON'
          to MPS_RCL message destination;
   Return;
   }


ELSE

}

Send Message "<clSequenceId> Reactive Control Logic executed but not in Fill/Replenish/Drain'
          to MPS_RCL message destination;


}

Return;


//  END REACTIVE SEQUENCE
```

3.3.11.2.4. determineState()

3.3.11.2.4.1. Descriptive Text

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

This method will determine the enumerated state of the RCL sequence based on the associated initiating measurements and will return either: ACTIVE OR INHIBITED.

3.3.11.2.4.2. Sequence

IF (getConstraintInfo<MPS_V41X1529X1_RCL> = 'ACTIVE' OR
getConstraintInfo<MPS_V41X1545X1_RCL> = 'ACTIVE') THEN CLState == 'ACTIVE';

IF (getConstraintInfo<MPS_V41X1529X1_RCL> = 'INHIBITED' AND
getConstraintInfo<MPS_V41X1545X1_RCL> = 'INHIBITED') THEN CLState == 'INHIBITED';

3.3.11.2.5. activate()

3.3.11.2.5.1. Descriptive Text

This method will activate RCL notification for all unbypassed initiating measurements.

3.3.11.2.5.2. Sequence

IF <V41X1529X1>.BYP is OFF, activate<MPS_V41X1529X1_RCL>;
IF <V41X1545X1>.BYP is OFF, activate<MPS_V41X1545X1_RCL>;

3.3.11.2.6. inhibit()

3.3.11.2.6.1. Descriptive Text

This method will inhibit RCL notification for all initiating measurements.

3.3.11.2.6.2. Sequence

inhibit<MPS_V41X1529X1_RCL>;
inhibit<MPS_V41X1545X1_RCL>;

3.3.11.2.7. statusRCL()

3.3.11.2.7.1. Descriptive Text

This method will status all the RCL initiating measurements and output the status (active/inhibited), limit (ON/OFF) and also the CLNotificationFlag.

3.3.11.2.7.2. Sequence

Record Message '<clSequenceId> status is:'

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

'initiating measurements: ' <V41X1529X1>.getDescription, <V41X1529X1>.getValue'
' <V41X1545X1>.getDescription, <V41X1545X1>.getValue()';
'RCL Limits: ' getConstraintInfo<MPS_V41X1529X1_RCL>;
getConstraintInfo<MPS_V41X1545X1_RCL>;
'RCL Notification: ' <CLNotificationFlag>.getValue();
'RCL State: ' MPS_RCL_LO2_PD1_OP_Ind::getState();

3.3.11.2.8. assertRCL()

3.3.11.2.8.1. Descriptive Text

This method will initialize the RCL monitoring for each initiating measurement by accessing the assert() function of the Constraint Manager.

3.3.11.2.8.2. Sequence

// Initialize Constraints

assertAtomicConstraint<V41X1529X1>(ON,MPS_V41X1529X1_RCL) with callback pointer to
MPS_RCL_LO2_PD1_OP_Ind::run();
assertAtomicConstraint<V41X1545X1>(ON,MPS_V41X1545X1_RCL) with callback pointer to
MPS_RCL_LO2_PD1_OP_Ind::run();

Record Message '<clSequenceId> Monitoring has been initialized' to <PAGE-A>;

3.3.11.2.9. release()

3.3.11.2.9.1. Descriptive Text

This method will terminate the RCL monitoring for each initiating measurement by accessing the release() function of the Constraint Manager.

3.3.11.2.9.2. Sequence

releaseAtomicConstraint<MPS_V41X1529X1_RCL>;
releaseAtomicConstraint<MPS_V41X1545X1_RCL>;

Record Message '<clSequenceId> Monitoring has been terminated' to <PAGE-A>;

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.3.12. MPS_RCL_LH2_PV11_OP_Ind

3.3.12.1. Description

This class Contains the Reactive Control Logic that provides protection for the Orbiter MPS LH2 PV11 Open Indicator. Protection will iniitiate an LH2 stopflow. This class inherits from the class CAS_Control_Logic. Reference - LPS Program GCH56

3.3.12.1.1. Instance Index

Instance	Other Users
MPS_RCL_LH2_PV11_OP_Ind	LH2

3.3.12.1.2. Initiating Measurements

Measurement
<V41X1388E1>

3.3.12.1.3. Class Diagram

<insert class diagram here>

3.3.12.1.4. Operations Index

Operation	Description
run()	This method will perform the execution of the RCL sequence. Notification of the sequence execution will be performed.
determineState()	This method will determine the enumerated state of the RCL sequence based on the associated initiating measurements and will return either: ACTIVE INHIBITED
getState()	This method is inherited from the base class - CAS_Control_Logic
activate()	This method will activate RCL Notification for all unbypassed initiating measurements.
inhibit()	This method will inhibit RCL Notification for all initiating measurements.
setCLOverrideOn()	This method is inherited from the base class - CAS_Control_Logic
setCLOverrideOff()	This method is inherited from the base class - CAS_Control_Logic
statusRCL()	This method will status all the RCL initiating measurements and output the status (active/inhibited), limits (OFF,ON) and also the CLNotificationFlag.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

resetCLNotification()	This method is inherited from the base class - CAS_Control_Logic
assertRCL()	This method will initialize the RCL monitoring for each initiating measurement by accessing the assert() function of the Constraint Manager.
release()	This method will terminate the RCL monitoring for each initiating measurement by accessing the release() function of the Constraint Manager.

3.3.12.1.5. Associations

Association	Type	Class	Description
MPS_RCL_LH2_PV11_OP_Ind	Static	LH2_RCL_StopFlow	Allows: 1. MPS_RCL_LH2_PV11_OP_Ind.run() to call LH2_RCL_StopFlow.runStopflow

3.3.12.2. Operations

3.3.12.2.1. getState()

3.3.12.2.1.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.3.12.2.2. resetCLNotification()

3.3.12.2.2.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.3.12.2.3. run()

3.3.12.2.3.1. Descriptive Text

This method will perform the execution of the RCL sequence. Notification of the sequence execution will be performed.

3.3.12.2.3.2. Sequence

```
// BEGIN  MPS_RCL_LH2_PV11_OP_Ind  Reactive Control Logic


// CHECK IF PV11 OPEN INDICATOR IS BYPASSED

   IF  <V41X1388E1>.BYP = ON  THEN
    {
      Send Message "<clSequenceId> executed but indicator was Bypassed'
            to MPS_RCL message destination;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
        Return;
      }



// VERIFY PV11 OPEN INDICATOR IS NOT ON

    IF  <V41X1388E1> = ON  THEN
      {
       Send Message "<clSequenceId> executed but OP IND currently is ON'
                to MPS_RCL message destination;
        Return;
      }



// VERIFY LH2 LOADING SEQUENCER IS AUTOFILL OR REPLENISH OR DRAIN

IF  <NLH2_SEQUENCER_STATE>  =   "AUTOFILL"  OR
    <NLH2_OPERATION_STATE>   =   "REPLENISH"  OR
    <NLH2_SEQUENCER_STATE>  =   "AUTODRAIN"  THEN
  {

    // SAFE ORBITER

    // PV11 OPEN

    Set <V41K1391XL>  to ON;
    Set <V41K1391NL> to ON;
    Set <V41K1393XL>  to OFF;

     // CALL FOR LH2 STOPFLOW

    LH2_RCL_StopFlow.runStopFlow();


     // SET/SEND NOTIFICATION


      Set <CLNotificationFlag> to ON;

     Send Message "<clSequenceId> Reactive Control Logic executed'
                  to MPS_RCL message destination;

}

Return;


// END REACTIVE SEQUENCE
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.3.12.2.4. determineState()

3.3.12.2.4.1. Descriptive Text

This method will determine the enumerated state of the RCL sequence based on the associated and initiating measurements and will return either: ACTIVE OR INHIBITED.

3.3.12.2.4.2. Sequence

IF getConstraintInfo<MPS_V41X1388E1_RCL> = 'ACTIVE' THEN CLState == 'ACTIVE';

IF getConstraintInfo<MPS_V41X1388E1_RCL> = 'INHIBITED' THEN CLState == 'INHIBITED';

3.3.12.2.5. activate()

3.3.12.2.5.1. Descriptive Text

This method will activate RCL notification for all unbypassed initiating measurements.

3.3.12.2.5.2. Sequence

IF <V41X1388E1>.BYP, activate<MPS_V41X1388E1_RCL>;

3.3.12.2.6. inhibit()

3.3.12.2.6.1. Descriptive Text

This method will inhibit RCL notification for all initiating measurements.

3.3.12.2.6.2. Sequence

inhibit<MPS_V41X1388E1_RCL>;

3.3.12.2.7. statusRCL()

3.3.12.2.7.1. Descriptive Text

This method will status all the RCL initiating measurements and output the status (active/inhibited), limit (ON/OFF) and also the CLNotificationFlag.

3.3.12.2.7.2. Sequence

Record Message '<clSequenceId> status is:'

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

'initiating measurements: ' <V41X1388E1>.get.description, <V41X1388E1>.get.value()';
'RCL Limits: ' getConstraintInfo<MPS_V41X1388E1_RCL>;
'RCL Notification: ' <CLNotificationFlag>.getValue();
'RCL State: ' MPS_RCL_LH2_PV11_OP_Ind::getState();

3.3.12.2.8. assertRCL()

3.3.12.2.8.1. Descriptive Text

This method will initialize the RCL monitoring for each initiating measurement by accessing the assert() function of the Constraint Manager.

3.3.12.2.8.2. Sequence

// Initialize Constraints

assertAtomicConstraint<V41X1388E1>(ON,MPS_V41X1388E1_RCL) with callback pointer to MPS_RCL_LH2_PV11_OP_Ind::run();

Record Message '<clSequenceId> Monitoring has been initialized' to <PAGE-A>;

3.3.12.2.9. release()

3.3.12.2.9.1. Descriptive Text

This method will terminate the RCL monitoring for each initiating measurement by accessing the release() function of the Constraint Manager.

3.3.12.2.9.2. Sequence

releaseAtomicConstraint<MPS_V41X1388E1_RCL>;

Record Message '<clSequenceId> Monitoring has been terminated' to <PAGE-A>;

3.3.13. MPS_RCL_LH2_PV12_OP_Ind

3.3.13.1. Description

This class Contains the Reactive Control Logic that provides protection for the Orbiter MPS LH2 PV12 Open Indicator. Protection will iniitiate an LH2 stopflow. This class inherits from the class CAS_Control_Logic. Reference - LPS Program GCL57

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.3.13.1.1. Instance Index

Instance	Other Users
MPS_RCL_LH2_PV12_OP_Ind	LH2

3.3.13.1.2. Initiating Measurements

Measurement
<V41X1409E1>

3.3.13.1.3. Class Diagram

<insert class diagram here>

3.3.13.1.4. Operations Index

Operation	Description
run()	This method will perform the execution of the RCL sequence. Notification of the sequence execution will be performed.
determineState()	This method will determine the enumerated state of the RCL sequence based on the associated initiating measurements and will return either: ACTIVE INHIBITED
getState()	This method is inherited from the base class - CAS_Control_Logic
activate()	This method will activate RCL Notification for all unbypassed initiating measurements.
inhibit()	This method will inhibit RCL Notification for all initiating measurements.
statusRCL()	This method will status all the RCL initiating measurements and output the status (active/inhibited), limits (OFF,ON) and also the CLNotificationFlag.
resetCLNotification()	This method is inherited from the base class - CAS_Control_Logic
assertRCL()	This method will initialize the RCL monitoring for each initiating measurement by accessing the assert() function of the Constraint Manager.
release()	This method will terminate the RCL monitoring for each initiating measurement by accessing the release() function of the Constraint Manager.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.3.13.1.5. Associations

Association	Type	Class	Description
MPS_RCL_LH2_PV12_OP_Ind	Static	LH2_RCL_StopFlow	Allows: 1. MPS_RCL_LH2_PV12_OP_Ind.run() to call LH2_RCL_Sequencer.runStopFlo

3.3.13.2. Operations

3.3.13.2.1. getState()

3.3.13.2.1.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.3.13.2.2. resetCLNotification()

3.3.13.2.2.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.3.13.2.3. run()

3.3.13.2.3.1. Descriptive Text

This method will perform the execution of the RCL sequence. Notification of the sequence execution will be performed.

3.3.13.2.3.2. Sequence

```
// BEGIN  MPS_RCL_LH2_PV12_OP_Ind  Reactive Control Logic


// CHECK IF PV12 OPEN INDICATOR IS BYPASSED

   IF  <V41X1409E1>.BYP = ON  THEN
    {
     Send Message "<clSequenceId> executed but indicator was Bypassed'
              to MPS_RCL message destination;
     Return;
    }


// VERIFY PV12 OPEN INDICATOR IS NOT ON

   IF  <V41X1409E1>.getValue = ON  THEN
    {
     Send Message "<clSequenceId> executed but OP IND currently is ON'
              to MPS_RCL message destination;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
        Return;
      }


// VERIFY LH2 LOADING SEQUENCER IS AUTOFILL OR  DRAIN

IF  (<NLH2_SEQUENCER_STATE> =  "AUTOFILL"  AND
     <NLH2_OPERATION_STATE>  !=  "REPLENISH")  OR
    <NLH2_SEQUENCER_STATE> =  "AUTODRAIN"  THEN

  {

    //  SAFE ORBITER

    //  PV12  OPEN

    Set <V41K1401XL> to ON;
    Set <V41K1402XL> to ON;
    Set <V41K1403NL> to ON;
    Set <V41K1412XL> to OFF;



     // CALL FOR LH2 STOPFLOW

    LH2_RCL_StopFlow.runStopFlow();


     //  SET/SEND NOTIFICATION


      Set <CLNotificationFlag> to ON;

     Send Message "<clSequenceId> Reactive Control Logic executed'
                  to MPS_RCL message destination;

}

Return;


//  END REACTIVE SEQUENCE
```

3.3.13.2.4. determineState()

3.3.13.2.4.1. Descriptive Text

This method will determine the enumerated state of the RCL sequence based on the associated initiating measurements and will return either: ACTIVE OR INHIBITED.

3.3.13.2.4.2. Sequence

```
IF  getConstraintInfo<MPS_V41X1409E1_RCL> = 'ACTIVE'   THEN  CLState == 'ACTIVE';
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

IF getConstraintInfo<MPS_V41X1409E1_RCL> = 'INHIBITED' THEN CLState == 'INHIBITED';

3.3.13.2.5. activate()

3.3.13.2.5.1. Descriptive Text

This method will activate RCL notification for all unbypassed initiating measurements.

3.3.13.2.5.2. Sequence

IF <V41X1409E1>.BYP is OFF, activate<MPS_V41X1409E1_RCL>;

3.3.13.2.6. inhibit()

3.3.13.2.6.1. Descriptive Text

This method will inhibit RCL notification for all initiating measurements.

3.3.13.2.6.2. Sequence

inhibit<MPS_V41X1409E1_RCL>;

3.3.13.2.7. statusRCL()

3.3.13.2.7.1. Descriptive Text

This method will status all the RCL initiating measurements and output the status (active/inhibited), limit (ON/OFF) and also the RCL Override.

3.3.13.2.7.2. Sequence

Record Message '<clSequenceId> status is:'
'initiating measurements: ' <V41X1409E1>.getdescription, <V41X1409E1>.getvalue()';
'RCL Limits: ' getConstraintInfo<MPS_V41X1409E1_RCL>;
'RCL Notification: ' <CLNotificationFlag>.getValue();
'RCL State: ' MPS_RCL_LH2_PV12_OP_Ind::getState();

3.3.13.2.8. assertRCL()

3.3.13.2.8.1. Descriptive Text

This method will initialize the RCL monitoring for each initiating measurement by accessing the assert() function of the Constraint Manager.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.3.13.2.8.2. Sequence

// Initialize Constraints

assertAtomicConstraint<V41X1409E1>(ON,MPS_V41X1409E1_RCL) with callback pointer to
MPS_RCL_LH2_PV12_OP_Ind::run();

Record Message '<clSequenceId> Monitoring has been initialized' to <PAGE-A>;

3.3.13.2.9. release()

3.3.13.2.9.1. Descriptive Text

This method will terminate the RCL monitoring for each initiating measurement by accessing the release() function of the Constraint Manager.

3.3.13.2.9.2. Sequence

releaseAtomicConstraint<MPS_V41X1409E1_RCL>;

Record Message '<clSequenceId> Monitoring has been terminated' to <PAGE-A>;

3.3.14. MPS_RCL_LH2_PV13_OP_Ind

3.3.14.1. Description

This class Contains the Reactive Control Logic that provides protection for the Orbiter MPS LH2 PV13 Open Indicator. Protection will iniitiate an LH2 stopflow. This class inherits from the class CAS_Control_Logic. Reference - LPS Program GCL58

3.3.14.1.1. Instance Index

Instance	**Other Users**
MPS_RCL_LH2_PV13_OP_Ind	LH2

3.3.14.1.2. Initiating Measurements

Measurement
<V41X1453E1>

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.3.14.1.3. Class Diagram

<insert class diagram here>

3.3.14.1.4. Operations Index

Operation	Description
run()	This method will perform the execution of the RCL sequence. Notification of the sequence execution will be performed.
determineState()	This method will determine the enumerated state of the RCL sequence based on the associated initiating measurements and will return either: ACTIVE INHIBITED
getState()	This method is inherited from the base class - CAS_Control_Logic
activate()	This method will activate RCL Notification for all unbypassed initiating measurements.
inhibit()	This method will inhibit RCL Notification for all initiating measurements.
statusRCL()	This method will status all the RCL initiating measurements and output the status (active/inhibited), limits (OFF,ON) and also the CLNotificationFlag.
resetCLNotification()	This method is inherited from the base class - CAS_Control_Logic
assertRCL()	This method will initialize the RCL monitoring for each initiating measurement by accessing the assert() function of the Constraint Manager.
release()	This method will terminate the RCL monitoring for each initiating measurement by accessing the release() function of the Constraint Manager.

3.3.14.1.5. Associations

Association	Type	Class	Description
MPS_RCL_LH2_PV13_OP_Ind	Static	LH2_RCL_StopFlow	Allows: 1. MPS_RCL_LH2_PV13_OP_Ind.run() to call LH2_RCL_StopFlow.runStopFlo

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.3.14.2. Operations

3.3.14.2.1. getState()

3.3.14.2.1.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.3.14.2.2. resetCLNotification()

3.3.14.2.2.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.3.14.2.3. run()

3.3.14.2.3.1. Descriptive Text

This method will perform the execution of the RCL sequence. Notification of the sequence execution will be performed.

3.3.14.2.3.2. Sequence

```
// BEGIN MPS_RCL_LH2_PV13_OP_Ind Reactive Control Logic


// CHECK IF PV13 OPEN INDICATOR IS BYPASSED

   IF <V41X1453E1>.BYP = ON THEN
    {
     Send Message "<clSequenceId> executed but indicator was Bypassed'
            to MPS_RCL message destination;
     Return;
    }



// VERIFY PV13 OPEN INDICATOR IS NOT ON

   IF  <V41X1453E1> = ON THEN
    {
     Send Message "<clSequenceId> executed but OP IND currently is ON'
            to MPS_RCL message destination;
     Return;
    }



// VERIFY LH2 LOADING SEQUENCER IS AUTOFILL OR REPLENISH OR DRAIN

IF <NLH2_SEQUENCER_STATE> = "AUTOFILL" OR
   <NLH2_OPERATION_STATE> = "REPLENISH" OR
   <NLH2_SEQUENCER_STATE> = "AUTODRAIN" THEN

  {
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
    //  SAFE ORBITER

    //    PV13 OPEN

    Set <V41K1411XL> to ON;

     // CALL FOR LO2 STOPFLOW

    LH2_RCL_StopFlow.runStopFlow();


     //  SET/SEND NOTIFICATION


      Set <CLNotificationFlag> to ON;

     Send Message "<clSequenceId> Reactive Control Logic executed'
                  to MPS_RCL message destination;
}

Return;


//  END REACTIVE SEQUENCE
```

3.3.14.2.4. determineState()

3.3.14.2.4.1. Descriptive Text

This method will determine the enumerated state of the RCL sequence based on the associated initiating measurements and will return either: ACTIVE OR INHIBITED.

3.3.14.2.4.2. Sequence

```
IF  getConstraintInfo<MPS_V41X1453E1_RCL> = 'ACTIVE'  THEN  CLState == 'ACTIVE';

IF  getConstraintInfo<MPS_V41X1453E1_RCL> = 'INHIBITED'  THEN   CLState == 'INHIBITED';
```

3.3.14.2.5. activate()

3.3.14.2.5.1. Descriptive Text

This method will activate RCL notification for all unbypassed initiating measurements.

3.3.14.2.5.2. Sequence

```
IF <V41X1453E1>.BYP,  activate<MPS_V41X1453E1_RCL>;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.3.14.2.6. inhibit()

3.3.14.2.6.1. Descriptive Text

This method will inhibit RCL notification for all initiating measurements.

3.3.14.2.6.2. Sequence

inhibit<MPS_V41X1453E1_RCL>;

3.3.14.2.7. statusRCL()

3.3.14.2.7.1. Descriptive Text

This method will status all the RCL initiating measurements and output the status (active/inhibited), limit (ON/OFF) and also the RCL Override.

3.3.14.2.7.2. Sequence

Record Message '<clSequenceId> status is:'
'initiating measurements: ' <V41X1453E1>.getdescription, <V41X1453E1>.getValue()';
'RCL Limits: ' getConstraintInfo<MPS_V41X1453E1_RCL>;
'RCL Notification: ' <CLNotificationFlag>.getValue();
'RCL State: ' MPS_RCL_LH2_PV13_OP_Ind::getState();

3.3.14.2.8. assertRCL()

3.3.14.2.8.1. Descriptive Text

This method will initialize the RCL monitoring for each initiating measurement by accessing the assert() function of the Constraint Manager.

3.3.14.2.8.2. Sequence

// Initialize Constraints

assertAtomicConstraint<V41X1453E1>(ON,MPS_V41X1453E1_RCL) with callback pointer to
MPS_RCL_LH2_PV13_OP_Ind::run();

Record Message '<clSequenceId> Monitoring has been initialized' to <PAGE-A>;

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.3.14.2.9. release()

3.3.14.2.9.1. Descriptive Text

This method will terminate the RCL monitoring for each initiating measurement by accessing the release() function of the Constraint Manager.

3.3.14.2.9.2. Sequence

releaseAtomicConstraint<MPS_V41X1453E1_RCL>;

Record Message '<clSequenceId> Monitoring has been terminated' to <PAGE-A>;

3.3.15. MPS_RCL_LH2_PD3_OP_Ind

3.3.15.1. Description

This class Contains the Reactive Control Logic that provides protection for the Orbiter MPS LH2 PD3 Open Indicator. Protection will iniitiate an LH2 stopflow. This class inherits from the class CAS_Control_Logic. Reference - LPS Program GCH60

3.3.15.1.1. Instance Index

Instance	**Other Users**
MPS_RCL_LH2_PD3_OP_Ind	LH2

3.3.15.1.2. Initiating Measurements

Measurement
<V41X1419E1>

3.3.15.1.3. Class Diagram

<insert class diagram here>

3.3.15.1.4. Operations Index

Operation	**Description**
run()	This method will perform the execution of the RCL sequence. Notification of the sequence execution will be performed.
determineState()	This method will determine the enumerated state of the RCL sequence based on the associated initiating measurements and will return either: ACTIVE

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

	INHIBITED
getState()	This method is inherited from the base class - CAS_Control_Logic
activate()	This method will activate RCL Notification for all unbypassed initiating measurements.
inhibit()	This method will inhibit RCL Notification for all initiating measurements.
statusRCL()	This method will status all the RCL initiating measurements and output the status (active/inhibited), limits (OFF,ON) and also the CLNotificationFlag.
resetCLNotification()	This method is inherited from the base class - CAS_Control_Logic
assertRCL()	This method will initialize the RCL monitoring for each initiating measurement by accessing the assert() function of the Constraint Manager.
release()	This method will terminate the RCL monitoring for each initiating measurement by accessing the release() function of the Constraint Manager.

3.3.15.1.5. Associations

Association	Type	Class	Description
MPS_RCL_LH2_PD3_OP_Ind	Static	LH2_RCL_StopFlow	Allows: 1. MPS_RCL_LH2_PD3_OP_Ind.run() to call LH2_RCL_StopFlow.runStopFlo

3.3.15.2. Operations

3.3.15.2.1. getState()

3.3.15.2.1.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.3.15.2.2. resetCLNotification()

3.3.15.2.2.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.3.15.2.3. run()

3.3.15.2.3.1. Descriptive Text

This method will perform the execution of the RCL sequence. Notification of the sequence execution will be performed.

3.3.15.2.3.2. Sequence

```
// BEGIN MPS_RCL_LH2_PD3_OP_Ind Reactive Control Logic


// CHECK IF PD3 OPEN INDICATOR IS BYPASSED

   IF <V41X1419E1>.BYP = ON THEN
    {
     Send Message "<clSequenceId> executed but indicator was Bypassed'
            to MPS_RCL message destination;
     Return;
    }


// VERIFY PD3 OPEN INDICATOR IS NOT ON

   IF <V41X1419E1> = ON THEN
    {
     Send Message "<clSequenceId> executed but OP IND currently is ON'
            to MPS_RCL message destination;
     Return;
    }


// VERIFY LH2 LOADING SEQUENCER IS AUTOFILL OR REPLENISH

IF  <NLH2_SEQUENCER_STATE> = "AUTOFILL" OR
   <NLH2_OPERATION_STATE> = "REPLENISH" THEN
 {

   // SAFE ORBITER

   // PD3 OPEN

   Set <V41K1421XL> to ON;
   Set <V41K1422XL> to OFF;

    // CALL FOR LH2 STOPFLOW

   LH2_RCL_StopFlow.runStopFlow();
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
// SET/SEND NOTIFICATION

 Set <CLNotificationFlag> to ON;

Send Message "<clSequenceId> Reactive Control Logic executed'
            to MPS_RCL message destination;

}

Return;

// END REACTIVE SEQUENCE
```

3.3.15.2.4. determineState()

3.3.15.2.4.1. Descriptive Text

This method will determine the enumerated state of the RCL sequence based on the associated initiating measurements and will return either: ACTIVE OR INHIBITED.

3.3.15.2.4.2. Sequence

```
IF getConstraintInfo<MPS_V41X1419E1_RCL> = 'ACTIVE' THEN CLState == 'ACTIVE';

IF getConstraintInfo<MPS_V41X1419E1_RCL> = 'INHIBITED' THEN CLState == 'INHIBITED';
```

3.3.15.2.5. activate()

3.3.15.2.5.1. Descriptive Text

This method will activate RCL notification for all unbypassed initiating measurements.

3.3.15.2.5.2. Sequence

```
IF <V41X1419E1>.BYP, activate<MPS_V41X1419E1_RCL>;
```

3.3.15.2.6. inhibit()

3.3.15.2.6.1. Descriptive Text

This method will inhibit RCL notification for all initiating measurements.

3.3.15.2.6.2. Sequence

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

inhibit<MPS_V41X1419E1_RCL>;

3.3.15.2.7. statusRCL()

3.3.15.2.7.1. Descriptive Text

This method will status all the RCL initiating measurements and output the status (active/inhibited), limit (ON/OFF) and also the CLNotificationFlag.

3.3.15.2.7.2. Sequence

Record Message '<clSequenceId> status is:'
'initiating measurements: ' <V41X1419E1>.get.description, <V41X1419E1>.get.value()';
'RCL Limits: ' getConstraintInfo<MPS_V41X1419E1_RCL>;
'RCL Notification: ' <CLNotificationFlag>.getValue();
'RCL State: ' MPS_RCL_LH2_PD3_OP_Ind::getState();

3.3.15.2.8. assertRCL()

3.3.15.2.8.1. Descriptive Text

This method will initialize the RCL monitoring for each initiating measurement by accessing the assert() function of the Constraint Manager.

3.3.15.2.8.2. Sequence

// Initialize Constraints

assertAtomicConstraint<V41X1419E1>(ON,MPS_V41X1419E1_RCL) with callback pointer to
MPS_RCL_LH2_PD3_OP_Ind::run();

Record Message '<clSequenceId> Monitoring has been initialized' to <PAGE-A>;

3.3.15.2.9. release()

3.3.15.2.9.1. Descriptive Text

This method will terminate the RCL monitoring for each initiating measurement by accessing the release() function of the Constraint Manager.

3.3.15.2.9.2. Sequence

releaseAtomicConstraint<MPS_V41X1419E1_RCL>;

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

Record Message '<clSequenceId> Monitoring has been terminated' to <PAGE-A>;

3.3.16. MPS_RCL_LH2_PD2_OP_Ind

3.3.16.1. Description

This class Contains the Reactive Control Logic that provides protection for the Orbiter MPS LH2 PD2 Open Indicators.. Protection will iniitiate an LH2 stopflow. This class inherits from the class CAS_Control_Logic. Reference - LPS Program GCH59

3.3.16.1.1. Instance Index

Instance	**Other Users**
MPS_RCL_LH2_PD2_OP_Ind	LH2

3.3.16.1.2. Initiating Measurements

Measurement
<V41X1429X1>
<V41X1445X1>

3.3.16.1.3. Class Diagram

<insert class diagram here>

3.3.16.1.4. Operations Index

Operation	**Description**
run()	This method will perform the execution of the RCL sequence. Notification of the sequence execution will be performed.
determineState()	This method will determine the enumerated state of the RCL sequence based on the associated initiating measurements and will return either: ACTIVE INHIBITED
getState()	This method is inherited from the base class - CAS_Control_Logic
activate()	This method will activate RCL Notification for all unbypassed initiating measurements.
inhibit()	This method will inhibit RCL Notification for all initiating measurements.
statusRCL()	This method will status all the RCL initiating measurements and output the status (active/inhibited), limits (OFF,ON) and also the

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

	CLNotificationFlag.
resetCLNotification()	This method is inherited from the base class - CAS_Control_Logic
assertRCL()	This method will initialize the RCL monitoring for each initiating measurement by accessing the assert() function of the Constraint Manager.
release()	This method will terminate the RCL monitoring for each initiating measurement by accessing the release() function of the Constraint Manager.

3.3.16.1.5. Associations

Association	Type	Class	Description
MPS_RCL_LH2_PD2_OP_Ind	Static	LH2_RCL_StopFlow	Allows: 1. MPS_RCL_LH2_PD2_OP_Ind.run() to call LH2_RCL_StopFlow.runStopFlo

3.3.16.2. Operations

3.3.16.2.1. getState()

3.3.16.2.1.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.3.16.2.2. resetCLNotification()

3.3.16.2.2.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.3.16.2.3. run()

3.3.16.2.3.1. Descriptive Text

This method will perform the execution of the RCL sequence. Notification of the sequence execution will be performed.

3.3.16.2.3.2. Sequence

```
// BEGIN  MPS_RCL_LH2_PD2_OP_Ind  Reactive Control Logic

// CHECK IF PD2 OPEN INDICATORS ARE BYPASSED

   IF  <V41X1429X1>.BYP = ON  AND
       <V41X1445X1>.BYP = ON  THEN
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
    {
     Send Message "<clSequenceId> executed but both indicators were Bypassed'
              to MPS_RCL message destination;
     Return;
    }



// VERIFY AT LEAST 1 UNBYPASSED PD2 OPEN INDICATOR IS OFF


   IF  <V41X1429X1>.BYP = OFF AND
       <V41X1429X1> = OFF
                 OR
       <V41X1445X1>.BYP = OFF  AND
       <V41X1445X1> = OFF THEN

     {



// VERIFY LH2 LOADING SEQUENCER IS AUTOFILL OR REPLENISH


IF  (<NLO2_SEQUENCER_STATE>  =   "AUTOFILL"  AND
   <NLO2_OPERATION_STATE>   !=    "REPLENISH")  OR
   <NLO2_SEQUENCER_STATE>  =   "AUTODRAIN"  THEN

       {

        // CALL FOR LH2 STOPFLOW

       LH2_RCL_StopFlow.runStopFlow();


         //  SET/SEND NOTIFICATION


        Set <CLNotificationFlag> to ON;

       Send Message "<clSequenceId> Reactive Control Logic executed'
                 to MPS_RCL message destination;


       Return;
       }

  ELSE

     {

    IF  <NLH2_OPERATION_STATE>   =    "REPLENISH"  THEN

      {
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
      Send Message "<clSequenceId> Reactive Control Logic executed but Loading sequencer in Replenish
                                    - no safing perfromed'  to MPS_RCL message destination;

      Return;
        }
     }

   }


   ELSE

    {
     Send Message "<clSequenceId> executed but unbypassed OP IND's are currently ON'
              to MPS_RCL message destination;
      Return;
    }



// END REACTIVE SEQUENCE
```

3.3.16.2.4. determineState()

3.3.16.2.4.1. Descriptive Text

This method will determine the enumerated state of the RCL sequence based on the associated CLNotificationFlag and initiating measurements and will return either: ACTIVE, INHIBITED OR EXECUTED.

3.3.16.2.4.2. Sequence

```
IF (getConstraintInfo<MPS_V41X1429X1_RCL> = 'ACTIVE' OR
   getConstraintInfo<MPS_V41X1445X1_RCL> = 'ACTIVE')   THEN  CLState == 'ACTIVE';

IF  (getConstraintInfo<MPS_V41X1429X1_RCL> = 'INHIBITED' AND
    getConstraintInfo<MPS_V41X1445X1_RCL> = 'INHIBITED')  THEN  CLState == 'INHIBITED';
```

3.3.16.2.5. activate()

3.3.16.2.5.1. Descriptive Text

This method will activate RCL notification for all unbypassed initiating measurements.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.3.16.2.5.2. Sequence

IF <V41X1429X1>.BYP is OFF, activate<MPS_V41X1429X1_RCL>;
IF <V41X1445X1>.BYP is OFF, activate<MPS_V41X1445X1_RCL>;

3.3.16.2.6. inhibit()

3.3.16.2.6.1. Descriptive Text

This method will inhibit RCL notification for all initiating measurements.

3.3.16.2.6.2. Sequence

inhibit<MPS_V41X1429X1_RCL>;
inhibit<MPS_V41X1445X1_RCL>;

3.3.16.2.7. statusRCL()

3.3.16.2.7.1. Descriptive Text

This method will status all the RCL initiating measurements and output the status (active/inhibited), limit (ON/OFF) and also the RCL Override.

3.3.16.2.7.2. Sequence

Record Message '<clSequenceId> status is:'
'initiating measurements: ' <V41X1429X1>.getDescription, <V41X1429X1>.getValue'
' <V41X1445X1>.getDescription, <V41X1445X1>.getValue()';
'RCL Limits: ' getConstraintInfo<MPS_V41X1429X1_RCL>;
getConstraintInfo<MPS_V41X1445X1_RCL>;
'RCL Notification: ' <CLNotificationFlag>.getValue();
'RCL State: ' MPS_RCL_LH2_PD2_OP_Ind::getState();

3.3.16.2.8. assertRCL()

3.3.16.2.8.1. Descriptive Text

This method will initialize the RCL monitoring for each initiating measurement by accessing the assert() function of the Constraint Manager.

3.3.16.2.8.2. Sequence

// Initialize Constraints

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

assertAtomicConstraint<V41X1429X1>(ON,MPS_V41X1429X1_RCL) with callback pointer to
MPS_RCL_LH2_PD2_OP_Ind::run();
assertAtomicConstraint<V41X1445X1>(ON,MPS_V41X1445X1_RCL) with callback pointer to
MPS_RCL_LH2_PD2_OP_Ind::run();

Record Message '<clSequenceId> Monitoring has been initialized' to <PAGE-A>;

3.3.16.2.9. release()

3.3.16.2.9.1. Descriptive Text

This method will terminate the RCL monitoring for each initiating measurement by accessing the release() function of the Constraint Manager.

3.3.16.2.9.2. Sequence

releaseAtomicConstraint<MPS_V41X1429X1_RCL>;
releaseAtomicConstraint<MPS_V41X1445X1_RCL>;

Record Message '<clSequenceId> Monitoring has been terminated' to <PAGE-A>;

3.4. PREREQUISITE CONTROL LOGIC CLASSES

3.4.1. MPS_PCL_PV9_CL_CMD

3.4.1.1. Description

This PCL provides protection for the MPS LO2 Outboard Fill and Drain Valve Close Command.
This class inherits from the class CAS_Control_Logic. The Algorithm will be implemented directly in C++ Code.
Reference - LPS Program GCL18/23

3.4.1.1.1. Instance

Instance Name
MPS_PCL_PV9_CL_CMD

3.4.1.1.2. PCL Initiating Commands

Command
V41K1515XL

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.4.1.1.3. Operations Index

Operation	Description
determineState()	This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE INHIBITED ERROR
activate()	This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.
inhibit()	This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.
getState()	This method is inherited from the base class - CAS_Control_Logic
resetCLNotification()	This method is inherited from the base class - CAS_Control_Logic

3.4.1.2. Operations

3.4.1.2.1. getState()

3.4.1.2.1.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.1.2.2. resetCLNotification()

3.4.1.2.2.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.1.2.3. determineState()

3.4.1.2.3.1. Descriptive Text

This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE, INHIBITED or ERROR.

3.4.1.2.3.2. Sequence

```
if (<V41K1515XL>.PCLSTAT = active)
  {
  CLState = ACTIVE;
  }
else
  {
   if (<V41K1515XL>.PCLSTAT = inhibited)
     {
     CLState = INHIBITED;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
    }
  else
    {
   CLState = ERROR;
    }
 }
```

3.4.1.2.4. activate()

3.4.1.2.4.1. Descriptive Text

This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.1.2.4.2. Sequence

A FD <V41K1515XL> PCL;

return;

3.4.1.2.5. inhibit()

3.4.1.2.5.1. Descriptive Text

This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.1.2.5.2. Sequence

I FD <V41K1515XL> PCL;

return;

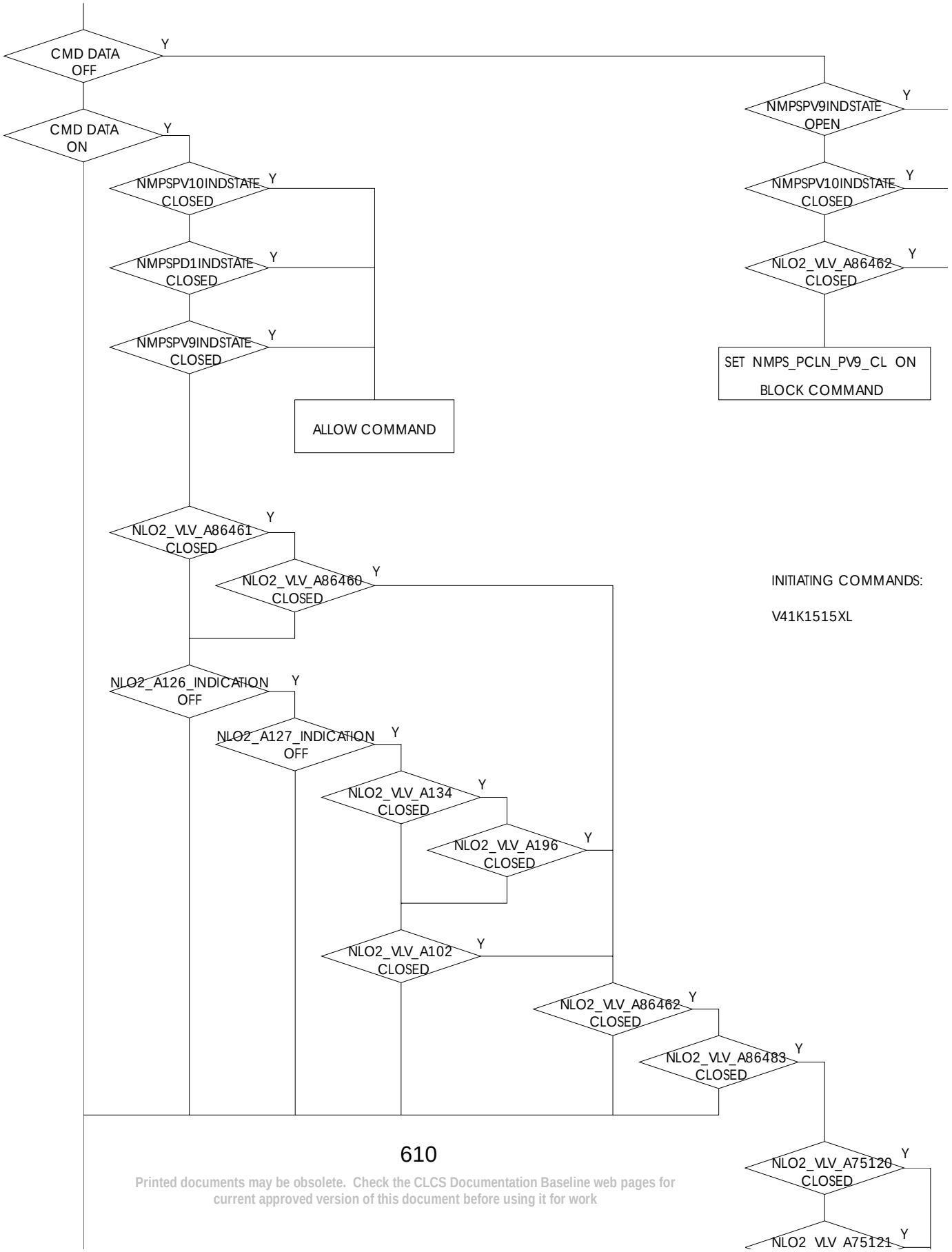
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.4.1.3. Algorithm

3.4.1.3.1. PCL Algorithm

3.4.1.3.1.1. Flowchart

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.4.1.3.1.2. Sequence

```
#include "mps_pcl_pv9_cl_cmd.h"
#include "SME.h"
#include "LO2.h"
#include "MPS.h"
#include "LH2.h"

static const char* seqName = "MPS_PCL_PV9_CL_CMD";
static const char* algorithmDescription = "Revision Date - 01/20/2000\n\
if (A) ||\n\
if Command_Data is ON &&\n\
  if (K || L || M || (((B && C || (D && E && ((F && G) || H))) && I && J) && (N || O) && (P || Q))) ||\n\
if Command_Data is OFF &&\n\
  if (R || K || I)\n\
where:   A  = NMPS_PCLO_PV9_CL - PCL Bypass ON\n\
         B  = A86461 - LO2 XFER LINE FILL VALVE - CLOSED\n\
         C  = A86460 - LO2 REPLENISH VALVE - CLOSED\n\
         D  = A126 - LO2 PUMP - OFF\n\
         E  = A127 - LO2 PUMP - OFF\n\
         F  = A134 - LO2 XFER LINE DRAIN VALVE - CLOSED\n\
         G = A196 - LO2 BYPASS VALVE - CLOSED\n\
         H = A102 - LO2 DRAIN LINE SOV - CLOSED\n\
         I   = A86462 - LO2 TSM DRAIN VALVE - CLOSED\n\
         J  = A86463 LO2 TSM VENT VALVE - CLOSED\n\
         K = PV10 - LO2 INBOARD F/D VALVE - CLOSED\n\
         L  = PD1 - LO2 ORB/ET FEED DISC VALVE - CLOSED\n\
         M = PV9 - LO2 OUTOARD F/D VALVE - CLOSED\n\
         N = A75120 - LO2 PRG SOV - CLOSED\n\
         O = A75121 - LO2 PRG SOV - CLOSED\n\
         P = A75127 - LO2 LK/CK SOV - CLOSED\n\
         Q = A75128 - LO2 LK/CK SOV - CLOSED\n\
         R = PV9 - LO2 OUTBOARD F/D VALVE - OPEN";

MPS_PCL_PV9_CL_CMD::MPS_PCL_PV9_CL_CMD(const char*
seqName):asv_pcsVPCL_DS_ON_OFF_(seqName)
{
  userInit();
}

MPS_PCL_PV9_CL_CMD::~MPS_PCL_PV9_CL_CMD()
{
}


void MPS_PCL_PV9_CL_CMD::userInit(void)
{
  initiatingFD(V41K1515XL);
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
}

int MPS_PCL_PV9_CL_CMD::userExecute(Discrete::ON_OFF dsVar)
{

 if (Discrete::On == (NMPS_PCLO_PV9_CL.getValue()))
{
 return PCL_ALLOW_CMD;
}

 if (Discrete::On==dsVar)
{
  if (K() || L() || M() || (((B()&&C() || (D()&&E()&&((F()&&G()) || H())))
      &&I()&&J())&&(N() || O())&&(P() || Q())))
  {
   return PCL_ALLOW_CMD;
  }
  else
  {
   NMPS_PCLN_PV9_CL.Set(Discrete::On);
   return PCL_BLOCK_CMD;
  }
}
if (Discrete::Off==dsVar)
{
if (R() || K() || I())
  {
   return PCL_ALLOW_CMD;
  }
else
  {
   NMPS_PCLN_PV9_CL.Set(Discrete::On);
   return PCL_BLOCK_CMD;
  }
}

}




// A86461 - LO2 Xfer line fill valve - CLOSED

int MPS_PCL_PV9_CL_CMD::B()
{
int  valReturnB = PCL_VOTER_FAIL;

try
{
            if (NLO2_VLV_A86461.getValue() == LH2_VALVE_STATE::CLOSED)
    {
              valReturnB =  PCL_VOTER_PASS;
             }
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
else
            {
       valReturnB = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnB = PCL_VOTER_FAIL;
}

return valReturnB;
}




// A86460 - LO2 Replenish valve - CLOSED

int MPS_PCL_PV9_CL_CMD::C()
{
int  valReturnC = PCL_VOTER_FAIL;

try
{
            if (NLO2_VLV_A86460.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnC = PCL_VOTER_PASS;
             }
else
            {
       valReturnC = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnC = PCL_VOTER_FAIL;
}

  return valReturnC;
}




// A126 - LO2 PUMP - OFF

int MPS_PCL_PV9_CL_CMD::D()
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
{
  int  valReturnD = PCL_VOTER_FAIL;

 try
{
            if (NLO2_PUMP_A126.getValue() == Discrete::Off)
     {
              valReturnD =  PCL_VOTER_PASS;
             }
else
            {
       valReturnD = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnD =  PCL_VOTER_FAIL;
}

return valReturnD;
}


// A127 - LO2 PUMP - OFF

int MPS_PCL_PV9_CL_CMD::E()
{
  int  valReturnE = PCL_VOTER_FAIL;

 try
{
            if (NLO2_PUMP_A127.getValue() == Discrete::Off)
     {
              valReturnE = PCL_VOTER_PASS;
             }
else
            {
       valReturnE = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnE = PCL_VOTER_FAIL;
}

  return valReturnE;
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
// A134 - LO2 Xfer line drain valve - CLOSED

int MPS_PCL_PV9_CL_CMD::F()
{
  int  valReturnF = PCL_VOTER_FAIL;

try
{
            if (NLO2_VLV_A134.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnF = PCL_VOTER_PASS;
             }
else
            {
       valReturnF = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnF = PCL_VOTER_FAIL;
}

  return valReturnF;
}


// A196 - LO2 bypass valve - CLOSED

int MPS_PCL_PV9_CL_CMD::G()
{
  int  valReturnG = PCL_VOTER_FAIL;

try
{
            if (NLO2_VLV_A196.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnG = PCL_VOTER_PASS;
             }
else
            {
       valReturnG = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnG = PCL_VOTER_FAIL;
}

  return valReturnG;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
}




// A102 - LO2 Drain line SOV - CLOSED

int MPS_PCL_PV9_CL_CMD::H()
{
  int  valReturnH = PCL_VOTER_FAIL;

 try
{
            if (NLO2_VLV_A102.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnH = PCL_VOTER_PASS;
             }
else
            {
       valReturnH = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnH = PCL_VOTER_FAIL;
}

  return valReturnH;
}




// A86462 - LO2 TSM drain valve - CLOSED

int MPS_PCL_PV9_CL_CMD::I()
{
  int  valReturnI = PCL_VOTER_FAIL;

 try
{
            if (NLO2_VLV_A86462.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnI = PCL_VOTER_PASS;
             }
else
            {
       valReturnI = PCL_VOTER_FAIL;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
            }
}

catch(...)
{
    valReturnI = PCL_VOTER_FAIL;
}



 return valReturnI;
}



// A86483 - LO2 TSM vent valve - CLOSED

int MPS_PCL_PV9_CL_CMD::J()
{
 int  valReturnJ = PCL_VOTER_FAIL;

try
{
            if (NLO2_VLV_A86483.getValue() == LH2_VALVE_STATE::CLOSED)
    {
              valReturnJ = PCL_VOTER_PASS;
             }
else
            {
      valReturnJ = PCL_VOTER_FAIL;
            }
}

catch(...)
{
    valReturnJ = PCL_VOTER_FAIL;
}

 return valReturnJ;
}



// PV10 - LO2 Inboard F/D Valve - CLOSED

int MPS_PCL_PV9_CL_CMD::K()
{
 int  valReturnK = PCL_VOTER_FAIL;

try
{
            if (NMPSPV10INDSTATE.getValue() == MPS_VALVE_IND_STATE::CLOSED)
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
    {
            valReturnK = PCL_VOTER_PASS;
           }
else
           {
      valReturnK = PCL_VOTER_FAIL;
           }
}

catch(...)
{
    valReturnK = PCL_VOTER_FAIL;
}

 return valReturnK;
}
```

```
//PD1 - LO2 ORB/ET Feed Disc Valve - CLOSED

int MPS_PCL_PV9_CL_CMD::L()
{
 int  valReturnL = PCL_VOTER_FAIL;

try
{
           if (NMPSPD1INDSTATE.getValue() == MPS_VALVE_IND_STATE::CLOSED)
    {
            valReturnL = PCL_VOTER_PASS;
           }
else
           {
      valReturnL = PCL_VOTER_FAIL;
           }
}

catch(...)
{
    valReturnL = PCL_VOTER_FAIL;
}

 return valReturnL;
}
```

```
// PV9 - LO2 Outboard F/D Valve - CLOSED

int MPS_PCL_PV9_CL_CMD::M()
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
{
  int  valReturnM = PCL_VOTER_FAIL;

try
{
            if (NMPSPV9INDSTATE.getValue() == MPS_VALVE_IND_STATE::CLOSED)
     {
              valReturnM = PCL_VOTER_PASS;
             }
else
            {
       valReturnM = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnM = PCL_VOTER_FAIL;
}

  return valReturnM;
}




// A75120 - PRG SOV - CLOSED

int MPS_PCL_PV9_CL_CMD::N()
{
  int  valReturnN = PCL_VOTER_FAIL;

try
{
            if (NLO2_VLV_A75120.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnN = PCL_VOTER_PASS;
             }
else
            {
       valReturnN = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnN = PCL_VOTER_FAIL;
}

return valReturnN;
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
// A75121 - PRG SOV - CLOSED

int MPS_PCL_PV9_CL_CMD::O()
{
  int  valReturnO = PCL_VOTER_FAIL;

try
{
            if (NLO2_VLV_A75121.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnO = PCL_VOTER_PASS;
             }
else
            {
       valReturnO = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnO = PCL_VOTER_FAIL;
}



  return valReturnO;
}




// A75127 - LK/CK VLV - CLOSED

int MPS_PCL_PV9_CL_CMD::P()
{
  int  valReturnP = PCL_VOTER_FAIL;

 try
{
            if (NLO2_VLV_A75127.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnP = PCL_VOTER_PASS;
             }
else
            {
       valReturnP = PCL_VOTER_FAIL;
            }
}

catch(...)
{
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
    valReturnP = PCL_VOTER_FAIL;
}

 return valReturnP;
}



// A75128 - LK/CK VLV - CLOSED

int MPS_PCL_PV9_CL_CMD::Q()
{
 int  valReturnQ = PCL_VOTER_FAIL;

try
{
          if (NLO2_VLV_A75128.getValue() == LH2_VALVE_STATE::CLOSED)
    {
            valReturnQ = PCL_VOTER_PASS;
           }
else
          {
      valReturnQ = PCL_VOTER_FAIL;
          }
}

catch(...)
{
    valReturnQ = PCL_VOTER_FAIL;
}

 return valReturnQ;
}




// PV9 - LO2 Inboard F/D Valve - OPEN

int MPS_PCL_PV9_CL_CMD::R()
{
 int  valReturnR = PCL_VOTER_FAIL;

try
{
          if (NMPSPV9INDSTATE.getValue() == MPS_VALVE_IND_STATE::OPEN)
    {
            valReturnR = PCL_VOTER_PASS;
           }
else
          {
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
        valReturnR = PCL_VOTER_FAIL;
            }
}

catch(...)
{
    valReturnR = PCL_VOTER_FAIL;
}

  return valReturnR;
}


MPS_PCL_PV9_CL_CMD mps_pcl_pv9_cl_cmd(seqName);
```

3.4.2. MPS_PCL_PV9_OP_CMD

3.4.2.1. Description

This PCL provides protection for the MPS LO2 Outboard Fill and Drain Valve Open Commands.
This class inherits from the class CAS_Control_Logic. The Algorithm will be implemented directly in C++ Code.
Reference - LPS Program GCL18/23

3.4.2.1.1. Instance

Instance Name
MPS_PCL_PV9_OP_CMD

3.4.2.1.2. PCL Initiating Commands

Command
V41K1518XL
V41K1518NL

3.4.2.1.3. Operations Index

Operation	**Description**

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

determineState()	This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE INHIBITED ERROR
activate()	This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.
inhibit()	This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.
getState()	This method is inherited from the base class - CAS_Control_Logic
resetCLNotification()	This method is inherited from the base class - CAS_Control_Logic

3.4.2.2. Operations

3.4.2.2.1. getState()

3.4.2.2.1.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.2.2.2. resetCLNotification()

3.4.2.2.2.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.2.2.3. determineState()

3.4.2.2.3.1. Descriptive Text

This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE, INHIBITED or ERROR.

3.4.2.2.3.2. Sequence

```
if (<V41K1518XL>.PCLSTAT = active  AND
   <V41K1518NL>.PCLSTAT = active)
  {
  CLState = ACTIVE;
  }
else
  {

   if (<V41K1518XL>.PCLSTAT = inhibited AND
      <V41K1518NL>.PCLSTAT = inhibited)
    {
    CLState = INHIBITED;
     }
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
   else
     {
    CLState = ERROR;
     }
  }
```

3.4.2.2.4. activate()

3.4.2.2.4.1. Descriptive Text

This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.2.2.4.2. Sequence

A FD <V41K1518XL> PCL;
A FD <V41K1518NL> PCL;

return;

3.4.2.2.5. inhibit()

3.4.2.2.5.1. Descriptive Text

This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.2.2.5.2. Sequence

I FD <V41K1518XL> PCL;
I FD <V41K1518NL> PCL;

return;

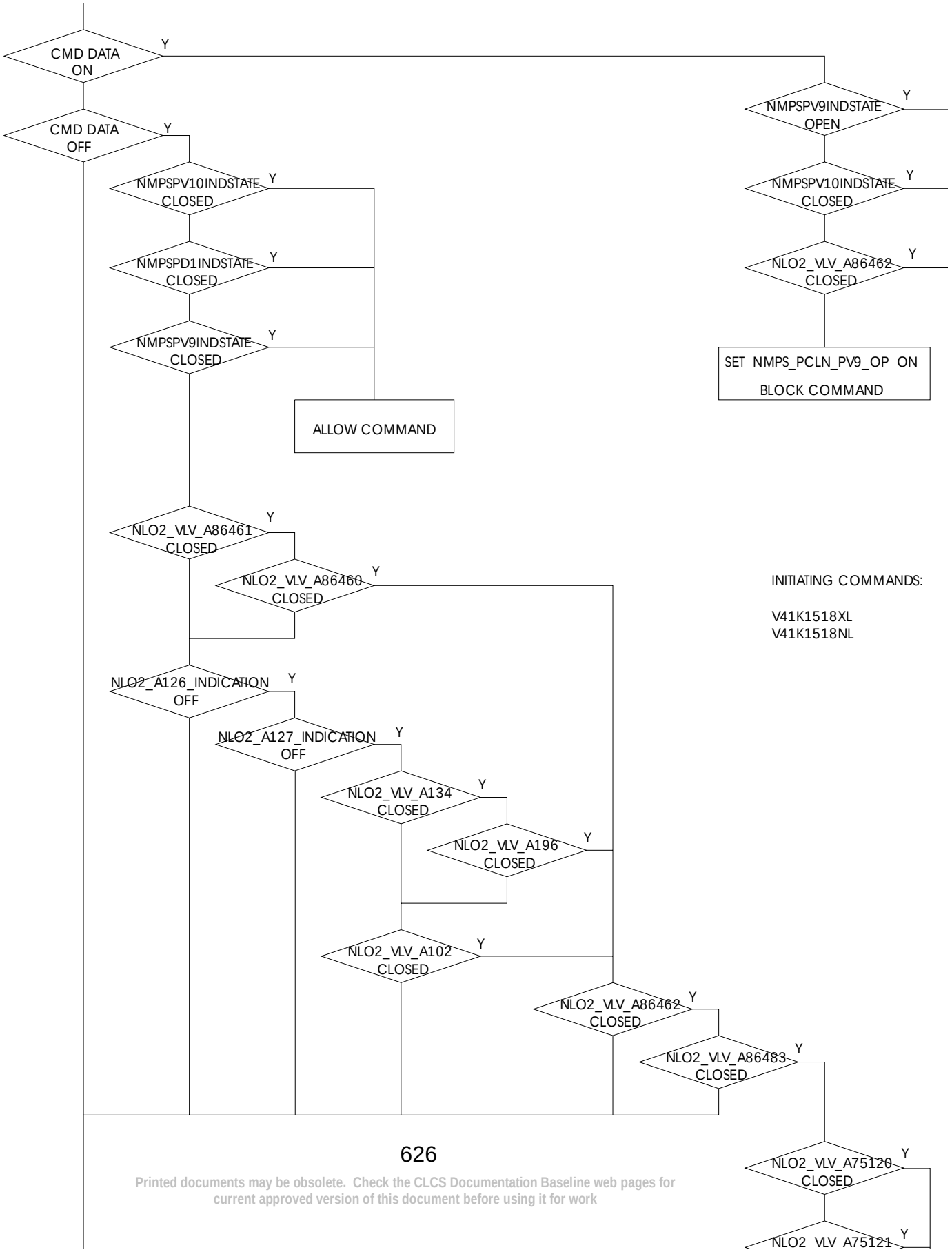
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.4.2.3. Algorithm

3.4.2.3.1. PCL Algorithm

3.4.2.3.1.1. Flowchart

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.4.2.3.1.2. Sequence

```
#include "mps_pcl_pv9_op_cmd.h"
#include "SME.h"
#include "LO2.h"
#include "MPS.h"
#include "LH2.h"

static const char* seqName = "MPS_PCL_PV9_OP_CMD";
static const char* algorithmDescription = "Revision Date - 01/20/2000\n\
if (A) ||\n\
if Command_Data is OFF &&\n\
  if (K || L || M || (((B && C || (D && E && ((F && G) || H))) && I && J) && (N || O) && (P || Q))) ||\n\
if Command_Data is ON &&\n\
  if (R || K || I)\n\
where:   A  = NMPS_PCLO_PV9_OP - PCL Bypass ON\n\
         B  = A86461 - LO2 XFER LINE FILL VALVE - CLOSED\n\
         C  = A86460 - LO2 REPLENISH VALVE - CLOSED\n\
         D  = A126 - LO2 PUMP - OFF\n\
         E  = A127 - LO2 PUMP - OFF\n\
         F  = A134 - LO2 XFER LINE DRAIN VALVE - CLOSED\n\
         G = A196 - LO2 BYPASS VALVE - CLOSED\n\
         H = A102 - LO2 DRAIN LINE SOV - CLOSED\n\
         I   = A86462 - LO2 TSM DRAIN VALVE - CLOSED\n\
         J  = A86463 LO2 TSM VENT VALVE - CLOSED\n\
         K = PV10 - LO2 INBOARD F/D VALVE - CLOSED\n\
         L  = PD1 - LO2 ORB/ET FEED DISC VALVE - CLOSED\n\
         M = PV9 - LO2 OUTOARD F/D VALVE - CLOSED\n\
         N = A75120 - LO2 PRG SOV - CLOSED\n\
         O = A75121 - LO2 PRG SOV - CLOSED\n\
         P = A75127 - LO2 LK/CK SOV - CLOSED\n\
         Q = A75128 - LO2 LK/CK SOV - CLOSED\n\
         R = PV9 - LO2 OUTBOARD F/D VALVE - OPEN";

MPS_PCL_PV9_OP_CMD::MPS_PCL_PV9_OP_CMD(const char*
seqName):asv_pcsVPCL_DS_ON_OFF_(seqName)
{
  userInit();
}

MPS_PCL_PV9_OP_CMD::~MPS_PCL_PV9_OP_CMD()
{
}


void MPS_PCL_PV9_OP_CMD::userInit(void)
{
  initiatingFD(V41K1518XL);
  initiatingFD(V41K1518NL);
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
int MPS_PCL_PV9_OP_CMD::userExecute(Discrete::ON_OFF dsVar)
{

 if (Discrete::On == (NMPS_PCLO_PV9_OP.getValue()))
{
 return PCL_ALLOW_CMD;
}

 if (Discrete::Off==dsVar)
{
  if (K() || L() || M() || (((B()&&C() || (D()&&E()&&((F()&&G()) || H())))
      &&I()&&J())&&(N() || O())&&(P() || Q())))
  {
   return PCL_ALLOW_CMD;
  }
  else
  {
   NMPS_PCLN_PV9_OP.Set(Discrete::On);
   return PCL_BLOCK_CMD;
  }
}
if (Discrete::On==dsVar)
{
if (R() || K() || I())
  {
   return PCL_ALLOW_CMD;
  }
else
  {

   NMPS_PCLN_PV9_OP.Set(Discrete::On);
   return PCL_BLOCK_CMD;
  }
}

}




// A86461 - LO2 Xfer line fill valve - CLOSED

int MPS_PCL_PV9_OP_CMD::B()
{
int  valReturnB = PCL_VOTER_FAIL;

try
{
            if (NLO2_VLV_A86461.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnB = PCL_VOTER_PASS;
             }
else
            {
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
       valReturnB = PCL_VOTER_FAIL;
            }
}

catch(...)
{
    valReturnB = PCL_VOTER_FAIL;
}

return valReturnB;
}
```

```
// A86460 - LO2 Replenish valve - CLOSED

int MPS_PCL_PV9_OP_CMD::C()
{
int  valReturnC = PCL_VOTER_FAIL;

try
{
            if (NLO2_VLV_A86460.getValue() == LH2_VALVE_STATE::CLOSED)
    {
              valReturnC = PCL_VOTER_PASS;
             }
else
            {
       valReturnC = PCL_VOTER_FAIL;
            }
}

catch(...)
{
    valReturnC = PCL_VOTER_FAIL;
}

  return valReturnC;
}
```

```
// A126 - LO2 PUMP - OFF

int MPS_PCL_PV9_OP_CMD::D()
{
  int  valReturnD = PCL_VOTER_FAIL;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
 try
{
            if (NLO2_PUMP_A126.getValue() == Discrete::Off)
     {
              valReturnD = PCL_VOTER_PASS;
             }
else
            {
       valReturnD = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnD = PCL_VOTER_FAIL;
}

return valReturnD;
}


// A127 - LO2 PUMP - OFF

int MPS_PCL_PV9_OP_CMD::E()
{
  int  valReturnE = PCL_VOTER_FAIL;

 try
{
            if (NLO2_PUMP_A127.getValue() == Discrete::Off)
     {
              valReturnE = PCL_VOTER_PASS;
             }
else
            {
       valReturnE = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnE = PCL_VOTER_FAIL;
}

  return valReturnE;
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
// A134 - LO2 Xfer line drain valve - CLOSED

int MPS_PCL_PV9_OP_CMD::F()
{
  int  valReturnF = PCL_VOTER_FAIL;

try
{
            if (NLO2_VLV_A134.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnF = PCL_VOTER_PASS;
             }
else
            {
       valReturnF = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnF = PCL_VOTER_FAIL;
}

  return valReturnF;
}


// A196 - LO2 bypass valve - CLOSED

int MPS_PCL_PV9_OP_CMD::G()
{
  int  valReturnG = PCL_VOTER_FAIL;

try
{
            if (NLO2_VLV_A196.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnG = PCL_VOTER_PASS;
             }
else
            {
       valReturnG = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnG = PCL_VOTER_FAIL;
}

  return valReturnG;
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
// A102 - LO2 Drain line SOV - CLOSED

int MPS_PCL_PV9_OP_CMD::H()
{
  int  valReturnH = PCL_VOTER_FAIL;

 try
{
            if (NLO2_VLV_A102.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnH = PCL_VOTER_PASS;
             }
else
            {
       valReturnH = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnH = PCL_VOTER_FAIL;
}

  return valReturnH;
}




// A86462 - LO2 TSM drain valve - CLOSED

int MPS_PCL_PV9_OP_CMD::I()
{
  int  valReturnI = PCL_VOTER_FAIL;

 try
{
            if (NLO2_VLV_A86462.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnI = PCL_VOTER_PASS;
             }
else
            {
       valReturnI = PCL_VOTER_FAIL;
            }
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
catch(...)
{
    valReturnI = PCL_VOTER_FAIL;
}



 return valReturnI;
}



// A86483 - LO2 TSM vent valve - CLOSED

int MPS_PCL_PV9_OP_CMD::J()
{
 int  valReturnJ = PCL_VOTER_FAIL;

try
{
            if (NLO2_VLV_A86483.getValue() == LH2_VALVE_STATE::CLOSED)
    {
              valReturnJ = PCL_VOTER_PASS;
             }
else
            {
      valReturnJ = PCL_VOTER_FAIL;
            }
}

catch(...)
{
    valReturnJ = PCL_VOTER_FAIL;
}

 return valReturnJ;
}



// PV10 - LO2 Inboard F/D Valve - CLOSED

int MPS_PCL_PV9_OP_CMD::K()
{
 int  valReturnK = PCL_VOTER_FAIL;

try
{
            if (NMPSPV10INDSTATE.getValue() == MPS_VALVE_IND_STATE::CLOSED)
    {
              valReturnK = PCL_VOTER_PASS;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
            }
else
            {
       valReturnK = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnK = PCL_VOTER_FAIL;
}

  return valReturnK;
}
```

```
//PD1 - LO2 ORB/ET Feed Disc Valve - CLOSED

int MPS_PCL_PV9_OP_CMD::L()
{
  int  valReturnL = PCL_VOTER_FAIL;

try
{
            if (NMPSPD1INDSTATE.getValue() == MPS_VALVE_IND_STATE::CLOSED)
     {
              valReturnL = PCL_VOTER_PASS;
             }
else
            {
       valReturnL = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnL = PCL_VOTER_FAIL;
}

  return valReturnL;
}
```

```
// PV9 - LO2 Outboard F/D Valve - CLOSED

int MPS_PCL_PV9_OP_CMD::M()
{
  int  valReturnM = PCL_VOTER_FAIL;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
try
{
            if (NMPSPV9INDSTATE.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnM = PCL_VOTER_PASS;
             }
else
            {
       valReturnM = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnM = PCL_VOTER_FAIL;
}

  return valReturnM;
}




// A75120 - PRG SOV - CLOSED

int MPS_PCL_PV9_OP_CMD::N()
{
  int  valReturnN = PCL_VOTER_FAIL;

try
{
            if (NLO2_VLV_A75120.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnN = PCL_VOTER_PASS;
             }
else
            {
       valReturnN = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnN = PCL_VOTER_FAIL;
}

return valReturnN;
}


// A75121 - PRG SOV - CLOSED
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
int MPS_PCL_PV9_OP_CMD::O()
{
  int  valReturnO = PCL_VOTER_FAIL;

try
{
            if (NLO2_VLV_A75121.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnO = PCL_VOTER_PASS;
             }
else
            {
       valReturnO = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnO = PCL_VOTER_FAIL;
}

  return valReturnO;
}




// A75127 - LK/CK VLV - CLOSED

int MPS_PCL_PV9_OP_CMD::P()
{
  int  valReturnP = PCL_VOTER_FAIL;

 try
{
            if (NLO2_VLV_A75127.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnP = PCL_VOTER_PASS;
             }
else
            {
       valReturnP = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnP = PCL_VOTER_FAIL;
}

  return valReturnP;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
}



// A75128 - LK/CK VLV - CLOSED

int MPS_PCL_PV9_OP_CMD::Q()
{
  int  valReturnQ = PCL_VOTER_FAIL;

try
{
            if (NLO2_VLV_A75128.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnQ = PCL_VOTER_PASS;
             }
else
            {
       valReturnQ = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnQ = PCL_VOTER_FAIL;
}

  return valReturnQ;
}




// PV9 - LO2 Inboard F/D Valve - OPEN

int MPS_PCL_PV9_OP_CMD::R()
{
  int  valReturnR = PCL_VOTER_FAIL;

try
{
            if (NMPSPV9INDSTATE.getValue() == MPS_VALVE_IND_STATE::OPEN)
     {
              valReturnR = PCL_VOTER_PASS;
             }
else
            {
       valReturnR = PCL_VOTER_FAIL;
            }
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
catch(...)
{
    valReturnR = PCL_VOTER_FAIL;
}

 return valReturnR;
}


MPS_PCL_PV9_OP_CMD mps_pcl_pv9_op_cmd(seqName);
```

3.4.3. MPS_PCL_PV10_CL_CMD

3.4.3.1. Description

This PCL provides protection for the MPS LO2 Inboard Fill and Drain Valve Close Command.
This class inherits from the class CAS_Control_Logic. The Algorithm will be implemented directly in C++ Code.
Reference - LPS Program GCL19/24

3.4.3.1.1. Instance

Instance Name
MPS_PCL_PV10_CL_CMD

3.4.3.1.2. PCL Initiating Commands

Command
V41K1512XL

3.4.3.1.3. Operations Index

Operation	**Description**
determineState()	This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE INHIBITED

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

	ERROR
activate()	This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.
inhibit()	This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.
getState()	This method is inherited from the base class - CAS_Control_Logic
resetCLNotification()	This method is inherited from the base class - CAS_Control_Logic

3.4.3.2. Operations

3.4.3.2.1. getState()

3.4.3.2.1.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.3.2.2. resetCLNotification()

3.4.3.2.2.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.3.2.3. determineState()

3.4.3.2.3.1. Descriptive Text

This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE, INHIBITED or ERROR.

3.4.3.2.3.2. Sequence

```
if (<V41K1512XL>.PCLSTAT = active)
  {
  CLState = ACTIVE;
  }
else
  {
   if (<V41K1512XL>.PCLSTAT = inhibited)
     {
     CLState = INHIBITED;
      }
    else
     {
     CLState = ERROR;
      }
   }
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.4.3.2.4. activate()

3.4.3.2.4.1. Descriptive Text

This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.3.2.4.2. Sequence

A FD <V41K1512XL> PCL;

return;

3.4.3.2.5. inhibit()

3.4.3.2.5.1. Descriptive Text

This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.3.2.5.2. Sequence

I FD <V41K1512XL> PCL;

return;

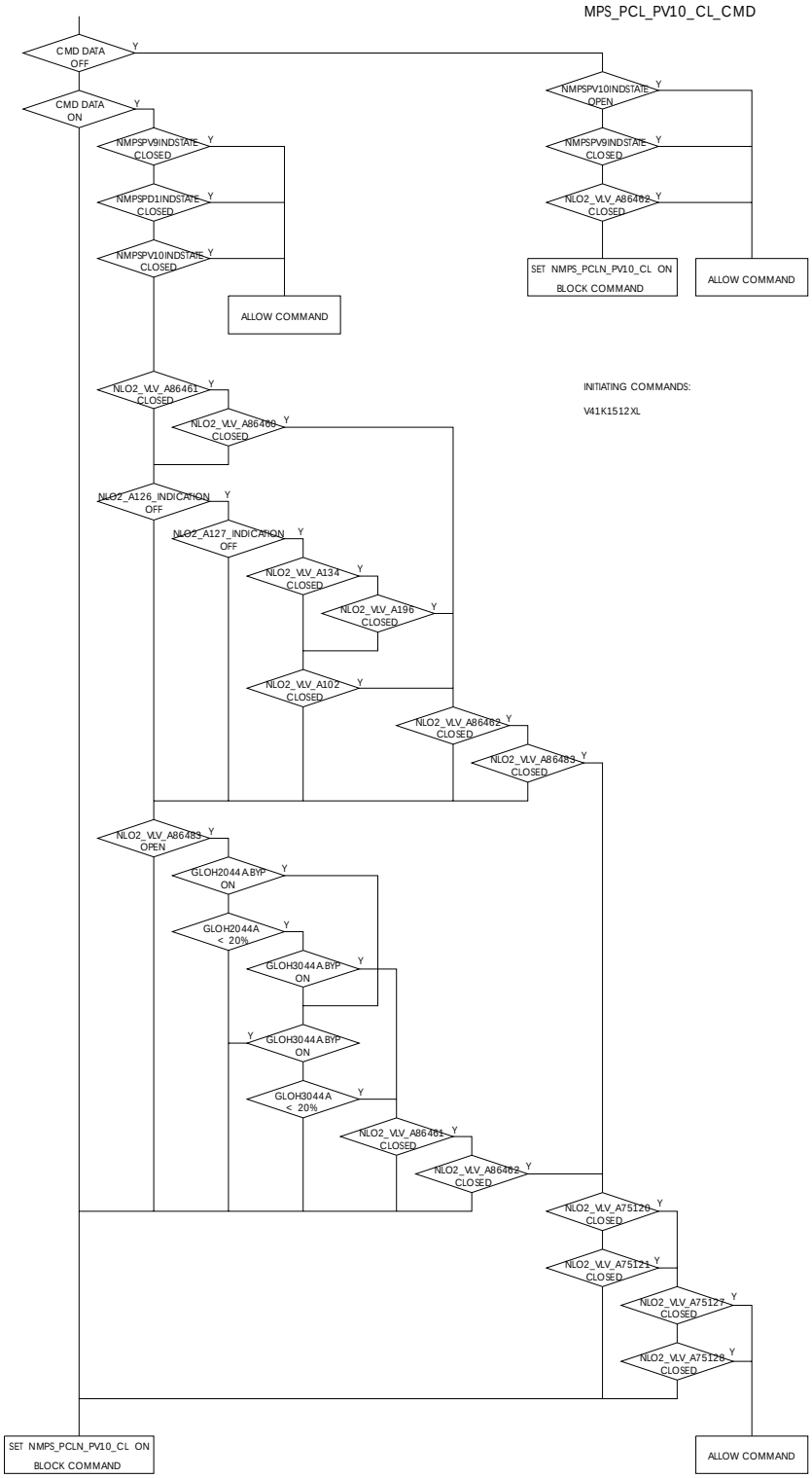
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.4.3.3. Algorithm

3.4.3.3.1. PCL Algorithm

3.4.3.3.1.1. Flowchart

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



3.4.3.3.1.2. Sequence

#include "mps_pcl_pv10_cl_cmd.h"

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
#include "SME.h"
#include "LO2.h"
#include "MPS.h"
#include "LH2.h"

static const char* seqName = "MPS_PCL_PV10_CL_CMD";
static const char* algorithmDescription = "Revision Date - 01/20/2000\n\
if (A) ||\n\
if Command_Data is ON &&\n\
  if (K || L || M || ((((B && C || (D && E && ((F && G) || H))) && I && J) || (N && O && B && I )) && (P || Q)
&& (R || S))) ||\n\
if Command_Data is OFF &&\n\
  if (T || K || I)\n\
where:   A  = NMPS_PCLO_PV10_CL - PCL Bypass ON\n\
         B  = A86461 - LO2 XFER LINE FILL VALVE - CLOSED\n\
         C  = A86460 - LO2 REPLENISH VALVE - CLOSED\n\
         D  = A126 - LO2 PUMP - OFF\n\
         E  = A127 - LO2 PUMP - OFF\n\
         F  = A134 - LO2 XFER LINE DRAIN VALVE - CLOSED\n\
         G = A196 - LO2 BYPASS VALVE - CLOSED\n\
         H = A102 - LO2 DRAIN LINE SOV - CLOSED\n\
         I   = A86462 - LO2 TSM DRAIN VALVE - CLOSED\n\
         J  = A86463 LO2 TSM VENT VALVE - CLOSED\n\
         K = PV9 - LO2 OUTBOARD F/D VALVE - CLOSED\n\
         L  = PD1 - LO2 ORB/ET FEED DISC VALVE - CLOSED\n\
         M = PV10 - LO2 INBOARD F/D VALVE - CLOSED\n\
         N = A86483 - LO2 TSM VENT - OPEN\n\
         O = A86460 - LO2 REPLENISH VALVE - < 20% OPEN\n\
         P = A75120 - LO2 PRG SOV - CLOSED\n\
         Q = A75121 - LO2 PRG SOV - CLOSED\n\
         R = A75127 - LO2 LK/CK SOV - CLOSED\n\
         S = A75128 - LO2 LK/CK SOV - CLOSED\n\
         T = PV10 - LO2 OUTBOARD F/D VALVE - OPEN";

MPS_PCL_PV10_CL_CMD::MPS_PCL_PV10_CL_CMD(const char*
seqName):asv_pcsVPCL_DS_ON_OFF_(seqName)
{
  userInit();
}

MPS_PCL_PV10_CL_CMD::~MPS_PCL_PV10_CL_CMD()
{
}


void MPS_PCL_PV10_CL_CMD::userInit(void)
{
  initiatingFD(V41K1512XL);
}

int MPS_PCL_PV10_CL_CMD::userExecute(Discrete::ON_OFF dsVar)
{

 if (Discrete::On == (NMPS_PCLO_PV10_CL.getValue()))
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
 {
 return PCL_ALLOW_CMD;
}

 if (Discrete::On==dsVar)
{
  if (K() || L() || M() || ((((B()&&C() || (D()&&E()&&((F()&&G()) || H())))
      &&I()&&J()) || (N()&&O()&&B()&&I()))&&(P() || Q())&&(R() || S())))
  {
   return PCL_ALLOW_CMD;
  }
  else
  {
  NMPS_PCLN_PV10_CL.Set(Discrete::On);
   return PCL_BLOCK_CMD;
  }
}
if (Discrete::Off==dsVar)
{
if (T() || K() || I())
  {
   return PCL_ALLOW_CMD;
  }
else
  {
   NMPS_PCLN_PV10_CL.Set(Discrete::On);
   return PCL_BLOCK_CMD;
  }
}

}


// Function that determines the state of the indicator bypasses.

int MPS_PCL_PV10_CL_CMD::indicatorBypass(PD_ &pseudo_ind)
{
  if ( Discrete::On == pseudo_ind.getValue())
   return IND_BYPASS_ON;
 else
   return IND_BYPASS_OFF;
}



// A86461 - LO2 Xfer line fill valve - CLOSED

int MPS_PCL_PV10_CL_CMD::B()
{
int  valReturnB = PCL_VOTER_FAIL;

try
{
            if (NLO2_VLV_A86461.getValue() == LH2_VALVE_STATE::CLOSED)
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
    {
            valReturnB = PCL_VOTER_PASS;
           }
else
           {
      valReturnB = PCL_VOTER_FAIL;
           }
}

catch(...)
{
    valReturnB = PCL_VOTER_FAIL;
}

return valReturnB;
}
```

```
// A86460 - LO2 Replenish valve - CLOSED

int MPS_PCL_PV10_CL_CMD::C()
{
int  valReturnC = PCL_VOTER_FAIL;

try
{
           if (NLO2_VLV_A86460.getValue() == LH2_VALVE_STATE::CLOSED)
    {
            valReturnC = PCL_VOTER_PASS;
           }
else
           {
      valReturnC = PCL_VOTER_FAIL;
           }
}

catch(...)
{
    valReturnC = PCL_VOTER_FAIL;
}

 return valReturnC;
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
// A126 - LO2 PUMP - OFF

int MPS_PCL_PV10_CL_CMD::D()
{
  int  valReturnD = PCL_VOTER_FAIL;

 try
{
            if (NLO2_PUMP_A126.getValue() == Discrete::Off)
     {
              valReturnD = PCL_VOTER_PASS;
             }
else
            {
       valReturnD = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnD = PCL_VOTER_FAIL;
}

return valReturnD;
}



// A127 - LO2 PUMP - OFF

int MPS_PCL_PV10_CL_CMD::E()
{
  int  valReturnE = PCL_VOTER_FAIL;

 try
{
            if (NLO2_PUMP_A127.getValue() == Discrete::Off)
     {
              valReturnE = PCL_VOTER_PASS;
             }
else
            {
       valReturnE = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnE = PCL_VOTER_FAIL;
}

  return valReturnE;
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
// A134 - LO2 Xfer line drain valve - CLOSED

int MPS_PCL_PV10_CL_CMD::F()
{
  int  valReturnF = PCL_VOTER_FAIL;

try
{
            if (NLO2_VLV_A134.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnF = PCL_VOTER_PASS;
             }
else
            {
       valReturnF = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnF = PCL_VOTER_FAIL;
}

  return valReturnF;
}


// A196 - LO2 bypass valve - CLOSED

int MPS_PCL_PV10_CL_CMD::G()
{
  int  valReturnG = PCL_VOTER_FAIL;

try
{
            if (NLO2_VLV_A196.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnG = PCL_VOTER_PASS;
             }
else
            {
       valReturnG = PCL_VOTER_FAIL;
            }
}

catch(...)
{
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
    valReturnG = PCL_VOTER_FAIL;
}

 return valReturnG;
}
```

```
// A102 - LO2 Drain line SOV - CLOSED

int MPS_PCL_PV10_CL_CMD::H()
{
 int  valReturnH = PCL_VOTER_FAIL;

 try
{
            if (NLO2_VLV_A102.getValue() == LH2_VALVE_STATE::CLOSED)
    {
              valReturnH = PCL_VOTER_PASS;
             }
else
            {
       valReturnH = PCL_VOTER_FAIL;
            }
}

catch(...)
{
    valReturnH = PCL_VOTER_FAIL;
}

 return valReturnH;
}
```

```
// A86462 - LO2 TSM drain valve - CLOSED

int MPS_PCL_PV10_CL_CMD::I()
{
 int  valReturnI = PCL_VOTER_FAIL;

 try
{
            if (NLO2_VLV_A86462.getValue() == LH2_VALVE_STATE::CLOSED)
    {
              valReturnI = PCL_VOTER_PASS;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
            }
else
            {
       valReturnI = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnI = PCL_VOTER_FAIL;
}



  return valReturnI;
}



// A86483 - LO2 TSM vent valve - CLOSED

int MPS_PCL_PV10_CL_CMD::J()
{
  int  valReturnJ = PCL_VOTER_FAIL;

try
{
            if (NLO2_VLV_A86483.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnJ = PCL_VOTER_PASS;
             }
else
            {
       valReturnJ = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnJ = PCL_VOTER_FAIL;
}

  return valReturnJ;
}



// PV9 - LO2 Outboard F/D Valve - CLOSED

int MPS_PCL_PV10_CL_CMD::K()
{
  int  valReturnK = PCL_VOTER_FAIL;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
try
{
            if (NMPSPV9INDSTATE.getValue() == MPS_VALVE_IND_STATE::CLOSED)
     {
              valReturnK = PCL_VOTER_PASS;
             }
else
            {
       valReturnK = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnK = PCL_VOTER_FAIL;
}

  return valReturnK;
}
```

```
//PD1 - LO2 ORB/ET Feed Disc Valve - CLOSED

int MPS_PCL_PV10_CL_CMD::L()
{
  int  valReturnL = PCL_VOTER_FAIL;

try
{
            if (NMPSPD1INDSTATE.getValue() == MPS_VALVE_IND_STATE::CLOSED)
     {
              valReturnL = PCL_VOTER_PASS;
             }
else
            {
       valReturnL = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnL = PCL_VOTER_FAIL;
}

  return valReturnL;
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
// PV10 - LO2 Inboard F/D Valve - CLOSED

int MPS_PCL_PV10_CL_CMD::M()
{
  int  valReturnM = PCL_VOTER_FAIL;

try
{
            if (NMPSPV10INDSTATE.getValue() == MPS_VALVE_IND_STATE::CLOSED)
     {
              valReturnM = PCL_VOTER_PASS;
             }
else
            {
       valReturnM = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnM = PCL_VOTER_FAIL;
}

  return valReturnM;
}



// A86483 - LO2 TSM vent valve - OPEN

int MPS_PCL_PV10_CL_CMD::N()
{
  int  valReturnN = PCL_VOTER_FAIL;

try
{
            if (NLO2_VLV_A86483.getValue() == MPS_VALVE_IND_STATE::OPEN)
     {
              valReturnN = PCL_VOTER_PASS;
             }
else
            {
       valReturnN = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnN = PCL_VOTER_FAIL;
}

  return valReturnN;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
}



// A86460 - LO2 Replenish valve < 20% OPEN

int MPS_PCL_PV10_CL_CMD::O()
{
  int  valReturnO = PCL_VOTER_FAIL;
  int  stalecount  = 0;
  int  bypasscount  = 0;

  if (indicatorBypass(NLOH2044A) == IND_BYPASS_OFF)
  {
    try
    {

            if (GLOH2044A.getValue() < 20)
           {
              valReturnO = PCL_VOTER_PASS;
             }
          else
             {
             valReturnO = PCL_VOTER_FAIL;
             }

    }

    catch(...)
    {
     stalecount = 1;
     valReturnO = PCL_VOTER_FAIL;
    }
  }

  else
  {
    bypasscount = 1;
    valReturnO = PCL_VOTER_FAIL;
  }


  if  (indicatorBypass(NLOH3044A) == IND_BYPASS_OFF)
  {
    try
    {

            if (GLOH3044A.getValue() < 20)
           {
              valReturnO &= PCL_VOTER_PASS;
             }
          else
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
            {
             valReturnO &= PCL_VOTER_FAIL;
            }

   }

   catch(...)
   {
    if (1 == stalecount)
       {
        valReturnO = PCL_VOTER_FAIL;
       }
   }
 }

 else
 {
  if (1 == bypasscount)
       {
        valReturnO = PCL_VOTER_FAIL;
       }
 }

 return valReturnO;
}
```

```
// A75120 - PRG SOV - CLOSED

int MPS_PCL_PV10_CL_CMD::P()
{
  int  valReturnP = PCL_VOTER_FAIL;

try
{
            if (NLO2_VLV_A75120.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnP = PCL_VOTER_PASS;
             }
else
            {
       valReturnP = PCL_VOTER_FAIL;
            }
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
catch(...)
{
    valReturnP = PCL_VOTER_FAIL;
}

return valReturnP;
}


// A75121 - PRG SOV - CLOSED

int MPS_PCL_PV10_CL_CMD::Q()
{
 int  valReturnQ = PCL_VOTER_FAIL;

try
{
            if (NLO2_VLV_A75121.getValue() == LH2_VALVE_STATE::CLOSED)
    {
              valReturnQ = PCL_VOTER_PASS;
             }
else
            {
       valReturnQ = PCL_VOTER_FAIL;
            }
}

catch(...)
{
    valReturnQ = PCL_VOTER_FAIL;
}



 return valReturnQ;
}




// A75127 - LK/CK VLV - CLOSED

int MPS_PCL_PV10_CL_CMD::R()
{
 int  valReturnR = PCL_VOTER_FAIL;

 try
{
            if (NLO2_VLV_A75127.getValue() == LH2_VALVE_STATE::CLOSED)
    {
              valReturnR = PCL_VOTER_PASS;
             }
else
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
            {
       valReturnR = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnR = PCL_VOTER_FAIL;
}

  return valReturnR;
}



// A75128 - LK/CK VLV - CLOSED

int MPS_PCL_PV10_CL_CMD::S()
{
  int  valReturnS = PCL_VOTER_FAIL;

try
{
            if (NLO2_VLV_A75128.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnS = PCL_VOTER_PASS;
             }
else
            {
       valReturnS = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnS = PCL_VOTER_FAIL;
}

  return valReturnS;
}




// PV10 - LO2 Inboard F/D Valve - OPEN

int MPS_PCL_PV10_CL_CMD::T()
{
  int  valReturnT = PCL_VOTER_FAIL;

try
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
{
          if (NMPSPV10INDSTATE.getValue() == MPS_VALVE_IND_STATE::OPEN)
    {
            valReturnT = PCL_VOTER_PASS;
           }
else
          {
      valReturnT = PCL_VOTER_FAIL;
          }
}

catch(...)
{
    valReturnT = PCL_VOTER_FAIL;
}

 return valReturnT;
}
```

MPS_PCL_PV10_CL_CMD mps_pcl_pv10_cl_cmd(seqName);

3.4.4. MPS_PCL_PV10_OP_CMD

3.4.4.1. Description

This PCL provides protection for the MPS LO2 Inboard Fill and Drain Valve Open Commands.
This class inherits from the class CAS_Control_Logic. The Algorithm will be implemented directly in C++ Code.
Reference - LPS Program GCL19/24

3.4.4.1.1. Instance

Instance Name
MPS_PCL_PV10_OP_CMD

3.4.4.1.2. PCL Initiating Commands

Command

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

V41K1501XL
V41K1502XL
V41K1503NL

3.4.4.1.3. Operations Index

Operation	Description
determineState()	This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE INHIBITED ERROR
activate()	This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.
inhibit()	This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.
getState()	This method is inherited from the base class - CAS_Control_Logic
resetCLNotification()	This method is inherited from the base class - CAS_Control_Logic

3.4.4.2. Operations

3.4.4.2.1. getState()

3.4.4.2.1.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.4.2.2. resetCLNotification()

3.4.4.2.2.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.4.2.3. determineState()

3.4.4.2.3.1. Descriptive Text

This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE, INHIBITED or ERROR.

3.4.4.2.3.2. Sequence

if (<V41K1501XL>.PCLSTAT = active AND
<V41K1502XL>.PCLSTAT = active AND

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
    <V41K1503NL>.PCLSTAT = active)
  {
  CLState = ACTIVE;
  }
else
  {
   if  (<V41K1501XL>.PCLSTAT = inhibited AND
        <V41K1502XL>.PCLSTAT = inhibited AND
        <V41K1503NL>.PCLSTAT = inhibited)
     {
     CLState = INHIBITED;
      }
    else
      {
     CLState = ERROR;
      }
   }
```

3.4.4.2.4. activate()

3.4.4.2.4.1. Descriptive Text

This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.4.2.4.2. Sequence

```
A FD <V41K1501XL> PCL;
A FD <V41K1502XL> PCL;
A FD <V41K1503NL> PCL;

return;
```

3.4.4.2.5. inhibit()

3.4.4.2.5.1. Descriptive Text

This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.4.2.5.2. Sequence

```
I FD <V41K1501XL> PCL;
I FD <V41K1502XL> PCL;
I FD <V41K1503NL> PCL;

return;
```

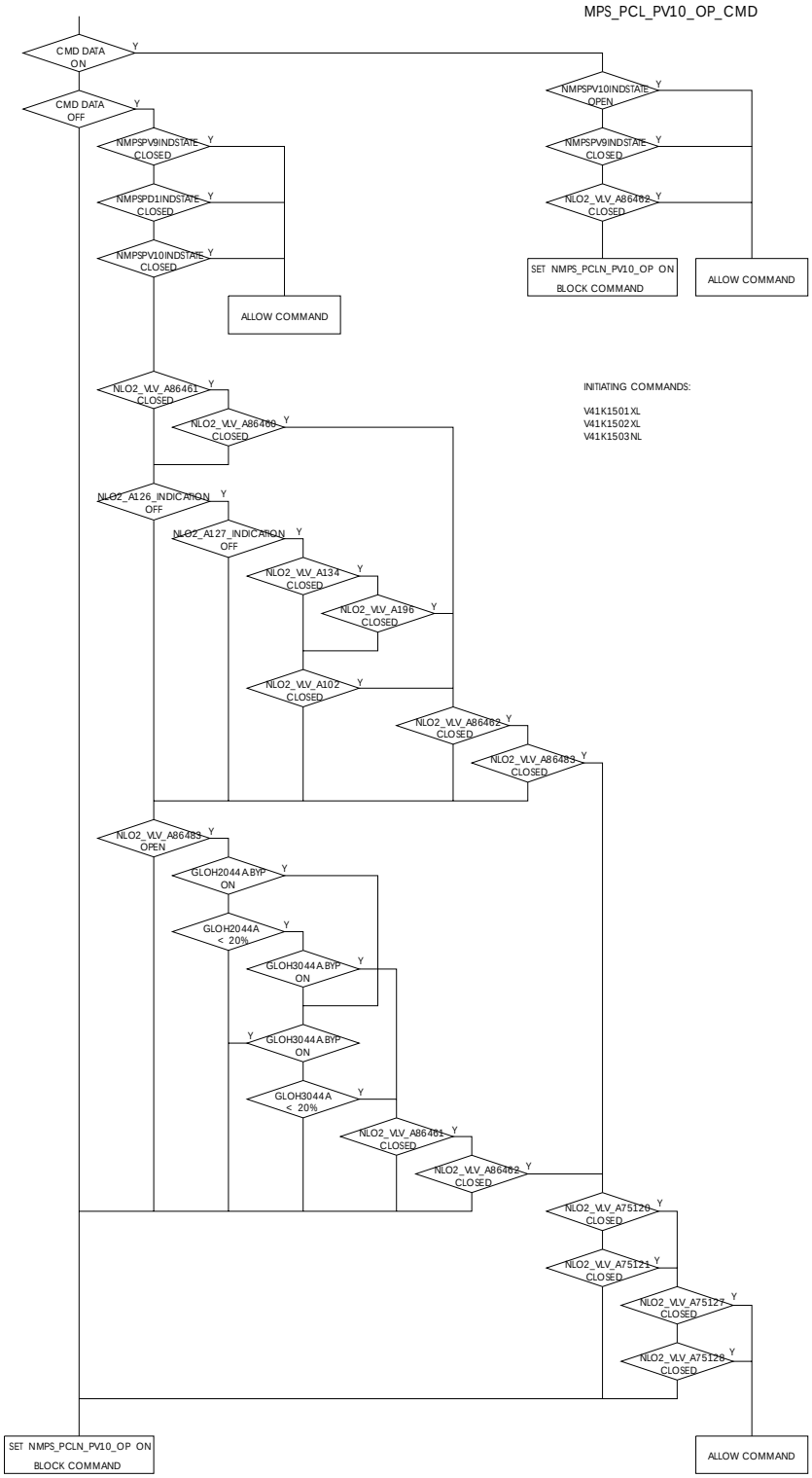

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.4.4.3. Algorithm

3.4.4.3.1. PCL Algorithm

3.4.4.3.1.1. Flowchart

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



3.4.4.3.1.2. Sequence

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
#include "mps_pcl_pv10_op_cmd.h"
#include "SME.h"
#include "LO2.h"
#include "MPS.h"
#include "LH2.h"

static const char* seqName = "MPS_PCL_PV10_OP_CMD";
static const char* algorithmDescription = "Revision Date - 01/20/2000\n\
if (A) ||\n\
if Command_Data is OFF &&\n\
  if (K || L || M || ((((B && C || (D && E && ((F && G) || H))) && I && J) || (N && O && B && I )) && (P || Q)
&& (R || S))) ||\n\
if Command_Data is ON &&\n\
  if (T || K || I)   \n\
where:   A  = NMPS_PCLO_PV10_OP - PCL Bypass ON\n\
         B  = A86461 - LO2 XFER LINE FILL VALVE - CLOSED\n\
         C  = A86460 - LO2 REPLENISH VALVE - CLOSED\n\
         D  = A126 - LO2 PUMP - OFF\n\
         E  = A127 - LO2 PUMP - OFF\n\
         F  = A134 - LO2 XFER LINE DRAIN VALVE - CLOSED\n\
         G = A196 - LO2 BYPASS VALVE - CLOSED\n\
         H = A102 - LO2 DRAIN LINE SOV - CLOSED\n\
         I   = A86462 - LO2 TSM DRAIN VALVE - CLOSED\n\
         J  = A86463 LO2 TSM VENT VALVE - CLOSED\n\
         K = PV9 - LO2 OUTBOARD F/D VALVE - CLOSED\n\
         L  = PD1 - LO2 ORB/ET FEED DISC VALVE - CLOSED\n\
         M = PV10 - LO2 INBOARD F/D VALVE - CLOSED\n\
         N = A86483 - LO2 TSM VENT - OPEN\n\
         O = A86460 - LO2 REPLENISH VALVE - < 20% OPEN\n\
         P = A75120 - LO2 PRG SOV - CLOSED\n\
         Q = A75121 - LO2 PRG SOV - CLOSED\n\
         R = A75127 - LO2 LK/CK SOV - CLOSED\n\
         S = A75128 - LO2 LK/CK SOV - CLOSED\n\
         T = PV10 - LO2 OUTBOARD F/D VALVE - OPEN";

MPS_PCL_PV10_OP_CMD::MPS_PCL_PV10_OP_CMD(const char*
seqName):asv_pcsVPCL_DS_ON_OFF_(seqName)
{
  userInit();
}

MPS_PCL_PV10_OP_CMD::~MPS_PCL_PV10_OP_CMD()
{
}


void MPS_PCL_PV10_OP_CMD::userInit(void)
{
  initiatingFD(V41K1501XL);
  initiatingFD(V41K1502XL);
  initiatingFD(V41K1503NL);
}

int MPS_PCL_PV10_OP_CMD::userExecute(Discrete::ON_OFF dsVar)
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
{

 if (Discrete::On == (NMPS_PCLO_PV10_OP.getValue()))
{
 return PCL_ALLOW_CMD;
}

 if (Discrete::Off==dsVar)
{
  if (K() || L() || M() || ((((B()&&C() || (D()&&E()&&((F()&&G()) || H())))
      &&I()&&J()) || (N()&&O()&&B()&&I()))&&(P() || Q())&&(R() || S())))
  {
   return PCL_ALLOW_CMD;
  }
  else
  {
   NMPS_PCLN_PV10_OP.Set(Discrete::On);
   return PCL_BLOCK_CMD;
  }
}
if (Discrete::On==dsVar)
{
if (T() || K() || I())
  {
   return PCL_ALLOW_CMD;
  }
else
  {
   NMPS_PCLN_PV10_OP.Set(Discrete::On);
   return PCL_BLOCK_CMD;
  }
}

}


// Function that determines the state of the indicator bypasses.

int MPS_PCL_PV10_OP_CMD::indicatorBypass(PD_ &pseudo_ind)
{
  if ( Discrete::On == pseudo_ind.getValue())
   return IND_BYPASS_ON;
 else
   return IND_BYPASS_OFF;
}



// A86461 - LO2 Xfer line fill valve - CLOSED

int MPS_PCL_PV10_OP_CMD::B()
{
int  valReturnB = PCL_VOTER_FAIL;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
try
{
            if (NLO2_VLV_A86461.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnB = PCL_VOTER_PASS;
             }
else
            {
       valReturnB = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnB = PCL_VOTER_FAIL;
}

return valReturnB;
}
```

```
// A86460 - LO2 Replenish valve - CLOSED

int MPS_PCL_PV10_OP_CMD::C()
{
int  valReturnC = PCL_VOTER_FAIL;

try
{
            if (NLO2_VLV_A86460.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnC = PCL_VOTER_PASS;
             }
else
            {
       valReturnC = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnC = PCL_VOTER_FAIL;
}

  return valReturnC;
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
// A126 - LO2 PUMP - OFF

int MPS_PCL_PV10_OP_CMD::D()
{
  int  valReturnD = PCL_VOTER_FAIL;

 try
{
            if (NLO2_PUMP_A126.getValue() == Discrete::Off)
     {
              valReturnD = PCL_VOTER_PASS;
             }
else
            {
       valReturnD = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnD = PCL_VOTER_FAIL;
}

return valReturnD;
}


// A127 - LO2 PUMP - OFF

int MPS_PCL_PV10_OP_CMD::E()
{
  int  valReturnE = PCL_VOTER_FAIL;

 try
{
            if (NLO2_PUMP_A127.getValue() == Discrete::Off)
     {
              valReturnE = PCL_VOTER_PASS;
             }
else
            {
       valReturnE = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnE = PCL_VOTER_FAIL;
}

  return valReturnE;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
}




// A134 - LO2 Xfer line drain valve - CLOSED

int MPS_PCL_PV10_OP_CMD::F()
{
  int  valReturnF = PCL_VOTER_FAIL;

try
{
            if (NLO2_VLV_A134.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnF = PCL_VOTER_PASS;
             }
else
            {
       valReturnF = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnF = PCL_VOTER_FAIL;
}

  return valReturnF;
}


// A196 - LO2 bypass valve - CLOSED

int MPS_PCL_PV10_OP_CMD::G()
{
  int  valReturnG = PCL_VOTER_FAIL;

try
{
            if (NLO2_VLV_A196.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnG = PCL_VOTER_PASS;
             }
else
            {
       valReturnG = PCL_VOTER_FAIL;
            }
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
catch(...)
{
    valReturnG = PCL_VOTER_FAIL;
}

 return valReturnG;
}
```

```
// A102 - LO2 Drain line SOV - CLOSED

int MPS_PCL_PV10_OP_CMD::H()
{
 int  valReturnH = PCL_VOTER_FAIL;

 try
{
            if (NLO2_VLV_A102.getValue() == LH2_VALVE_STATE::CLOSED)
    {
              valReturnH = PCL_VOTER_PASS;
             }
else
            {
       valReturnH = PCL_VOTER_FAIL;
            }
}

catch(...)
{
    valReturnH = PCL_VOTER_FAIL;
}

 return valReturnH;
}
```

```
// A86462 - LO2 TSM drain valve - CLOSED

int MPS_PCL_PV10_OP_CMD::I()
{
 int  valReturnI = PCL_VOTER_FAIL;

 try
{
            if (NLO2_VLV_A86462.getValue() == LH2_VALVE_STATE::CLOSED)
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
    {
            valReturnI = PCL_VOTER_PASS;
           }
else
           {
      valReturnI = PCL_VOTER_FAIL;
           }
}

catch(...)
{
    valReturnI = PCL_VOTER_FAIL;
}

 return valReturnI;
}



// A86483 - LO2 TSM vent valve - CLOSED

int MPS_PCL_PV10_OP_CMD::J()
{
 int  valReturnJ = PCL_VOTER_FAIL;

try
{
           if (NLO2_VLV_A86483.getValue() == LH2_VALVE_STATE::CLOSED)
    {
            valReturnJ = PCL_VOTER_PASS;
           }
else
           {
      valReturnJ = PCL_VOTER_FAIL;
           }
}

catch(...)
{
    valReturnJ = PCL_VOTER_FAIL;
}

 return valReturnJ;
}



// PV9 - LO2 Outboard F/D Valve - CLOSED

int MPS_PCL_PV10_OP_CMD::K()
{
 int  valReturnK = PCL_VOTER_FAIL;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
try
{
            if (NMPSPV9INDSTATE.getValue() == MPS_VALVE_IND_STATE::CLOSED)
     {
              valReturnK = PCL_VOTER_PASS;
             }
else
            {
       valReturnK = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnK = PCL_VOTER_FAIL;
}

  return valReturnK;
}
```

```
//PD1 - LO2 ORB/ET Feed Disc Valve - CLOSED

int MPS_PCL_PV10_OP_CMD::L()
{
  int  valReturnL = PCL_VOTER_FAIL;

try
{
            if (NMPSPD1INDSTATE.getValue() == MPS_VALVE_IND_STATE::CLOSED)
     {
              valReturnL = PCL_VOTER_PASS;
             }
else
            {
       valReturnL = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnL = PCL_VOTER_FAIL;
}

  return valReturnL;
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
// PV10 - LO2 Inboard F/D Valve - CLOSED

int MPS_PCL_PV10_OP_CMD::M()
{
  int  valReturnM = PCL_VOTER_FAIL;

try
{
            if (NMPSPV10INDSTATE.getValue() == MPS_VALVE_IND_STATE::CLOSED)
     {
              valReturnM = PCL_VOTER_PASS;
             }
else
            {
       valReturnM = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnM = PCL_VOTER_FAIL;
}

  return valReturnM;
}



// A86483 - LO2 TSM vent valve - OPEN

int MPS_PCL_PV10_OP_CMD::N()
{
  int  valReturnN = PCL_VOTER_FAIL;

try
{
            if (NLO2_VLV_A86483.getValue() == LH2_VALVE_STATE::OPEN)
     {
              valReturnN = PCL_VOTER_PASS;
             }
else
            {
       valReturnN = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnN = PCL_VOTER_FAIL;
}

  return valReturnN;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
}



// A86460 - LO2 Replenish valve < 20% OPEN

int MPS_PCL_PV10_OP_CMD::O()
{
 int  valReturnO = PCL_VOTER_FAIL;
 int  stalecount  = 0;
 int  bypasscount  = 0;

 if (indicatorBypass(NLOH2044A) == IND_BYPASS_OFF)
 {
  try
  {

          if (GLOH2044A.getValue() < 20)
         {
           valReturnO = PCL_VOTER_PASS;
          }
        else
          {
          valReturnO = PCL_VOTER_FAIL;
          }

  }

  catch(...)
  {
   stalecount = 1;
   valReturnO = PCL_VOTER_FAIL;
  }
 }

 else
 {
  bypasscount = 1;
  valReturnO = PCL_VOTER_FAIL;
 }


 if  (indicatorBypass(NLOH3044A) == IND_BYPASS_OFF)
 {
  try
  {

          if (GLOH3044A.getValue() < 20)
         {
           valReturnO &= PCL_VOTER_PASS;
          }
        else
          {
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
            valReturnO &= PCL_VOTER_FAIL;
           }

   }

   catch(...)
   {
    if (1 == stalecount)
       {
        valReturnO = PCL_VOTER_FAIL;
       }
   }
 }

 else
 {
  if (1 == bypasscount)
       {
        valReturnO = PCL_VOTER_FAIL;
       }
 }

 return valReturnO;
}




// A75120 - PRG SOV - CLOSED

int MPS_PCL_PV10_OP_CMD::P()
{
  int  valReturnP = PCL_VOTER_FAIL;

try
{
           if (NLO2_VLV_A75120.getValue() == LH2_VALVE_STATE::CLOSED)
    {
             valReturnP = PCL_VOTER_PASS;
            }
else
           {
      valReturnP = PCL_VOTER_FAIL;
           }
}

catch(...)
{
    valReturnP = PCL_VOTER_FAIL;
}

return valReturnP;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
}


// A75121 - PRG SOV - CLOSED

int MPS_PCL_PV10_OP_CMD::Q()
{
  int  valReturnQ = PCL_VOTER_FAIL;

try
{
            if (NLO2_VLV_A75121.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnQ = PCL_VOTER_PASS;
             }
else
            {
       valReturnQ = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnQ = PCL_VOTER_FAIL;
}



  return valReturnQ;
}




// A75127 - LK/CK VLV - CLOSED

int MPS_PCL_PV10_OP_CMD::R()
{
  int  valReturnR = PCL_VOTER_FAIL;

 try
{
            if (NLO2_VLV_A75127.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnR = PCL_VOTER_PASS;
             }
else
            {
       valReturnR = PCL_VOTER_FAIL;
            }
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
catch(...)
{
    valReturnR = PCL_VOTER_FAIL;
}

 return valReturnR;
}



// A75128 - LK/CK VLV - CLOSED

int MPS_PCL_PV10_OP_CMD::S()
{
 int  valReturnS = PCL_VOTER_FAIL;

try
{
          if (NLO2_VLV_A75128.getValue() == LH2_VALVE_STATE::CLOSED)
    {
            valReturnS = PCL_VOTER_PASS;
           }
else
          {
      valReturnS = PCL_VOTER_FAIL;
          }
}

catch(...)
{
    valReturnS = PCL_VOTER_FAIL;
}

 return valReturnS;
}




// PV10 - LO2 Inboard F/D Valve - OPEN

int MPS_PCL_PV10_OP_CMD::T()
{
 int  valReturnT = PCL_VOTER_FAIL;

try
{
          if (NMPSPV10INDSTATE.getValue() == MPS_VALVE_IND_STATE::OPEN)
    {
            valReturnT = PCL_VOTER_PASS;
           }
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
else
            {
       valReturnT = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnT = PCL_VOTER_FAIL;
}

  return valReturnT;
}



MPS_PCL_PV10_OP_CMD mps_pcl_pv10_op_cmd(seqName);
```

3.4.5. MPS_PCL_PD1_CL_CMD

3.4.5.1. Description

This PCL provides protection for the MPS LO2 ORB/ET Feed Disconnect Valve Close Commands.
This class inherits from the class CAS_Control_Logic. The Algorithm will be implemented directly in C++ Code.
Reference - LPS Program GCL20

3.4.5.1.1. Instance

Instance Name
MPS_PCL_PD1_CL_CMD

3.4.5.1.2. PCL Initiating Commands

Command
V41K1524XL
V41K1525XL
V41K1526XL

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.4.5.1.3. Operations Index

Operation	Description
determineState()	This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE INHIBITED ERROR
activate()	This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.
inhibit()	This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.
getState()	This method is inherited from the base class - CAS_Control_Logic
resetCLNotification()	This method is inherited from the base class - CAS_Control_Logic

3.4.5.2. Operations

3.4.5.2.1. getState()

3.4.5.2.1.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.5.2.2. resetCLNotification()

3.4.5.2.2.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.5.2.3. determineState()

3.4.5.2.3.1. Descriptive Text

This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE, INHIBITED or ERROR.

3.4.5.2.3.2. Sequence

```
if  (<V41K1524XL>.PCLSTAT = active  AND
    <V41K1525XL>.PCLSTAT = active  AND
    <V41K1526XL>.PCLSTAT = active)
  {
  CLState = ACTIVE;
  }
else
  {
   if  (<V41K1524XL>.PCLSTAT = inhibited AND
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
    <V41K1524XL>.PCLSTAT = inhibited AND
    <V41K1526XL>.PCLSTAT = inhibited)
  {
  CLState = INHIBITED;
   }
 else
   {
  CLState = ERROR;
   }
}
```

3.4.5.2.4. activate()

3.4.5.2.4.1. Descriptive Text

This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.5.2.4.2. Sequence

A FD <V41K1524XL> PCL;
A FD <V41K1525XL> PCL;
A FD <V41K1526XL> PCL;

return;

3.4.5.2.5. inhibit()

3.4.5.2.5.1. Descriptive Text

This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.5.2.5.2. Sequence

I FD <V41K1524XL> PCL;
I FD <V41K1525XL> PCL;
I FD <V41K1526XL> PCL;

return;

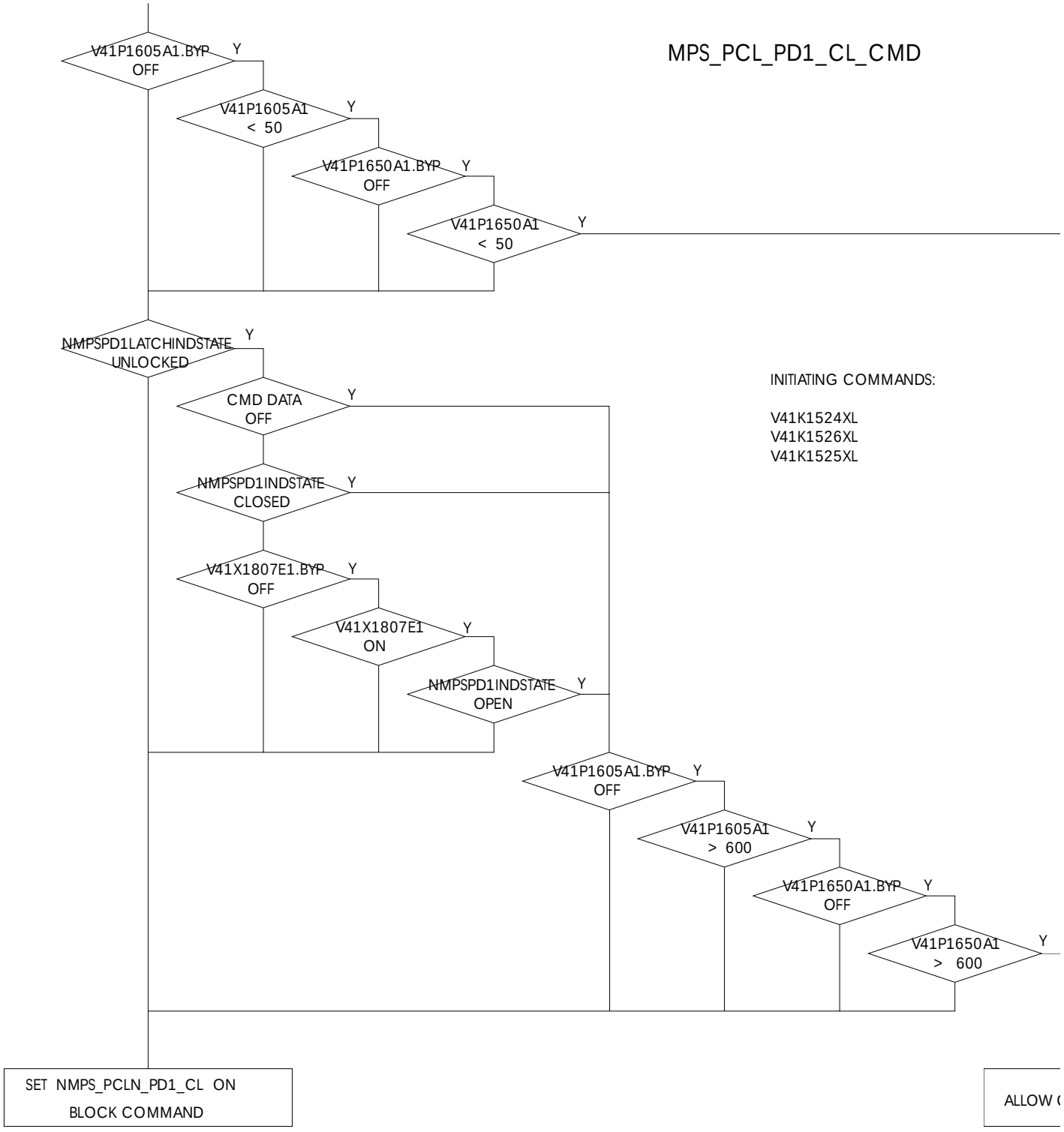
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.4.5.3. Algorithm

3.4.5.3.1. PCL Algorithm

3.4.5.3.1.1. Flowchart

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



3.4.5.3.1.2. Sequence

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
#include "mps_pcl_pd1_cl_cmd.h"
#include "SME.h"
#include "LO2.h"
#include "MPS.h"
#include "LH2.h"

static const char* seqName = "MPS_PCL_PD1_CL_CMD";
static const char* algorithmDescription = "Revision Date - 01/20/2000\n\
  if ((A && B) || (!B && C && D) || (!B && E && (H || (F && I) || G) && J && K))\n\
where:   A  = NMPS_PCLO_PD1_CL - PCL Emergency Override -  ON\n\
         B  = NLOK0408X - LO2 LOADING PREPS COMPLETE - ON\n\
         C  = MPS Pneu Reg Press < 50 psia\n\
         D  = MPS Pneu Accum Press < 50 psia\n\
         E  = PD1 - LO2 ORB/ET FEED DISC VALVE LATCH - UNLOCKED\n\
         F  = PD1 - LO2 ORB/ET FEED DISC VALVE OPEN POWER - ON\n\
         G  = PD1 - LO2 ORB/ET FEED DISC VALVE - CLOSED\n\
         H  = COMMAND DATA - OFF\n\
         I  = PD1 - LO2 ORB/ET FEED DISC VALVE - OPEN\n\
         J  = MPS Pneu Reg Press > 600 psia\n\
         K  = MPS Pneu Accum Press > 600 psia";

MPS_PCL_PD1_CL_CMD::MPS_PCL_PD1_CL_CMD(const char*
seqName):asv_pcsVPCL_DS_ON_OFF_(seqName)
{
  userInit();
}

MPS_PCL_PD1_CL_CMD::~MPS_PCL_PD1_CL_CMD()
{
}


void MPS_PCL_PD1_CL_CMD::userInit(void)
{
  initiatingFD(V41K1524XL);
  initiatingFD(V41K1525XL);
  initiatingFD(V41K1526XL);
}

int MPS_PCL_PD1_CL_CMD::userExecute(Discrete::ON_OFF dsVar)
{

if  ((A() && B()) || (!B() && C() && D()) || (!B() && E() && (H() || (F() && I()) || G()) && J() && K()))
   {
    return PCL_ALLOW_CMD;
   }
else
   {
    NMPS_PCLN_PD1_CL.Set(Discrete::On);
    return PCL_BLOCK_CMD;
   }

}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
// Function that determines the state of the indicator bypasses.


int MPS_PCL_PD1_CL_CMD::indicatorBypass(PD_ &pseudo_ind)
{
  if ( Discrete::On == pseudo_ind.getValue())
   return IND_BYPASS_ON;
 else
   return IND_BYPASS_OFF;
}



// Emergency Override ON

int MPS_PCL_PD1_CL_CMD::A()
{
  int  valReturnA = PCL_VOTER_FAIL;

    try
    {
      valReturnA =  (Discrete::On == (NMPS_PCLO_PD1_CL.getValue()));
    }

    catch(...)
    {
      valReturnA = PCL_VOTER_FAIL;
    }

  return valReturnA;
}



// LO2 Loading Preps Complete

int MPS_PCL_PD1_CL_CMD::B()
{
  int  valReturnB = PCL_VOTER_FAIL;

    try
    {
      valReturnB =  (Discrete::On == (NLOK0408X.getValue()));
    }

    catch(...)
    {
      valReturnB = PCL_VOTER_FAIL;
    }

  return valReturnB;
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
// MPS Pneu Reg Press < 50

int MPS_PCL_PD1_CL_CMD::C()
{
  int  valReturnC = PCL_VOTER_FAIL;

  if (indicatorBypass(N41P1605A1))
  {
    try
    {
      valReturnC =  (V41P1605A1.getValue() < 50);
    }

    catch(...)
    {
      valReturnC = PCL_VOTER_FAIL;
    }
  }

  else
  {
    valReturnC = PCL_VOTER_FAIL;
  }

  return valReturnC;
}




// MPS Pneu Accum Press < 50

int MPS_PCL_PD1_CL_CMD::D()
{
  int  valReturnD = PCL_VOTER_FAIL;

  if (indicatorBypass(N41P1650A1))
  {
    try
    {
      valReturnD =  (V41P1650A1.getValue() < 50);
    }

    catch(...)
    {
      valReturnD = PCL_VOTER_FAIL;
    }
  }

  else
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
  {
    valReturnD = PCL_VOTER_FAIL;
  }

  return valReturnD;
}



// PD1 - LO2 ORB/ET Feed Disconnect Latch - Unlocked

int MPS_PCL_PD1_CL_CMD::E()
{
  int  valReturnE = PCL_VOTER_FAIL;

try
{
            if (NMPSPD1LATCHINDSTATE.getValue() == MPS_ORB_LATCH_IND::UNLOCKED)
           {
              valReturnE = PCL_VOTER_PASS;
            }
else
            {
       valReturnE = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnE = PCL_VOTER_FAIL;
}

  return valReturnE;
}







// PD1 OPEN POWER - ON

int MPS_PCL_PD1_CL_CMD::F()
{
  int  valReturnF = PCL_VOTER_FAIL;

  if (indicatorBypass(N41X1807E1))
  {
    try
    {
      valReturnF =   (Discrete::On == (V41X1807E1.getValue()));
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
    }

    catch(...)
    {
      valReturnF = PCL_VOTER_FAIL;
    }
  }

  else
  {
    valReturnF = PCL_VOTER_FAIL;
  }

  return valReturnF;
}



//PD1 - LO2 ORB/ET Feed Disc Valve - CLOSED

int MPS_PCL_PD1_CL_CMD::G()
{
  int  valReturnG = PCL_VOTER_FAIL;

try
{
            if (NMPSPD1INDSTATE.getValue() == MPS_VALVE_IND_STATE::CLOSED)
     {
              valReturnG = PCL_VOTER_PASS;
             }
else
            {
       valReturnG = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnG = PCL_VOTER_FAIL;
}

  return valReturnG;
}




// COMMAND DATA - OFF

int MPS_PCL_PD1_CL_CMD::H()
{
  int  valReturnH = PCL_VOTER_FAIL;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
   try
   {
    valReturnH =  (Discrete::Off == dsVar);
   }

   catch(...)
   {
    valReturnH = PCL_VOTER_FAIL;
   }

  return valReturnH;
}



//PD1 - LO2 ORB/ET Feed Disc Valve - OPEN

int MPS_PCL_PD1_CL_CMD::I()
{
  int  valReturnI = PCL_VOTER_FAIL;

try
{
            if (NMPSPD1INDSTATE.getValue() == MPS_VALVE_IND_STATE::OPEN)
     {
              valReturnI = PCL_VOTER_PASS;
             }
else
            {
       valReturnI = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnI = PCL_VOTER_FAIL;
}

  return valReturnI;
}




// MPS Pneu Reg Press > 600

int MPS_PCL_PD1_CL_CMD::J()
{
  int  valReturnJ = PCL_VOTER_FAIL;

  if (indicatorBypass(N41P1605A1))
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
  {
   try
   {
    valReturnJ =  (V41P1605A1.getValue() > 600);
   }

   catch(...)
   {
    valReturnJ = PCL_VOTER_FAIL;
   }
  }

  else
  {
   valReturnJ = PCL_VOTER_FAIL;
  }

  return valReturnJ;
}




// MPS Pneu Accum Press > 600

int MPS_PCL_PD1_CL_CMD::K()
{
  int  valReturnK = PCL_VOTER_FAIL;

  if (indicatorBypass(N41P1650A1))
  {
   try
   {
    valReturnK =  (V41P1650A1.getValue() > 600);
   }

   catch(...)
   {
    valReturnK = PCL_VOTER_FAIL;
   }
  }

  else
  {
   valReturnK = PCL_VOTER_FAIL;
  }

  return valReturnK;
}


MPS_PCL_PD1_CL_CMD mps_pcl_pd1_cl_cmd(seqName);
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.4.6. MPS_PCL_PD1_OP_CMD

3.4.6.1. Description

This PCL provides protection for the MPS LO2 ORB/ET Feed Disconnect Valve Open Commands.
This class inherits from the class CAS_Control_Logic. The Algorithm will be implemented directly in C++ Code.
Reference - LPS Program GCL25

3.4.6.1.1. Instance

Instance Name
MPS_PCL_PD1_OP_CMD

3.4.6.1.2. PCL Initiating Commands

Command
V41K1521XL
V41K1522XL
V41K1523XL

3.4.6.1.3. Operations Index

Operation	**Description**
determineState()	This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE INHIBITED ERROR
activate()	This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.
inhibit()	This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.
getState()	This method is inherited from the base class - CAS_Control_Logic

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

resetCLNotification()	This method is inherited from the base class - CAS_Control_Logic

3.4.6.2. Operations

3.4.6.2.1. getState()

3.4.6.2.1.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.6.2.2. resetCLNotification()

3.4.6.2.2.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.6.2.3. determineState()

3.4.6.2.3.1. Descriptive Text

This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE, INHIBITED or ERROR.

3.4.6.2.3.2. Sequence

```
if  (<V41K1521XL>.PCLSTAT = active  AND
     <V41K1522XL>.PCLSTAT = active  AND
     <V41K1523XL>.PCLSTAT = active)
  {
  CLState = ACTIVE;
  }
else
  {
   if  (<V41K1521XL>.PCLSTAT = inhibited AND
        <V41K1522XL>.PCLSTAT = inhibited AND
        <V41K1523XL>.PCLSTAT = inhibited)
     {
     CLState = INHIBITED;
      }
    else
      {
     CLState = ERROR;
      }
   }
```

3.4.6.2.4. activate()

3.4.6.2.4.1. Descriptive Text

This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.6.2.4.2. Sequence

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

A FD <V41K1521XL> PCL;
A FD <V41K1522XL> PCL;
A FD <V41K1523XL> PCL;

return;

3.4.6.2.5. inhibit()

3.4.6.2.5.1. Descriptive Text

This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.6.2.5.2. Sequence

I FD <V41K1521XL> PCL;
I FD <V41K1522XL> PCL;
I FD <V41K1523XL> PCL;

return;

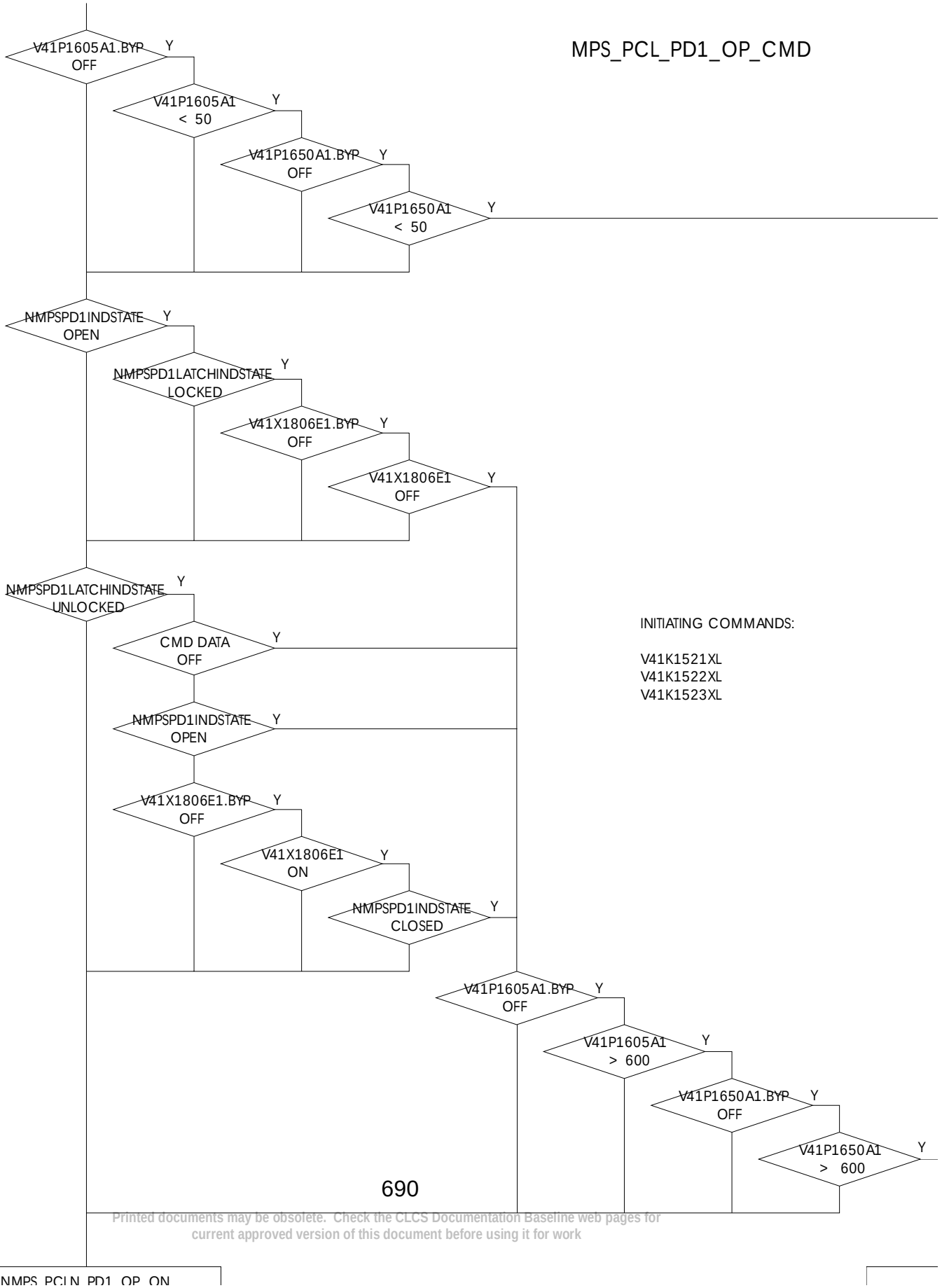
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.4.6.3. Algorithm

3.4.6.3.1. PCL Algorithm

3.4.6.3.1.1. Flowchart

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.4.6.3.1.2. Sequence

```
#include "mps_pcl_pd1_op_cmd.h"
#include "SME.h"
#include "LO2.h"
#include "MPS.h"
#include "LH2.h"

static const char* seqName = "MPS_PCL_PD1_OP_CMD";
static const char* algorithmDescription = "Revision Date - 01/20/2000\n\
  if ((A && B) || (!B && C && D) || (!B && E && F && G && J && K) || (!B && H && (L || (I && M) || E) &&
J && K))\n\
where:   A  = NMPS_PCLO_PD1_OP - PCL Emergency Override -  ON\n\
         B  = NLOK0408X - LO2 LOADING PREPS COMPLETE - ON\n\
         C  = MPS Pneu Reg Press < 50 psia\n\
         D  = MPS Pneu Accum Press < 50 psia\n\
         E  = PD1 - LO2 ORB/ET FEED DISC VALVE - OPEN\n\
         F  = PD1 - LO2 ORB/ET FEED DISC VALVE LATCH - LOCKED\n\
         G  = PD1 - LO2 ORB/ET FEED DISC VALVE CLOSE POWER - OFF\n\
         H  = PD1 - LO2 ORB/ET FEED DISC VALVE LATCH - UNLOCKED\n\
         I   = PD1 - LO2 ORB/ET FEED DISC VALVE CLOSE POWER - ON\n\
         J  = MPS Pneu Reg Press > 600 psia\n\
         K  = MPS Pneu Accum Press > 600 psia\n\
         L  = COMMAND DATA - OFF\n\
         M  = PD1 - LO2 ORB/ET FEED DISC VALVE - CLOSED";

MPS_PCL_PD1_OP_CMD::MPS_PCL_PD1_OP_CMD(const char*
seqName):asv_pcsVPCL_DS_ON_OFF_(seqName)
{
  userInit();
}

MPS_PCL_PD1_OP_CMD::~MPS_PCL_PD1_OP_CMD()
{
}


void MPS_PCL_PD1_OP_CMD::userInit(void)
{
  initiatingFD(V41K1521XL);
  initiatingFD(V41K1522XL);
  initiatingFD(V41K1523XL);
}

int MPS_PCL_PD1_OP_CMD::userExecute(Discrete::ON_OFF dsVar)
{

if  ((A() && B()) || (!B() && C() && D()) || (!B() && E() && F() && G() && J() && K()) || (!B() && H() &&
(L() || (I() && M()) || E()) && J() && K()))
  {
   return PCL_ALLOW_CMD;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
  }
else
  {
   NMPS_PCLN_PD1_OP.Set(Discrete::On);
   return PCL_BLOCK_CMD;
  }

}


// Function that determines the state of the indicator bypasses.

int MPS_PCL_PD1_OP_CMD::indicatorBypass(PD_ &pseudo_ind)
{
  if ( Discrete::On == pseudo_ind.getValue())
   return IND_BYPASS_ON;
 else
   return IND_BYPASS_OFF;
}



// Emergency Override ON

int MPS_PCL_PD1_OP_CMD::A()
{
  int  valReturnA = PCL_VOTER_FAIL;

    try
    {
      valReturnA =  (Discrete::On == (NMPS_PCLO_PD1_OP.getValue()));
    }

    catch(...)
    {
      valReturnA = PCL_VOTER_FAIL;
    }

  return valReturnA;
}



// LO2 Loading Preps Complete

int MPS_PCL_PD1_OP_CMD::B()
{
  int  valReturnB = PCL_VOTER_FAIL;

    try
    {
      valReturnB =    (Discrete::On == (NLOK0408X.getValue()));
    }
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
   catch(...)
   {
    valReturnB = PCL_VOTER_FAIL;
   }

 return valReturnB;
}



// MPS Pneu Reg Press < 50

int MPS_PCL_PD1_OP_CMD::C()
{
 int  valReturnC = PCL_VOTER_FAIL;

 if (indicatorBypass(N41P1605A1))
 {
  try
   {
    valReturnC =  (V41P1605A1.getValue() < 50);
   }

   catch(...)
   {
    valReturnC = PCL_VOTER_FAIL;
   }
 }

 else
 {
  valReturnC = PCL_VOTER_FAIL;
 }

 return valReturnC;
}




// MPS Pneu Accum Press < 50

int MPS_PCL_PD1_OP_CMD::D()
{
 int  valReturnD = PCL_VOTER_FAIL;

 if (indicatorBypass(N41P1650A1))
 {
  try
   {
    valReturnD =  (V41P1650A1.getValue() < 50);
   }
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
   catch(...)
   {
    valReturnD = PCL_VOTER_FAIL;
   }
  }

  else
  {
   valReturnD = PCL_VOTER_FAIL;
  }

  return valReturnD;
}




//PD1 - LO2 ORB/ET Feed Disc Valve - OPEN

int MPS_PCL_PD1_OP_CMD::E()
{
  int  valReturnE = PCL_VOTER_FAIL;

try
{
            if (NMPSPD1INDSTATE.getValue() == MPS_VALVE_IND_STATE::OPEN)
     {
              valReturnE = PCL_VOTER_PASS;
             }
else
            {
       valReturnE = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnE = PCL_VOTER_FAIL;
}

  return valReturnE;
}




// PD1 - LO2 ORB/ET Feed Disconnect Latch - Locked

int MPS_PCL_PD1_OP_CMD::F()
{
  int  valReturnF = PCL_VOTER_FAIL;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
try
{
            if (NMPSPD1LATCHINDSTATE.getValue() == MPS_ORB_LATCH_IND::LOCKED)
           {
              valReturnF = PCL_VOTER_PASS;
             }
else
            {
       valReturnF = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnF = PCL_VOTER_FAIL;
}

  return valReturnF;
}



// PD1 CLOSE POWER - OFF

int MPS_PCL_PD1_OP_CMD::G()
{
  int  valReturnG = PCL_VOTER_FAIL;

  if (indicatorBypass(N41X1806E1))
  {
    try
    {
      valReturnG =  (Discrete::Off == (V41X1806E1.getValue()));
    }

    catch(...)
    {
      valReturnG = PCL_VOTER_FAIL;
    }
  }

  else
  {
    valReturnG = PCL_VOTER_FAIL;
  }

  return valReturnG;
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
// PD1 - LO2 ORB/ET Feed Disconnect Latch - Unlocked

int MPS_PCL_PD1_OP_CMD::H()
{
  int  valReturnH = PCL_VOTER_FAIL;

try
{
            if (NMPSPD1LATCHINDSTATE.getValue() == MPS_ORB_LATCH_IND::UNLOCKED)
           {
              valReturnH = PCL_VOTER_PASS;
            }
else
            {
       valReturnH = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnH = PCL_VOTER_FAIL;
}

  return valReturnH;
}




// PD1 CLOSE POWER - ON

int MPS_PCL_PD1_OP_CMD::I()
{
  int  valReturnI = PCL_VOTER_FAIL;

  if (indicatorBypass(N41X1806E1))
  {
    try
    {
      valReturnI =    (Discrete::On == (V41X1806E1.getValue()));
    }

    catch(...)
    {
      valReturnI = PCL_VOTER_FAIL;
    }
  }

  else
  {
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
    valReturnI = PCL_VOTER_FAIL;
  }

  return valReturnI;
}




// MPS Pneu Reg Press > 600

int MPS_PCL_PD1_OP_CMD::J()
{
  int  valReturnJ = PCL_VOTER_FAIL;

  if (indicatorBypass(N41P1605A1))
  {
    try
    {
      valReturnJ =  (V41P1605A1.getValue() > 600);
    }

    catch(...)
    {
      valReturnJ = PCL_VOTER_FAIL;
    }
  }

  else
  {
    valReturnJ = PCL_VOTER_FAIL;
  }

  return valReturnJ;
}




// MPS Pneu Accum Press > 600

int MPS_PCL_PD1_OP_CMD::K()
{
  int  valReturnK = PCL_VOTER_FAIL;

  if (indicatorBypass(N41P1650A1))
  {
    try
    {
      valReturnK =  (V41P1650A1.getValue() > 600);
    }
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
   catch(...)
   {
    valReturnK = PCL_VOTER_FAIL;
   }
  }

  else
  {
   valReturnK = PCL_VOTER_FAIL;
  }

  return valReturnK;
}




// COMMAND DATA - OFF

int MPS_PCL_PD1_OP_CMD::L()
{
  int  valReturnL = PCL_VOTER_FAIL;

   try
   {
    valReturnL =  (Discrete::Off == dsVar);
   }

   catch(...)
   {
    valReturnL = PCL_VOTER_FAIL;
   }

  return valReturnL;
}




//PD1 - LO2 ORB/ET Feed Disc Valve - CLOSED

int MPS_PCL_PD1_OP_CMD::M()
{
  int  valReturnM = PCL_VOTER_FAIL;

try
{
          if (NMPSPD1INDSTATE.getValue() == MPS_VALVE_IND_STATE::CLOSED)
    {
           valReturnM = PCL_VOTER_PASS;
          }
else
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
            {
       valReturnM = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnM = PCL_VOTER_FAIL;
}

  return valReturnM;
}



MPS_PCL_PD1_OP_CMD mps_pcl_pd1_op_cmd(seqName);
```

3.4.7. MPS_PCL_PV19_CL_CMD

3.4.7.1. Description

This PCL provides protection for the MPS LO2 Overboard Bleed Valve Close Commands.
This class inherits from the class CAS_Control_Logic. The Algorithm will be implemented directly in C++ Code.
Reference - LPS Program GCL33

3.4.7.1.1. Instance

Instance Name
MPS_PCL_PV19_CL_CMD

3.4.7.1.2. PCL Initiating Commands

Command
V41K1584XL
V41K1585XL
V41K1586XL

3.4.7.1.3. Operations Index

Operation	**Description**

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

determineState()	This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE INHIBITED ERROR
activate()	This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.
inhibit()	This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.
getState()	This method is inherited from the base class - CAS_Control_Logic
resetCLNotification()	This method is inherited from the base class - CAS_Control_Logic

3.4.7.2. Operations

3.4.7.2.1. getState()

3.4.7.2.1.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.7.2.2. resetCLNotification()

3.4.7.2.2.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.7.2.3. determineState()

3.4.7.2.3.1. Descriptive Text

This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE, INHIBITED or ERROR.

3.4.7.2.3.2. Sequence

```
if  (<V41K1584XL>.PCLSTAT = active  AND
    <V41K1585XL>.PCLSTAT = active  AND
    <V41K1586XL>.PCLSTAT = active)
  {
  CLState = ACTIVE;
  }
else
  {
   if  (<V41K1584XL>.PCLSTAT = inhibited AND
       <V41K1585XL>.PCLSTAT = inhibited AND
       <V41K1586XL>.PCLSTAT = inhibited)
     {
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
    CLState = INHIBITED;
     }
   else
     {
    CLState = ERROR;
     }
  }
```

3.4.7.2.4. activate()

3.4.7.2.4.1. Descriptive Text

This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.7.2.4.2. Sequence

A FD <V41K1584XL> PCL;
A FD <V41K1585XL> PCL;
A FD <V41K1586XL> PCL;

return;

3.4.7.2.5. inhibit()

3.4.7.2.5.1. Descriptive Text

This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.7.2.5.2. Sequence

I FD <V41K1584XL> PCL;
I FD <V41K1585XL> PCL;
I FD <V41K1586XL> PCL;

return;

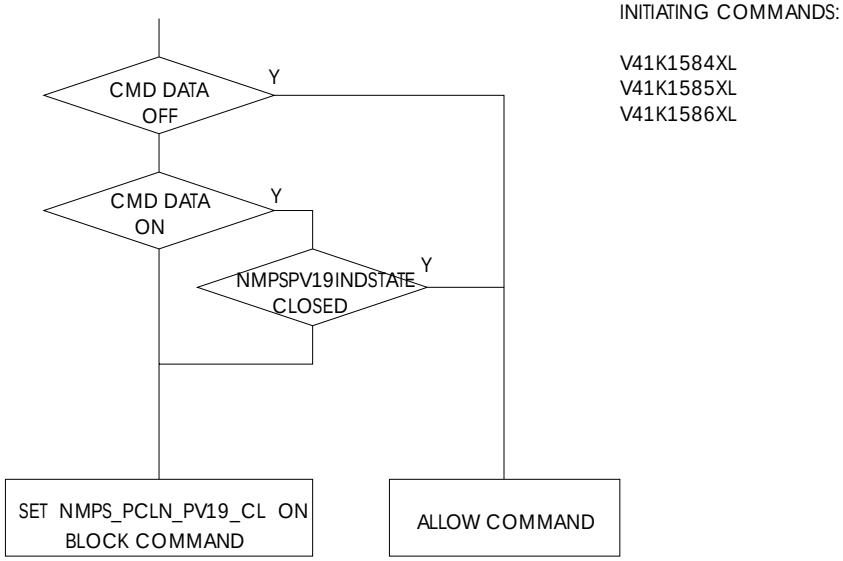
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.4.7.3. Algorithm

3.4.7.3.1. PCL Algorithm

3.4.7.3.1.1. Flowchart

MPS_PCL_PV19_CL_CMD



3.4.7.3.1.2. Sequence

```
#include "mps_pcl_pv19_cl_cmd.h"
#include "SME.h"
#include "LO2.h"
#include "MPS.h"
#include "LH2.h"

static const char* seqName = "MPS_PCL_PV19_CL_CMD";
static const char* algorithmDescription = "Revision Date - 01/20/2000\n\
if (A  || B || (C && D))\n\
where:   A  = NMPS_PCLO_PV19_CL - PCL Bypass ON\n\
         B  = COMMAND DATA - OFF\n\
         C  = COMMAND DATA - ON\n\
         D = PV9 - LO2 OVEROARD BLEED VALVE - CLOSED";

MPS_PCL_PV19_CL_CMD::MPS_PCL_PV19_CL_CMD(const char*
seqName):asv_pcsVPCL_DS_ON_OFF_(seqName)
{
  userInit();
}

MPS_PCL_PV19_CL_CMD::~MPS_PCL_PV19_CL_CMD()
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
{
}


void MPS_PCL_PV19_CL_CMD::userInit(void)
{
  initiatingFD(V41K1584XL);
  initiatingFD(V41K1585XL);
  initiatingFD(V41K1586XL);
}

int MPS_PCL_PV19_CL_CMD::userExecute(Discrete::ON_OFF dsVar)
{
  if (A() || B())
  {
   return PCL_ALLOW_CMD;
  }
  else
  {
   NMPS_PCLN_PV19_CL.Set(Discrete::On);
   return PCL_BLOCK_CMD;
  }
}




// PCL Override ON

int MPS_PCL_PV19_CL_CMD::A()
{
  int  valReturnA = PCL_VOTER_FAIL;

    try
    {
      valReturnA =  (Discrete::On == (NMPS_PCLO_PV19_CL.getValue()));
    }

    catch(...)
    {
      valReturnA = PCL_VOTER_FAIL;
    }

  return valReturnA;
}



// COMMAND DATA - OFF

int MPS_PCL_PV19_CL_CMD::B()
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
{
  int  valReturnB = PCL_VOTER_FAIL;

    try
    {
      valReturnB =  (Discrete::Off == dsVar);
    }

    catch(...)
    {
      valReturnB = PCL_VOTER_FAIL;
    }

  return valReturnB;
}



// COMMAND DATA - On

int MPS_PCL_PV19_CL_CMD::C()
{
  int  valReturnC = PCL_VOTER_FAIL;

    try
    {
      valReturnC =  (Discrete::On == dsVar);
    }

    catch(...)
    {
      valReturnC = PCL_VOTER_FAIL;
    }

  return valReturnC;
}




// PV19 - LO2 Overboard Bleed Valve - CLOSED

int MPS_PCL_PV19_CL_CMD::D()
{
  int  valReturnD = PCL_VOTER_FAIL;

try
{
            if (NMPSPV19INDSTATE.getValue() == MPS_VALVE_IND_STATE::CLOSED)
            {
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
            valReturnD = PCL_VOTER_PASS;
           }
else
           {
      valReturnD = PCL_VOTER_FAIL;
           }
}

catch(...)
{
    valReturnD = PCL_VOTER_FAIL;
}

 return valReturnD;
}
```

MPS_PCL_PV19_CL_CMD mps_pcl_pv19_cl_cmd(seqName);

3.4.8. MPS_PCL_PV7_CL_CMD

3.4.8.1. Description

This PCL provides protection for the MPS LO2 Feedline Relief Isolation Valve Close Commands.
This class inherits from the class CAS_Control_Logic. The Algorithm will be implemented directly in C++ Code.
Reference - LPS Program GCL38

3.4.8.1.1. Instance

Instance Name
MPS_PCL_PV7_CL_CMD

3.4.8.1.2. PCL Initiating Commands

Command
V41K1547XL
V41K1548XL
V41K1550XL

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.4.8.1.3. Operations Index

Operation	Description
determineState()	This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE INHIBITED ERROR
activate()	This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.
inhibit()	This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.
getState()	This method is inherited from the base class - CAS_Control_Logic
resetCLNotification()	This method is inherited from the base class - CAS_Control_Logic

3.4.8.2. Operations

3.4.8.2.1. getState()

3.4.8.2.1.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.8.2.2. resetCLNotification()

3.4.8.2.2.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.8.2.3. determineState()

3.4.8.2.3.1. Descriptive Text

This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE, INHIBITED or ERROR.

3.4.8.2.3.2. Sequence

```
if  (<V41K1547XL>.PCLSTAT = active  AND
    <V41K1548XL>.PCLSTAT = active  AND
    <V41K1550XL>.PCLSTAT = active)
  {
  CLState = ACTIVE;
  }
else
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
{
 if  (<V41K1547XL>.PCLSTAT = inhibited AND
      <V41K1548XL>.PCLSTAT = inhibited AND
      <V41K1550XL>.PCLSTAT = inhibited)
   {
   CLState = INHIBITED;
    }
  else
    {
   CLState = ERROR;
    }
 }
```

3.4.8.2.4. activate()

3.4.8.2.4.1. Descriptive Text

This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.8.2.4.2. Sequence

```
A FD <V41K1547XL> PCL;
A FD <V41K1548XL> PCL;
A FD <V41K1550XL> PCL;

return;
```

3.4.8.2.5. inhibit()

3.4.8.2.5.1. Descriptive Text

This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.8.2.5.2. Sequence

```
I FD <V41K1547XL> PCL;
I FD <V41K1548XL> PCL;
I FD <V41K1550XL> PCL;

return;
```

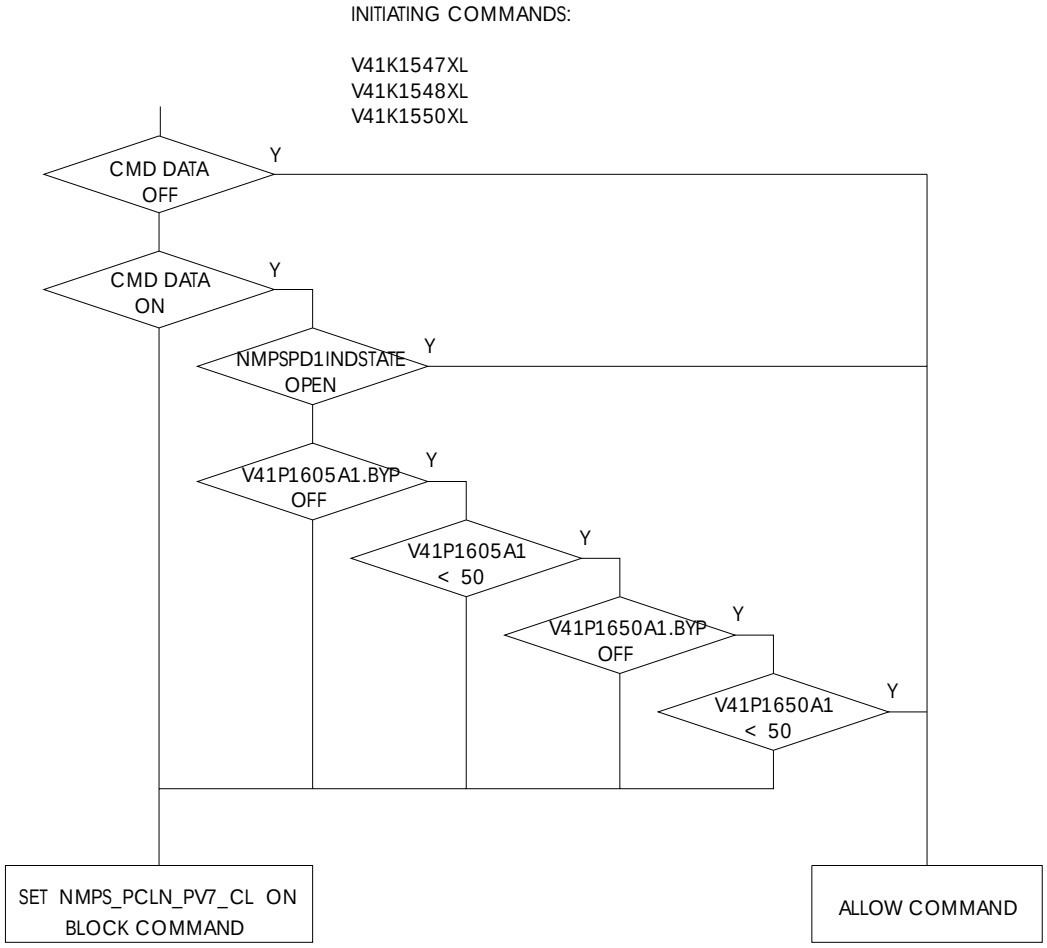

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.4.8.3. Algorithm

3.4.8.3.1. PCL Algorithm

3.4.8.3.1.1. Flowchart

MPS_PCL_PV7_CL_CMD



3.4.8.3.1.2. Sequence

```
#include "mps_pcl_pv7_cl_cmd.h"
#include "SME.h"
#include "LO2.h"
#include "MPS.h"
#include "LH2.h"

static const char* seqName = "MPS_PCL_PV7_CL_CMD";
static const char* algorithmDescription = "Revision Date - 01/20/2000\n\
if (A  || B || (C && (D || (E && F))))\n\
where:   A  = NMPS_PCLO_PV7_CL - PCL Bypass ON\n\
         B  = COMMAND DATA - OFF\n\
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
        C  = COMMAND DATA - ON\n\
        D  = PD1 - LO2 ORB/ET FEED DISC VALVE - OPEN\n\
        E  = MPS Pneu Reg Press < 50 psia\n\
        F  = MPS Pneu Accum Press < 50 psia";

MPS_PCL_PV7_CL_CMD::MPS_PCL_PV7_CL_CMD(const char*
seqName):asv_pcsVPCL_DS_ON_OFF_(seqName)
{
  userInit();
}

MPS_PCL_PV7_CL_CMD::~MPS_PCL_PV7_CL_CMD()
{
}


void MPS_PCL_PV7_CL_CMD::userInit(void)
{
  initiatingFD(V41K1547XL);
  initiatingFD(V41K1548XL);
  initiatingFD(V41K1550XL);
}

int MPS_PCL_PV7_CL_CMD::userExecute(Discrete::ON_OFF dsVar)
{
  if (A() || B() || (C() && D()))
  {
   return PCL_ALLOW_CMD;
  }
  else
  {
   NMPS_PCLN_PV7_CL.Set(Discrete::On);
   return PCL_BLOCK_CMD;
  }
}



// Function that determines the state of the indicator bypasses.

int MPS_PCL_PV7_OP_CMD::indicatorBypass(PD_ &pseudo_ind)
{
  if ( Discrete::On == pseudo_ind.getValue())
   return IND_BYPASS_ON;
 else
   return IND_BYPASS_OFF;
}




// PCL Override ON
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
int MPS_PCL_PV7_CL_CMD::A()
{
  int  valReturnA = PCL_VOTER_FAIL;

    try
    {
      valReturnA =  (Discrete::On == (NMPS_PCLO_PV7_CL.getValue()));
    }

    catch(...)
    {
      valReturnA = PCL_VOTER_FAIL;
    }

  return valReturnA;
}




// COMMAND DATA - OFF

int MPS_PCL_PV7_CL_CMD::B()
{
  int  valReturnB = PCL_VOTER_FAIL;

    try
    {
      valReturnB =  (Discrete::Off == dsVar);
    }

    catch(...)
    {
      valReturnB = PCL_VOTER_FAIL;
    }

  return valReturnB;
}




// COMMAND DATA - On

int MPS_PCL_PV7_CL_CMD::C()
{
  int  valReturnC = PCL_VOTER_FAIL;

    try
    {
      valReturnC =  (Discrete::On == dsVar);
    }
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
   catch(...)
   {
    valReturnC = PCL_VOTER_FAIL;
   }

  return valReturnC;
}




//PD1 - LO2 ORB/ET Feed Disc Valve - OPEN

int MPS_PCL_PV7_CL_CMD::D()
{
  int  valReturnD = PCL_VOTER_FAIL;

try
{
            if (NMPSPD1INDSTATE.getValue() == MPS_VALVE_IND_STATE::OPEN)
     {
              valReturnD = PCL_VOTER_PASS;
             }
else
            {
       valReturnD = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnD = PCL_VOTER_FAIL;
}

  return valReturnD;
}




// MPS Pneu Reg Press < 50

int MPS_PCL_PV7_CL_CMD::E()
{
  int  valReturnE = PCL_VOTER_FAIL;

  if (indicatorBypass(N41P1605A1))
  {
    try
    {
     valReturnE =  (V41P1605A1.getValue() < 50);
    }
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
   catch(...)
   {
    valReturnE = PCL_VOTER_FAIL;
   }
  }

  else
  {
   valReturnE = PCL_VOTER_FAIL;
  }

  return valReturnE;
}



// MPS Pneu Accum Press < 50

int MPS_PCL_PV7_CL_CMD::F()
{
  int  valReturnF = PCL_VOTER_FAIL;

  if (indicatorBypass(N41P1650A1))
  {
   try
   {
    valReturnF =  (V41P1650A1.getValue() < 50);
   }

   catch(...)
   {
    valReturnF = PCL_VOTER_FAIL;
   }
  }

  else
  {
   valReturnF = PCL_VOTER_FAIL;
  }

  return valReturnF;
}



MPS_PCL_PV7_CL_CMD mps_pcl_pv7_cl_cmd(seqName);
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.4.9. MPS_PCL_PD1_LATCH_LK_CMD

3.4.9.1. Description

This PCL provides protection for the MPS LO2 Feed Disconnect Valve Latch Lock Commands.
This class inherits from the class CAS_Control_Logic. The Algorithm will be implemented directly in C++ Code.
Reference - LPS Program GCL39

3.4.9.1.1. Instance

Instance Name
MPS_PCL_PD1_LATCH_LK_CMD

3.4.9.1.2. PCL Initiating Commands

Command
V41K1881XL
V41K1882XL
V41K1883XL

3.4.9.1.3. Operations Index

Operation	**Description**
determineState()	This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE INHIBITED ERROR
activate()	This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.
inhibit()	This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.
getState()	This method is inherited from the base class - CAS_Control_Logic
resetCLNotification()	This method is inherited from the base class - CAS_Control_Logic

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.4.9.2. Operations

3.4.9.2.1. getState()

3.4.9.2.1.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.9.2.2. resetCLNotification()

3.4.9.2.2.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.9.2.3. determineState()

3.4.9.2.3.1. Descriptive Text

This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE, INHIBITED or ERROR.

3.4.9.2.3.2. Sequence

```
if  (<V41K1881XL>.PCLSTAT = active  AND
     <V41K1882XL>.PCLSTAT = active  AND
     <V41K1883XL>.PCLSTAT = active)
  {
  CLState = ACTIVE;
  }
else
  {
   if  (<V41K1881XL>.PCLSTAT = inhibited AND
        <V41K1882XL>.PCLSTAT = inhibited AND
        <V41K1883XL>.PCLSTAT = inhibited)
     {
     CLState = INHIBITED;
      }
    else
      {
     CLState = ERROR;
      }
   }
```

3.4.9.2.4. activate()

3.4.9.2.4.1. Descriptive Text

This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.9.2.4.2. Sequence

```
A FD <V41K1881XL> PCL;
A FD <V41K1882XL> PCL;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

A FD <V41K1883XL> PCL;

return;

3.4.9.2.5. inhibit()

3.4.9.2.5.1. Descriptive Text

This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.9.2.5.2. Sequence

I FD <V41K1881XL> PCL;
I FD <V41K1882XL> PCL;
I FD <V41K1883XL> PCL;

return;

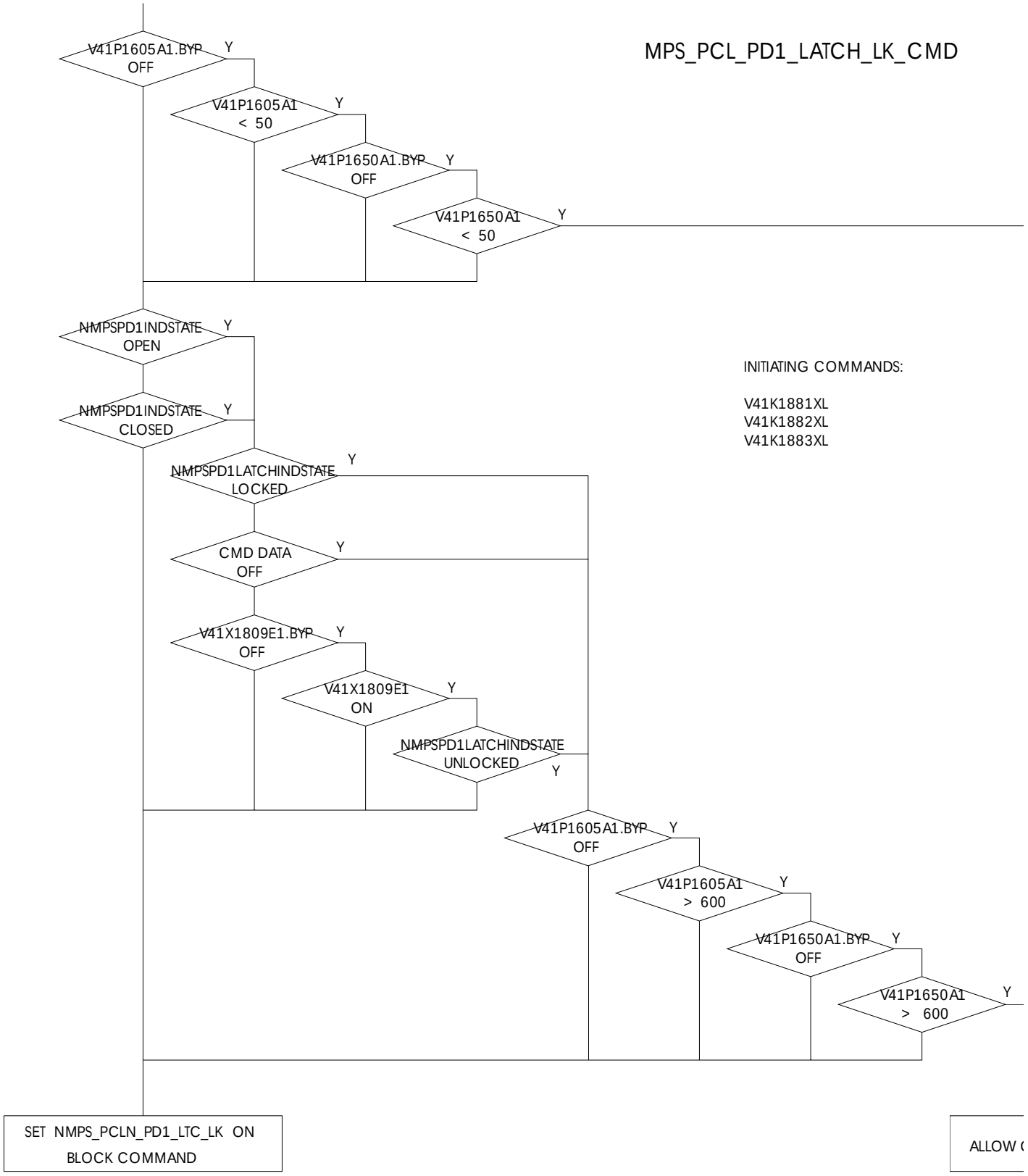
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.4.9.3. Algorithm

3.4.9.3.1. PCL Algorithm

3.4.9.3.1.1. Flowchart

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.4.9.3.1.2. Sequence

```
#include "mps_pcl_pd1_latch_lk_cmd.h"
#include "SME.h"
#include "LO2.h"
#include "MPS.h"
#include "LH2.h"

static const char* seqName = "MPS_PCL_PD1_LATCH_LK_CMD";
static const char* algorithmDescription = "Revision Date - 01/20/2000\n\
  if   (A  || (B && C) || ((D || E) && (G || (F && H) || I) && J && K))\n\
where:   A  = NMPS_PCLO_PD1_LTC_LK - PCL Bypass -  ON\n\
           B  = MPS Pneu Reg Press < 50 psia\n\
           C  = MPS Pneu Accum Press < 50 psia\n\
           D  = PD1 - LO2 ORB/ET FEED DISC VALVE - OPEN\n\
           E  = PD1 - LO2 ORB/ET FEED DISC VALVE - CLOSED\n\
           F  = PD1 - LO2 ORB/ET FEED DISC VALVE LATCH UNLOCK POWER - ON\n\
           G  = PD1 - LO2 ORB/ET FEED DISC VALVE LATCH - LOCKED\n\
           H  = PD1 - LO2 ORB/ET FEED DISC VALVE LATCH - UNLOCKED\n\
           I   = COMMAND DATA - OFF\n\
           J  = MPS Pneu Reg Press > 600 psia\n\
           K  = MPS Pneu Accum Press > 600 psia";

MPS_PCL_PD1_LATCH_LK_CMD::MPS_PCL_PD1_LATCH_LK_CMD
(const char* seqName):asv_pcsVPCL_DS_ON_OFF_(seqName)
{
  userInit();
}

MPS_PCL_PD1_LATCH_LK_CMD::~MPS_PCL_PD1_LATCH_LK_CMD()
{
}


void MPS_PCL_PD1_LATCH_LK_CMD::userInit(void)
{
  initiatingFD(V41K1881XL);
  initiatingFD(V41K1882XL);
  initiatingFD(V41K1883XL);
}

int MPS_PCL_PD1_LATCH_LK_CMD::userExecute(Discrete::ON_OFF dsVar)
{

if   (A() || (B() && C()) || ((D() || E()) && (G() || (F() && H()) || I()) && J() && K()))
   {
    return PCL_ALLOW_CMD;
   }
else
   {
    NMPS_PCLN_PD1_LTC_LK.Set(Discrete::On);
    return PCL_BLOCK_CMD;
   }
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
}


// Function that determines the state of the indicator bypasses.

int MPS_PCL_PD1_LATCH_LK_CMD::indicatorBypass(PD_ &pseudo_ind)
{
  if ( Discrete::On == pseudo_ind.getValue())
   return IND_BYPASS_ON;
 else
   return IND_BYPASS_OFF;
}



// PCL Bypass - ON

int MPS_PCL_PD1_LATCH_LK_CMD::A()
{
  int  valReturnA = PCL_VOTER_FAIL;

    try
    {
      valReturnA =  (Discrete::On == (NMPS_PCLO_PD1_LTC_LK.getValue()));
    }

    catch(...)
    {
      valReturnA = PCL_VOTER_FAIL;
    }

  return valReturnA;
}




// MPS Pneu Reg Press < 50

int MPS_PCL_PD1_LATCH_LK_CMD::B()
{
  int  valReturnB = PCL_VOTER_FAIL;

  if (indicatorBypass(N41P1605A1))
  {
    try
    {
      valReturnB =  (V41P1605A1.getValue() < 50);
    }

    catch(...)
    {
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
    valReturnB = PCL_VOTER_FAIL;
   }
  }

  else
  {
   valReturnB = PCL_VOTER_FAIL;
  }

  return valReturnB;
}




// MPS Pneu Accum Press < 50

int MPS_PCL_PD1_LATCH_LK_CMD::C()
{
  int  valReturnC = PCL_VOTER_FAIL;

  if (indicatorBypass(N41P1650A1))
  {
   try
   {
    valReturnC =  (V41P1650A1.getValue() < 50);
   }

   catch(...)
   {
    valReturnC = PCL_VOTER_FAIL;
   }
  }

  else
  {
   valReturnC = PCL_VOTER_FAIL;
  }

  return valReturnC;
}



//PD1 - LO2 ORB/ET Feed Disc Valve - OPEN

int MPS_PCL_PD1_LATCH_LK_CMD::D()
{
  int  valReturnD = PCL_VOTER_FAIL;

try
{
        if (NMPSPD1INDSTATE.getValue() == MPS_VALVE_IND_STATE::OPEN)
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
    {
            valReturnD = PCL_VOTER_PASS;
           }
else
           {
      valReturnD = PCL_VOTER_FAIL;
           }
}

catch(...)
{
    valReturnD = PCL_VOTER_FAIL;
}

 return valReturnD;
}



//PD1 - LO2 ORB/ET Feed Disc Valve - CLOSED

int MPS_PCL_PD1_LATCH_LK_CMD::E()
{
 int  valReturnE = PCL_VOTER_FAIL;

try
{
           if (NMPSPD1INDSTATE.getValue() == MPS_VALVE_IND_STATE::CLOSED)
    {
            valReturnE = PCL_VOTER_PASS;
           }
else
           {
      valReturnE = PCL_VOTER_FAIL;
           }
}

catch(...)
{
    valReturnE = PCL_VOTER_FAIL;
}

 return valReturnE;
}


// PD1 LATCH UNLOCK POWER - ON

int MPS_PCL_PD1_LATCH_LK_CMD::F()
{
 int  valReturnF = PCL_VOTER_FAIL;

 if (indicatorBypass(N41X1809E1))
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
 {
  try
  {
   valReturnF =  (Discrete::On == (V41X1809E1.getValue()));
  }

  catch(...)
  {
   valReturnF = PCL_VOTER_FAIL;
  }
 }

 else
 {
  valReturnF = PCL_VOTER_FAIL;
 }

 return valReturnF;
}



// PD1 - LO2 ORB/ET Feed Disconnect Latch - Locked

int MPS_PCL_PD1_LATCH_LK_CMD::G()
{
 int  valReturnG = PCL_VOTER_FAIL;

try
{
            if (NMPSPD1LATCHINDSTATE.getValue() == MPS_ORB_LATCH_IND::LOCKED)
           {
              valReturnG = PCL_VOTER_PASS;
            }
else
            {
      valReturnG = PCL_VOTER_FAIL;
            }
}

catch(...)
{
    valReturnG = PCL_VOTER_FAIL;
}

 return valReturnG;
}




// PD1 - LO2 ORB/ET Feed Disconnect Latch - Unlocked
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
int MPS_PCL_PD1_LATCH_LK_CMD::H()
{
  int  valReturnH = PCL_VOTER_FAIL;

try
{
            if (NMPSPD1LATCHINDSTATE.getValue() == MPS_ORB_LATCH_IND::UNLOCKED)
           {
              valReturnH = PCL_VOTER_PASS;
            }
else
            {
       valReturnH = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnH = PCL_VOTER_FAIL;
}

  return valReturnH;
}



// COMMAND DATA - OFF

int MPS_PCL_PD1_LATCH_LK_CMD::I()
{
  int  valReturnI = PCL_VOTER_FAIL;

    try
    {
      valReturnI =  (Discrete::Off == dsVar);
    }

    catch(...)
    {
      valReturnI = PCL_VOTER_FAIL;
    }

  return valReturnI;
}




// MPS Pneu Reg Press > 600

int MPS_PCL_PD1_LATCH_LK_CMD::J()
{
  int  valReturnJ = PCL_VOTER_FAIL;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
  if (indicatorBypass(N41P1605A1))
  {
    try
    {
      valReturnJ =  (V41P1605A1.getValue() > 600);
    }

    catch(...)
    {
      valReturnJ = PCL_VOTER_FAIL;
    }
  }

  else
  {
    valReturnJ = PCL_VOTER_FAIL;
  }

  return valReturnJ;
}




// MPS Pneu Accum Press > 600

int MPS_PCL_PD1_LATCH_LK_CMD::K()
{
  int  valReturnK = PCL_VOTER_FAIL;

  if (indicatorBypass(N41P1650A1))
  {
    try
    {
      valReturnK =  (V41P1650A1.getValue() > 600);
    }

    catch(...)
    {
      valReturnK = PCL_VOTER_FAIL;
    }
  }

  else
  {
    valReturnK = PCL_VOTER_FAIL;
  }

  return valReturnK;
}


MPS_PCL_PD1_LATCH_LK_CMD mps_pcl_pd1_latch_lk_cmd(seqName);
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.4.10. MPS_PCL_PD1_LATCH_UL_CMD

3.4.10.1. Description

This PCL provides protection for the MPS LO2 Feed Disconnect Valve Latch Unlock Commands.
This class inherits from the class CAS_Control_Logic. The Algorithm will be implemented directly in C++ Code.

Reference - LPS Program GCL40

3.4.10.1.1. Instance

Instance Name
MPS_PCL_PD1_LATCH_UL_CMD

3.4.10.1.2. PCL Initiating Commands

Command
V41K1884XL
V41K1885XL
V41K1886XL

3.4.10.1.3. Operations Index

Operation	**Description**
determineState()	This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE INHIBITED ERROR
activate()	This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.
inhibit()	This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

getState()	This method is inherited from the base class - CAS_Control_Logic
resetCLNotification()	This method is inherited from the base class - CAS_Control_Logic

3.4.10.2. Operations

3.4.10.2.1. getState()

3.4.10.2.1.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.10.2.2. resetCLNotification()

3.4.10.2.2.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.10.2.3. determineState()

3.4.10.2.3.1. Descriptive Text

This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE, INHIBITED or ERROR.

3.4.10.2.3.2. Sequence

```
if  (<V41K1884XL>.PCLSTAT = active  AND
    <V41K1885XL>.PCLSTAT = active  AND
    <V41K1886XL>.PCLSTAT = active)
  {
  CLState = ACTIVE;
  }
else
  {
   if  (<V41K1884XL>.PCLSTAT = inhibited AND
       <V41K1885XL>.PCLSTAT = inhibited AND
       <V41K1886XL>.PCLSTAT = inhibited)
     {
     CLState = INHIBITED;
      }
    else
      {
     CLState = ERROR;
      }
   }
```

3.4.10.2.4. activate()

3.4.10.2.4.1. Descriptive Text

This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.4.10.2.4.2. Sequence

A FD <V41K1884XL> PCL;
A FD <V41K1885XL> PCL;
A FD <V41K1886XL> PCL;

return;

3.4.10.2.5. inhibit()

3.4.10.2.5.1. Descriptive Text

This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.10.2.5.2. Sequence

I FD <V41K1884XL> PCL;
I FD <V41K1885XL> PCL;
I FD <V41K1886XL> PCL;

return;

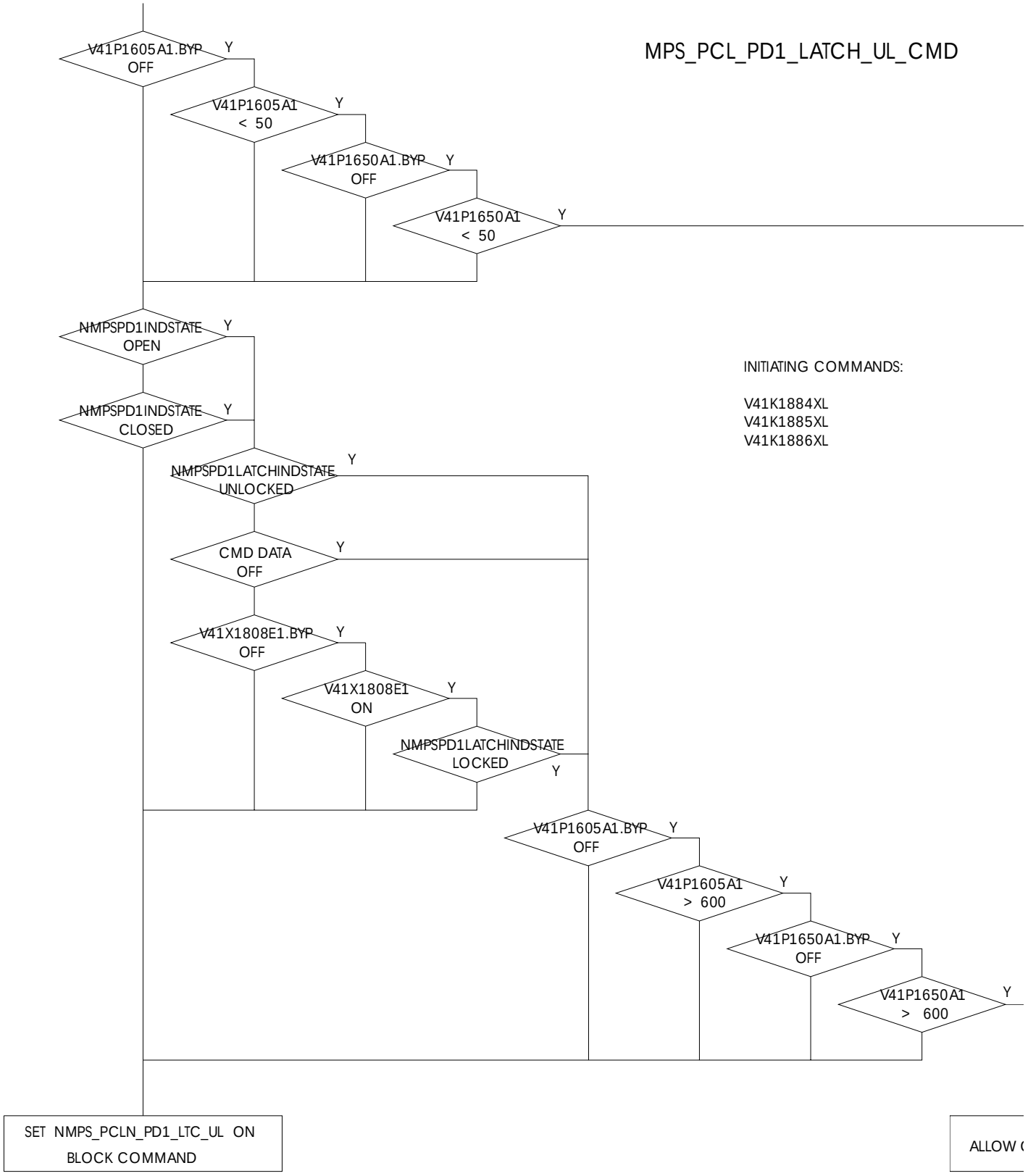
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.4.10.3. Algorithm

3.4.10.3.1. PCL Algorithm

3.4.10.3.1.1. Flowchart

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.4.10.3.1.2. Sequence

```
#include "mps_pcl_pd1_latch_ul_cmd.h"
#include "SME.h"
#include "LO2.h"
#include "MPS.h"
#include "LH2.h"

static const char* seqName = "MPS_PCL_PD1_LATCH_UL_CMD";
static const char* algorithmDescription = "Revision Date - 01/20/2000\n\
  if   (A  || (B && C) || ((D || E) && (G || (F && H) || I) && J && K))\n\
where:   A  = NMPS_PCLO_PD1_LTC_UL - PCL Bypass -  ON\n\
           B  = MPS Pneu Reg Press < 50 psia\n\
           C  = MPS Pneu Accum Press < 50 psia\n\
           D  = PD1 - LO2 ORB/ET FEED DISC VALVE - OPEN\n\
           E  = PD1 - LO2 ORB/ET FEED DISC VALVE - CLOSED\n\
           F  = PD1 - LO2 ORB/ET FEED DISC VALVE LATCH LOCK POWER - ON\n\
           G  = PD1 - LO2 ORB/ET FEED DISC VALVE LATCH - UNLOCKED\n\
           H  = PD1 - LO2 ORB/ET FEED DISC VALVE LATCH - LOCKED\n\
           I   = COMMAND DATA - OFF\n\
           J  = MPS Pneu Reg Press > 600 psia\n\
           K  = MPS Pneu Accum Press > 600 psia";

MPS_PCL_PD1_LATCH_UL_CMD::MPS_PCL_PD1_LATCH_UL_CMD
(const char* seqName):asv_pcsVPCL_DS_ON_OFF_(seqName)
{
  userInit();
}

MPS_PCL_PD1_LATCH_UL_CMD::~MPS_PCL_PD1_LATCH_UL_CMD()
{
}


void MPS_PCL_PD1_LATCH_UL_CMD::userInit(void)
{
  initiatingFD(V41K1884XL);
  initiatingFD(V41K1885XL);
  initiatingFD(V41K1886XL);
}

int MPS_PCL_PD1_LATCH_UL_CMD::userExecute(Discrete::ON_OFF dsVar)
{

if   (A() || (B() && C()) || ((D() || E()) && (G() || (F() && H()) || I()) && J() && K()))
   {
    return PCL_ALLOW_CMD;
   }
else
   {
    NMPS_PCLN_PD1_LTC_UL.Set(Discrete::On);
    return PCL_BLOCK_CMD;
   }
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
}


// Function that determines the state of the indicator bypasses.

int MPS_PCL_PD1_LATCH_UL_CMD::indicatorBypass(PD_ &pseudo_ind)
{
  if ( Discrete::On == pseudo_ind.getValue())
   return IND_BYPASS_ON;
 else
   return IND_BYPASS_OFF;
}



// PCL Bypass - ON

int MPS_PCL_PD1_LATCH_UL_CMD::A()
{
  int  valReturnA = PCL_VOTER_FAIL;

    try
    {
      valReturnA =  (Discrete::On == (NMPS_PCLO_PD1_LTC_UL.getValue()));
    }

    catch(...)
    {
      valReturnA = PCL_VOTER_FAIL;
    }

  return valReturnA;
}




// MPS Pneu Reg Press < 50

int MPS_PCL_PD1_LATCH_UL_CMD::B()
{
  int  valReturnB = PCL_VOTER_FAIL;

  if (indicatorBypass(N41P1605A1))
  {
    try
    {
      valReturnB =  (V41P1605A1.getValue() < 50);
    }

    catch(...)
    {
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
      valReturnB = PCL_VOTER_FAIL;
    }
  }

  else
  {
    valReturnB = PCL_VOTER_FAIL;
  }

  return valReturnB;
}




// MPS Pneu Accum Press < 50

int MPS_PCL_PD1_LATCH_UL_CMD::C()
{
  int  valReturnC = PCL_VOTER_FAIL;

  if (indicatorBypass(N41P1650A1))
  {
    try
    {
      valReturnC =  (V41P1650A1.getValue() < 50);
    }

    catch(...)
    {
      valReturnC = PCL_VOTER_FAIL;
    }
  }

  else
  {
    valReturnC = PCL_VOTER_FAIL;
  }

  return valReturnC;
}



//PD1 - LO2 ORB/ET Feed Disc Valve - OPEN

int MPS_PCL_PD1_LATCH_UL_CMD::D()
{
  int  valReturnD = PCL_VOTER_FAIL;

try
{
            if (NMPSPD1INDSTATE.getValue() == MPS_VALVE_IND_STATE::OPEN)
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
    {
            valReturnD = PCL_VOTER_PASS;
           }
else
           {
      valReturnD = PCL_VOTER_FAIL;
           }
}

catch(...)
{
    valReturnD = PCL_VOTER_FAIL;
}

 return valReturnD;
}



//PD1 - LO2 ORB/ET Feed Disc Valve - CLOSED

int MPS_PCL_PD1_LATCH_UL_CMD::E()
{
 int  valReturnE = PCL_VOTER_FAIL;

try
{
           if (NMPSPD1INDSTATE.getValue() == MPS_VALVE_IND_STATE::CLOSED)
    {
            valReturnE = PCL_VOTER_PASS;
           }
else
           {
      valReturnE = PCL_VOTER_FAIL;
           }
}

catch(...)
{
    valReturnE = PCL_VOTER_FAIL;
}

 return valReturnE;
}


// PD1 LATCH LOCK POWER - ON

int MPS_PCL_PD1_LATCH_UL_CMD::F()
{
 int  valReturnF = PCL_VOTER_FAIL;

 if (indicatorBypass(N41X1808E1))
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
 {
  try
  {
   valReturnF =  (V41X1808E1.getValue() == ON);
  }

  catch(...)
  {
   valReturnF = PCL_VOTER_FAIL;
  }
 }

 else
 {
  valReturnF = PCL_VOTER_FAIL;
 }

 return valReturnF;
}



// PD1 - LO2 ORB/ET Feed Disconnect Latch - Unlocked

int MPS_PCL_PD1_LATCH_UL_CMD::G()
{
 int  valReturnG = PCL_VOTER_FAIL;

try
{
            if (NMPSPD1LATCHINDSTATE.getValue() == MPS_ORB_LATCH_IND::UNLOCKED)
           {
              valReturnG = PCL_VOTER_PASS;
            }
else
            {
      valReturnG = PCL_VOTER_FAIL;
            }
}

catch(...)
{
    valReturnG = PCL_VOTER_FAIL;
}

 return valReturnG;
}



// PD1 - LO2 ORB/ET Feed Disconnect Latch - Locked
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
int MPS_PCL_PD1_LATCH_UL_CMD::H()
{
  int  valReturnH = PCL_VOTER_FAIL;

try
{
            if (NMPSPD1LATCHINDSTATE.getValue() == MPS_ORB_LATCH_IND::LOCKED)
           {
              valReturnH = PCL_VOTER_PASS;
            }
else
            {
       valReturnH = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnH = PCL_VOTER_FAIL;
}

  return valReturnH;
}



// COMMAND DATA - OFF

int MPS_PCL_PD1_LATCH_UL_CMD::I()
{
  int  valReturnI = PCL_VOTER_FAIL;

    try
    {
      valReturnI =  (Discrete::Off == dsVar);
    }

    catch(...)
    {
      valReturnI = PCL_VOTER_FAIL;
    }

  return valReturnI;
}




// MPS Pneu Reg Press > 600

int MPS_PCL_PD1_LATCH_UL_CMD::J()
{
  int  valReturnJ = PCL_VOTER_FAIL;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
  if (indicatorBypass(N41P1605A1))
  {
    try
    {
      valReturnJ =  (V41P1605A1.getValue() > 600);
    }

    catch(...)
    {
      valReturnJ = PCL_VOTER_FAIL;
    }
  }

  else
  {
    valReturnJ = PCL_VOTER_FAIL;
  }

  return valReturnJ;
}




// MPS Pneu Accum Press > 600

int MPS_PCL_PD1_LATCH_UL_CMD::K()
{
  int  valReturnK = PCL_VOTER_FAIL;

  if (indicatorBypass(N41P1650A1))
  {
    try
    {
      valReturnK =  (V41P1650A1.getValue() > 600);
    }

    catch(...)
    {
      valReturnK = PCL_VOTER_FAIL;
    }
  }

  else
  {
    valReturnK = PCL_VOTER_FAIL;
  }

  return valReturnK;
}


MPS_PCL_PD1_LATCH_UL_CMD mps_pcl_pd1_latch_ul_cmd(seqName);
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.4.11. MPS_PCL_LO2_PREVALVES_CL_CMD

3.4.11.1. Description

This PCL provides protection for the MPS LO2 Prevalves (PV1/PV2/PV3) Close Commands to prevent inadvertant closure of the valves during LO2 Load or Drain.
This class inherits from the class CAS_Control_Logic. The Algorithm will be implemented directly in C++ Code.

3.4.11.1.1. Instance

Instance Name
MPS_PCL_LO2_PREVALVES_CL_CMD

3.4.11.1.2. PCL Initiating Commands

Command
V41K1139XL
V41K1140XL
V41K1141XL
V41K1142XL
V41K1239XL
V41K1240XL
V41K1241XL
V41K1242XL
V41K1339XL
V41K1340XL
V41K1341XL
V41K1342XL

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.4.11.1.3. Operations Index

Operation	Description
determineState()	This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE INHIBITED ERROR
activate()	This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.
inhibit()	This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.
getState()	This method is inherited from the base class - CAS_Control_Logic
resetCLNotification()	This method is inherited from the base class - CAS_Control_Logic

3.4.11.2. Operations

3.4.11.2.1. getState()

3.4.11.2.1.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.11.2.2. resetCLNotification()

3.4.11.2.2.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.11.2.3. determineState()

3.4.11.2.3.1. Descriptive Text

This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE, INHIBITED or ERROR.

3.4.11.2.3.2. Sequence

```
if  (<V41K1139XL> .PCLSTAT = active  AND
    <V41K1140XL>. PCLSTAT = active  AND
    <V41K1141XL>. PCLSTAT = active  AND
    <V41K1142XL>. PCLSTAT = active  AND
    <V41K1239XL>. PCLSTAT = active  AND
    <V41K1240XL>. PCLSTAT = active  AND
    <V41K1241XL>. PCLSTAT = active  AND
    <V41K1242XL>. PCLSTAT = active  AND
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
    <V41K1339XL>. PCLSTAT = active  AND
    <V41K1340XL>. PCLSTAT = active  AND
    <V41K1341XL>. PCLSTAT = active  AND
    <V41K1342XL>. PCLSTAT = active  AND)
  {
  CLState = ACTIVE;
  }
else
  {
   if  (<V41K1139XL> .PCLSTAT =  inhibited  AND
       <V41K1140XL>. PCLSTAT =  inhibited  AND
       <V41K1141XL>. PCLSTAT =  inhibited  AND
       <V41K1142XL>. PCLSTAT =  inhibited  AND
       <V41K1239XL>. PCLSTAT =  inhibited  AND
       <V41K1240XL>. PCLSTAT =  inhibited  AND
       <V41K1241XL>. PCLSTAT =  inhibited  AND
       <V41K1242XL>. PCLSTAT =  inhibited  AND
       <V41K1339XL>. PCLSTAT =  inhibited  AND
       <V41K1340XL>. PCLSTAT =  inhibited  AND
       <V41K1341XL>. PCLSTAT =  inhibited  AND
       <V41K1342XL>. PCLSTAT =  inhibited  AND)
     {
     CLState = INHIBITED;
      }
    else
      {
     CLState = ERROR;
      }
   }
```

3.4.11.2.4. activate()

3.4.11.2.4.1. Descriptive Text

This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.11.2.4.2. Sequence

```
A FD <V41K1139XL> PCL;
A FD <V41K1140XL> PCL;
A FD <V41K1141XL> PCL;
A FD <V41K1142XL> PCL;
A FD <V41K1239XL> PCL;
A FD <V41K1240XL> PCL;
A FD <V41K1241XL> PCL;
A FD <V41K1242XL> PCL;
A FD <V41K1339XL> PCL;
A FD <V41K1340XL> PCL;
A FD <V41K1341XL> PCL;
A FD <V41K1342XL> PCL;

return;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.4.11.2.5. inhibit()

3.4.11.2.5.1. Descriptive Text

This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.11.2.5.2. Sequence

```
I FD <V41K1139XL> PCL;
I FD <V41K1140XL> PCL;
I FD <V41K1141XL> PCL;
I FD <V41K1142XL> PCL;
I FD <V41K1239XL> PCL;
I FD <V41K1240XL> PCL;
I FD <V41K1241XL> PCL;
I FD <V41K1242XL> PCL;
I FD <V41K1339XL> PCL;
I FD <V41K1340XL> PCL;
I FD <V41K1341XL> PCL;
I FD <V41K1342XL> PCL;

return;
```

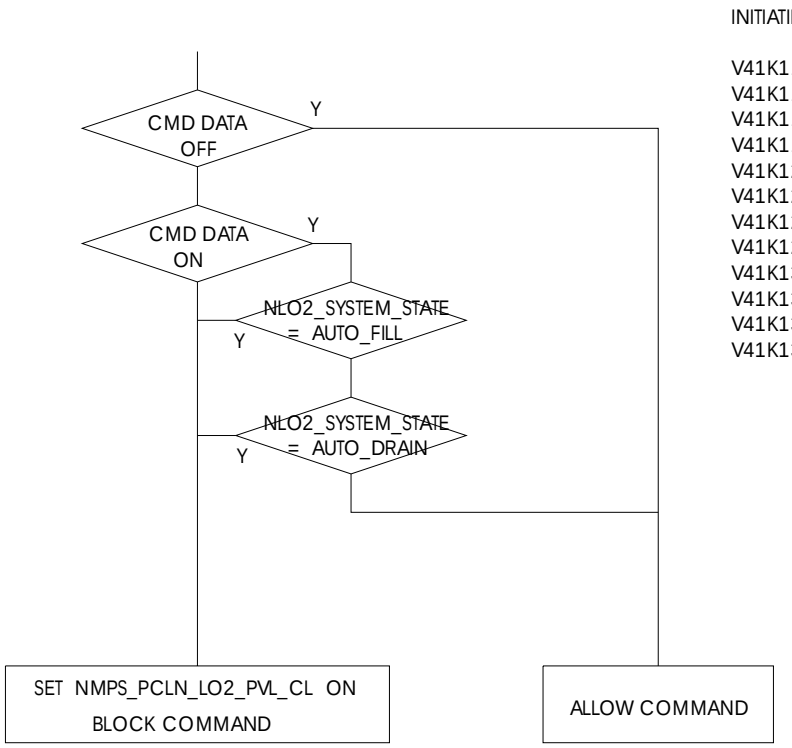

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.4.11.3. Algorithm

3.4.11.3.1. PCL Algorithm

3.4.11.3.1.1. Flowchart

MPS_PCL_LO2_PREVALVES_CL_CMD



3.4.11.3.1.2. Sequence

```
#include "mps_pcl_lo2_prevalves_cl_cmd.h"
#include "SME.h"
#include "LO2.h"
#include "MPS.h"
#include "LH2.h"

static const char* seqName = "MPS_PCL_LO2_PREVALVES_CL_CMD";
static const char* algorithmDescription = "Revision Date - 01/20/2000\n\
if (A  || B || (C && !D && !E))\n\
where:   A  = NMPS_PCLO_LO2_PVL_CL - PCL Bypass ON\n\
         B  = COMMAND DATA - OFF\n\
         C  = COMMAND DATA - ON\n\
         D  = LO2 LOAD IN PROGRESS\n\
         E  = LO2 DRAIN IN PROGRESS";
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
MPS_PCL_LO2_PREVALVES_CL_CMD::MPS_PCL_LO2_PREVALVES_CL_CMD(const char*
seqName):asv_pcsVPCL_DS_ON_OFF_(seqName)
{
  userInit();
}

MPS_PCL_LO2_PREVALVES_CL_CMD::~MPS_PCL_LO2_PREVALVES_CL_CMD()
{
}


void MPS_PCL_LO2_PREVALVES_CL_CMD::userInit(void)
{
  initiatingFD(V41K1139XL);
  initiatingFD(V41K1140XL);
  initiatingFD(V41K1141XL);
  initiatingFD(V41K1142XL);
  initiatingFD(V41K1239XL);
  initiatingFD(V41K1240XL);
  initiatingFD(V41K1241XL);
  initiatingFD(V41K1242XL);
  initiatingFD(V41K1339XL);
  initiatingFD(V41K1340XL);
  initiatingFD(V41K1341XL);
  initiatingFD(V41K1342XL);
}

int MPS_PCL_LO2_PREVALVES_CL_CMD::userExecute(Discrete::ON_OFF dsVar)
{
  if (A()  || B() || (C() && !D() && !E()))
  {
   return PCL_ALLOW_CMD;
  }
  else
  {

   NMPS_PCLN_LO2_PVL_CL.Set(Discrete::On);
   return PCL_BLOCK_CMD;
  }
}




// PCL Override ON

int MPS_PCL_LO2_PREVALVES_CL_CMD::A()
{
  int  valReturnA = PCL_VOTER_FAIL;

    try
    {
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
    valReturnA =  (Discrete::On == (NMPS_PCLO_LO2_PVL_CL.getValue()));
   }

   catch(...)
   {
    valReturnA = PCL_VOTER_FAIL;
   }

 return valReturnA;
}



// COMMAND DATA - OFF

int MPS_PCL_LO2_PREVALVES_CL_CMD::B()
{
 int  valReturnB = PCL_VOTER_FAIL;

   try
   {
    valReturnB =  (Discrete::Off == dsVar);
   }

   catch(...)
   {
    valReturnB = PCL_VOTER_FAIL;
   }

 return valReturnB;
}



// COMMAND DATA - On

int MPS_PCL_LO2_PREVALVES_CL_CMD::C()
{
 int  valReturnC = PCL_VOTER_FAIL;

   try
   {
    valReturnC =  (Discrete::On == dsVar);
   }

   catch(...)
   {
    valReturnC = PCL_VOTER_FAIL;
   }

 return valReturnC;
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
// LO2 LOAD IN PROGRESS

int MPS_PCL_LO2_PREVALVES_CL_CMD::D()
{
  int  valReturnD = PCL_VOTER_FAIL;

    try
    {
      valReturnD =  (Discrete::On == (NLOK0502X.getValue()));
    }

    catch(...)
    {
      valReturnD = PCL_VOTER_FAIL;
    }

  return valReturnD;
}



// LO2 DRAIN IN PROGRESS

int MPS_PCL_LO2_PREVALVES_CL_CMD::E()
{
  int  valReturnE = PCL_VOTER_FAIL;

    try
    {
      valReturnE =  (Discrete::On == (NLOK0505X.getValue()));
    }

    catch(...)
    {
      valReturnE = PCL_VOTER_FAIL;
    }

  return valReturnE;
}



MPS_PCL_LO2_PREVALVES_CL_CMD mps_pcl_lo2_prevalves_cl_cmd(seqName);
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.4.12. MPS_PCL_LO2_PREVALVES_OP_CMD

3.4.12.1. Description

This PCL provides protection for the MPS LO2 Prevalves (PV1/PV2/PV3) Open Commands to prevent inadvertant closure of the valves during LO2 Load or Drain.
This class inherits from the class CAS_Control_Logic. The Algorithm will be implemented directly in C++ Code.

3.4.12.1.1. Instance

Instance Name
MPS_PCL_LO2_PREVALVES_OP_CMD

3.4.12.1.2. PCL Initiating Commands

Command
V41K1136XL
V41K1137XL
V41K1138XL
V41K1143XL
V41K1236XL
V41K1237XL
V41K1238XL
V41K1243XL
V41K1336XL
V41K1337XL
V41K1338XL
V41K1343XL

3.4.12.1.3. Operations Index

Operation	**Description**
determineState()	This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either:

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

	ACTIVE INHIBITED ERROR
activate()	This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.
inhibit()	This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.
getState()	This method is inherited from the base class - CAS_Control_Logic
resetCLNotification()	This method is inherited from the base class - CAS_Control_Logic

3.4.12.2. Operations

3.4.12.2.1. getState()

3.4.12.2.1.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.12.2.2. resetCLNotification()

3.4.12.2.2.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.12.2.3. determineState()

3.4.12.2.3.1. Descriptive Text

This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE, INHIBITED or ERROR.

3.4.12.2.3.2. Sequence

```
if  (<V41K1136XL> .PCLSTAT = active  AND
    <V41K1137XL>. PCLSTAT = active  AND
    <V41K1138XL>. PCLSTAT = active  AND
    <V41K1143XL>. PCLSTAT = active  AND
    <V41K1236XL>. PCLSTAT = active  AND
    <V41K1237XL>. PCLSTAT = active  AND
    <V41K1238XL>. PCLSTAT = active  AND
    <V41K1243XL>. PCLSTAT = active  AND
    <V41K1336XL>. PCLSTAT = active  AND
    <V41K1337XL>. PCLSTAT = active  AND
    <V41K1338XL>. PCLSTAT = active  AND
    <V41K1343XL>. PCLSTAT = active  AND)
  {
  CLState = ACTIVE;
  }
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
else
  {
  if  (<V41K1136XL> .PCLSTAT =  inhibited  AND
       <V41K1137XL>. PCLSTAT =  inhibited  AND
       <V41K1138XL>. PCLSTAT =  inhibited  AND
       <V41K1143XL>. PCLSTAT =  inhibited  AND
       <V41K1236XL>. PCLSTAT =  inhibited  AND
       <V41K1237XL>. PCLSTAT =  inhibited  AND
       <V41K1238XL>. PCLSTAT =  inhibited  AND
       <V41K1243XL>. PCLSTAT =  inhibited  AND
       <V41K1336XL>. PCLSTAT =  inhibited  AND
       <V41K1337XL>. PCLSTAT =  inhibited  AND
       <V41K1338XL>. PCLSTAT =  inhibited  AND
       <V41K1343XL>. PCLSTAT =  inhibited  AND)
    {
    CLState = INHIBITED;
     }
    else
     {
    CLState = ERROR;
     }
  }
```

3.4.12.2.4. activate()

3.4.12.2.4.1. Descriptive Text

This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.12.2.4.2. Sequence

```
A FD <V41K1136XL> PCL;
A FD <V41K1137XL> PCL;
A FD <V41K1138XL> PCL;
A FD <V41K1143XL> PCL;
A FD <V41K1236XL> PCL;
A FD <V41K1237XL> PCL;
A FD <V41K1238XL> PCL;
A FD <V41K1243XL> PCL;
A FD <V41K1336XL> PCL;
A FD <V41K1337XL> PCL;
A FD <V41K1338XL> PCL;
A FD <V41K1343XL> PCL;

return;
```

3.4.12.2.5. inhibit()

3.4.12.2.5.1. Descriptive Text

This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.4.12.2.5.2. Sequence

I FD <V41K1136XL> PCL;
I FD <V41K1137XL> PCL;
I FD <V41K1138XL> PCL;
I FD <V41K1143XL> PCL;
I FD <V41K1236XL> PCL;
I FD <V41K1237XL> PCL;
I FD <V41K1238XL> PCL;
I FD <V41K1243XL> PCL;
I FD <V41K1336XL> PCL;
I FD <V41K1337XL> PCL;
I FD <V41K1338XL> PCL;
I FD <V41K1343XL> PCL;

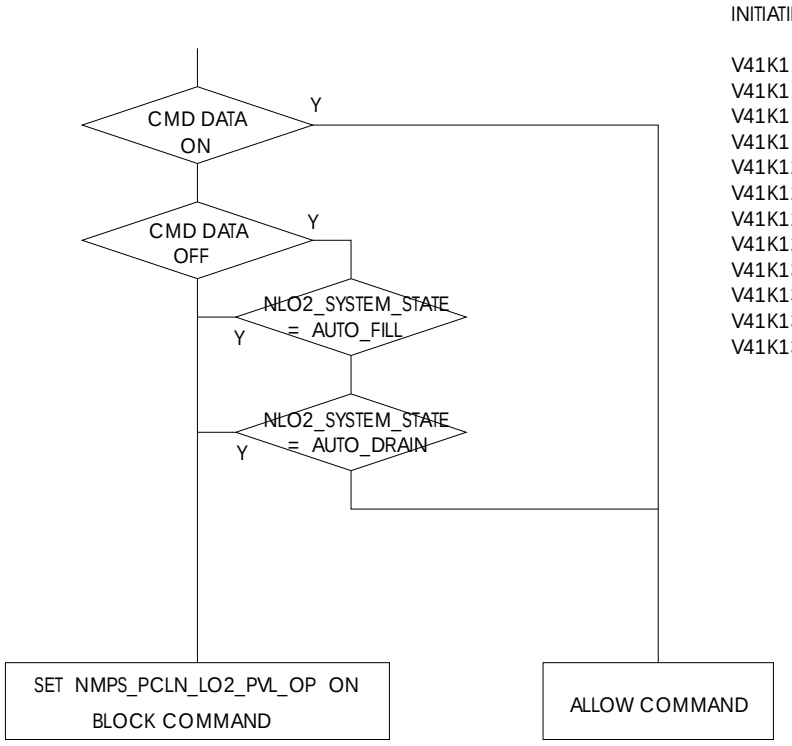
return;

3.4.12.3. Algorithm

3.4.12.3.1. PCL Algorithm

3.4.12.3.1.1. Flowchart

MPS_PCL_LO2_PREVALVES_OP_CMD



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.4.12.3.1.2. Sequence

```
#include "mps_pcl_lo2_prevalves_op_cmd.h"
#include "SME.h"
#include "LO2.h"
#include "MPS.h"
#include "LH2.h"

static const char* seqName = "MPS_PCL_LO2_PREVALVES_OP_CMD";
static const char* algorithmDescription = "Revision Date - 01/20/2000\n\
if (A  || B || (C && !D && !E))\n\
where:   A  = NMPS_PCLO_LO2_PVL_OP - PCL Bypass ON\n\
         B  = COMMAND DATA - ON\n\
         C  = COMMAND DATA - OFF\n\
         D  = LO2 LOAD IN PROGRESS\n\
         E  = LO2 DRAIN IN PROGRESS";

MPS_PCL_LO2_PREVALVES_OP_CMD::MPS_PCL_LO2_PREVALVES_OP_CMD(const char*
seqName):asv_pcsVPCL_DS_ON_OFF_(seqName)
{
  userInit();
}

MPS_PCL_LO2_PREVALVES_OP_CMD::~MPS_PCL_LO2_PREVALVES_OP_CMD()
{
}


void MPS_PCL_LO2_PREVALVES_OP_CMD::userInit(void)
{
  initiatingFD(V41K1136XL);
  initiatingFD(V41K1137XL);
  initiatingFD(V41K1138XL);
  initiatingFD(V41K1143XL);
  initiatingFD(V41K1236XL);
  initiatingFD(V41K1237XL);
  initiatingFD(V41K1238XL);
  initiatingFD(V41K1243XL);
  initiatingFD(V41K1336XL);
  initiatingFD(V41K1337XL);
  initiatingFD(V41K1338XL);
  initiatingFD(V41K1343XL);
}

int MPS_PCL_LO2_PREVALVES_OP_CMD::userExecute(Discrete::ON_OFF dsVar)
{
  if (A()  || B() || (C() && !D() && !E()))
  {
   return PCL_ALLOW_CMD;
  }
  else
  {
   NMPS_PCLO_LO2_PVL_OP.Set(Discrete::On);
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
   return PCL_BLOCK_CMD;
  }
}
```

```
// PCL Override ON

int MPS_PCL_LO2_PREVALVES_OP_CMD::A()
{
  int  valReturnA = PCL_VOTER_FAIL;

    try
    {
      valReturnA =  (Discrete::On == (NMPS_PCLO_LO2_PVL_OP.getValue()));
    }

    catch(...)
    {
      valReturnA = PCL_VOTER_FAIL;
    }

  return valReturnA;
}
```

```
// COMMAND DATA - ON

int MPS_PCL_LO2_PREVALVES_OP_CMD::B()
{
  int  valReturnB = PCL_VOTER_FAIL;

    try
    {
      valReturnB =  (Discrete::On == dsVar);
    }

    catch(...)
    {
      valReturnB = PCL_VOTER_FAIL;
    }

  return valReturnB;
}
```

```
// COMMAND DATA - OFF
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
int MPS_PCL_LO2_PREVALVES_OP_CMD::C()
{
  int  valReturnC = PCL_VOTER_FAIL;

    try
    {
      valReturnC =  (Discrete::Off == dsVar);
    }

    catch(...)
    {
      valReturnC = PCL_VOTER_FAIL;
    }

  return valReturnC;
}



// LO2 LOAD IN PROGRESS

int MPS_PCL_LO2_PREVALVES_OP_CMD::D()
{
  int  valReturnD = PCL_VOTER_FAIL;

    try
    {
      valReturnD =  (Discrete::On == (NLOK0502X.getValue()));
    }

    catch(...)
    {
      valReturnD = PCL_VOTER_FAIL;
    }

  return valReturnD;
}



// LO2 DRAIN IN PROGRESS

int MPS_PCL_LO2_PREVALVES_OP_CMD::E()
{
  int  valReturnE = PCL_VOTER_FAIL;

    try
    {
      valReturnE =  (Discrete::On == (NLOK0505X.getValue()));
    }

    catch(...)
    {
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
    valReturnE = PCL_VOTER_FAIL;
   }

 return valReturnE;
}
```

MPS_PCL_LO2_PREVALVES_OP_CMD mps_pcl_lo2_prevalves_op_cmd(seqName);

3.4.13. MPS_PCL_LO2_POGO_CL_CMD

3.4.13.1. Description

This PCL provides protection for the MPS LO2 POGO Valves (PV20/PV21) Close Commands to prevent inadvertant opening of the valves during LO2 Load or Drain.
This class inherits from the class CAS_Control_Logic. The Algorithm will be implemented directly in C++ Code.

3.4.13.1.1. Instance

Instance Name
MPS_PCL_LO2_POGO_CL_CMD

3.4.13.1.2. PCL Initiating Commands

Command
V41K1815XL
V41K1816XL
V41K1825XL
V41K1826XL

3.4.13.1.3. Operations Index

Operation	**Description**
determineState()	This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either:

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

	ACTIVE INHIBITED ERROR
activate()	This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.
inhibit()	This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.
getState()	This method is inherited from the base class - CAS_Control_Logic
resetCLNotification()	This method is inherited from the base class - CAS_Control_Logic

3.4.13.2. Operations

3.4.13.2.1. getState()

3.4.13.2.1.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.13.2.2. resetCLNotification()

3.4.13.2.2.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.13.2.3. determineState()

3.4.13.2.3.1. Descriptive Text

This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE, INHIBITED or ERROR.

3.4.13.2.3.2. Sequence

```
if  (<V41K1815XL>.PCLSTAT = active  AND
    <V41K1816XL>.PCLSTAT = active  AND
    <V41K1825XL>.PCLSTAT = active  AND
    <V41K1826XL>.PCLSTAT = active)
  {
  CLState = ACTIVE;
  }
else
  {
   if  (<V41K1815XL>.PCLSTAT = inhibited AND
       <V41K1816XL>.PCLSTAT = inhibited AND
       <V41K1825XL>.PCLSTAT = inhibited AND
       <V41K1826XL>.PCLSTAT = inhibited)
    {
    CLState = INHIBITED;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
    }
  else
    {
   CLState = ERROR;
    }
 }
```

3.4.13.2.4. activate()

3.4.13.2.4.1. Descriptive Text

This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.13.2.4.2. Sequence

```
A FD <V41K1815XL> PCL;
A FD <V41K1816XL> PCL;
A FD <V41K1825XL> PCL;
A FD <V41K1826XL> PCL;

return;
```

3.4.13.2.5. inhibit()

3.4.13.2.5.1. Descriptive Text

This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.13.2.5.2. Sequence

```
I FD <V41K1815XL> PCL;
I FD <V41K1816XL> PCL;
I FD <V41K1825XL> PCL;
I FD <V41K1826XL> PCL;

return;
```

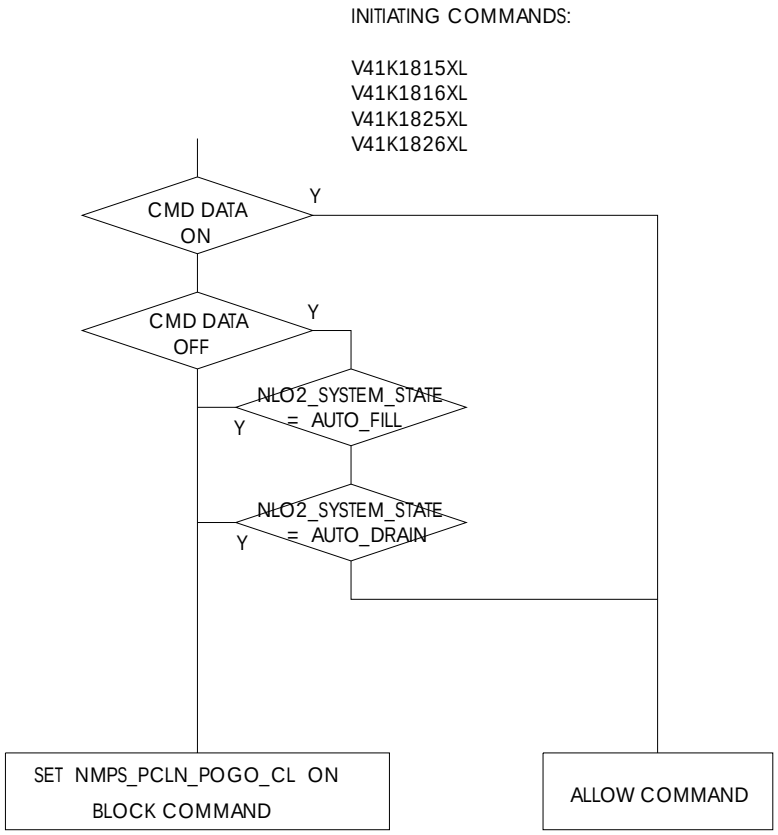

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.4.13.3. Algorithm

3.4.13.3.1. PCL Algorithm

3.4.13.3.1.1. Flowchart

MPS_PCL_LO2_POGO_CL_CMD



3.4.13.3.1.2. Sequence

```
#include "mps_pcl_lo2_pogo_cl_cmd.h"
#include "SME.h"
#include "LO2.h"
#include "MPS.h"
#include "LH2.h"

static const char* seqName = "MPS_PCL_LO2_POGO_CL_CMD";
static const char* algorithmDescription = "Revision Date - 01/20/2000\n\
if (A  || B || (C && !D && !E))\n\
where:   A  = NMPS_PCLO_POGO_CL - PCL Bypass ON\n\
         B  = COMMAND DATA - ON\n\
         C  = COMMAND DATA - OFF\n\
         D  = LO2 LOAD IN PROGRESS\n\
         E  = LO2 DRAIN IN PROGRESS";
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
MPS_PCL_LO2_POGO_CL_CMD::MPS_PCL_LO2_POGO_CL_CMD(const char*
seqName):asv_pcsVPCL_DS_ON_OFF_(seqName)
{
  userInit();
}

MPS_PCL_LO2_POGO_CL_CMD::~MPS_PCL_LO2_POGO_CL_CMD()
{
}


void MPS_PCL_LO2_POGO_CL_CMD::userInit(void)
{
  initiatingFD(V41K1815XL);
  initiatingFD(V41K1816XL);
  initiatingFD(V41K1825XL);
  initiatingFD(V41K1826XL);
}

int MPS_PCL_LO2_POGO_CL_CMD::userExecute(Discrete::ON_OFF dsVar)
{
  if (A()  || B() || (C() && !D() && !E()))
  {
   return PCL_ALLOW_CMD;
  }
  else
  {
   NMPS_PCLN_POGO_CL.Set(Discrete::On);
   return PCL_BLOCK_CMD;
  }
}




// PCL Override ON

int MPS_PCL_LO2_POGO_CL_CMD::A()
{
  int  valReturnA = PCL_VOTER_FAIL;

    try
    {
      valReturnA =   (Discrete::On == (NMPS_PCLO_POGO_CL.getValue()));
    }

    catch(...)
    {
      valReturnA = PCL_VOTER_FAIL;
    }

  return valReturnA;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
}



// COMMAND DATA - ON

int MPS_PCL_LO2_POGO_CL_CMD::B()
{
  int  valReturnB = PCL_VOTER_FAIL;

    try
    {
      valReturnB =  (Discrete::On == dsVar);
    }

    catch(...)
    {
      valReturnB = PCL_VOTER_FAIL;
    }

  return valReturnB;
}



// COMMAND DATA - OFF

int MPS_PCL_LO2_POGO_CL_CMD::C()
{
  int  valReturnC = PCL_VOTER_FAIL;

    try
    {
      valReturnC =  (Discrete::Off == dsVar);
    }

    catch(...)
    {
      valReturnC = PCL_VOTER_FAIL;
    }

  return valReturnC;
}



// LO2 LOAD IN PROGRESS

int MPS_PCL_LO2_POGO_CL_CMD::D()
{
  int  valReturnD = PCL_VOTER_FAIL;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
   try
   {
    valReturnD =  (Discrete::On == (NLOK0502X.getValue()));
   }

   catch(...)
   {
    valReturnD = PCL_VOTER_FAIL;
   }

  return valReturnD;
}



// LO2 DRAIN IN PROGRESS

int MPS_PCL_LO2_POGO_CL_CMD::E()
{
  int  valReturnE = PCL_VOTER_FAIL;

   try
   {
    valReturnE =  (Discrete::On == (NLOK0505X.getValue()));
   }

   catch(...)
   {
    valReturnE = PCL_VOTER_FAIL;
   }

  return valReturnE;
}



MPS_PCL_LO2_POGO_CL_CMD mps_pcl_lo2_pogo_cl_cmd(seqName);
```

3.4.14. MPS_PCL_PV11_CL_CMD

3.4.14.1. Description

This PCL provides protection for the MPS LH2 Outboard Fill and Drain Valve Close Command.
This class inherits from the class CAS_Control_Logic. The Algorithm will be implemented directly in C++ Code.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

Reference - LPS Program GCH01/02

3.4.14.1.1. Instance

Instance Name
MPS_PCL_PV11_CL_CMD

3.4.14.1.2. PCL Initiating Commands

Command
V41K1393XL

3.4.14.1.3. Operations Index

Operation	**Description**
determineState()	This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE INHIBITED ERROR
activate()	This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.
inhibit()	This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.
getState()	This method is inherited from the base class - CAS_Control_Logic
resetCLNotification()	This method is inherited from the base class - CAS_Control_Logic

3.4.14.2. Operations

3.4.14.2.1. getState()

3.4.14.2.1.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.14.2.2. resetCLNotification()

3.4.14.2.2.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.4.14.2.3. determineState()

3.4.14.2.3.1. Descriptive Text

This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE, INHIBITED or ERROR.

3.4.14.2.3.2. Sequence

```
if  (<V41K1393XL>.PCLSTAT = active)
  {
  CLState = ACTIVE;
  }
else
  {
   if  (<V41K1393XL>.PCLSTAT = inhibited)
     {
     CLState = INHIBITED;
      }
    else
      {
     CLState = ERROR;
      }
   }
```

3.4.14.2.4. activate()

3.4.14.2.4.1. Descriptive Text

This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.14.2.4.2. Sequence

```
A FD <V41K1393XL> PCL;

return;
```

3.4.14.2.5. inhibit()

3.4.14.2.5.1. Descriptive Text

This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.14.2.5.2. Sequence

```
I FD <V41K1393XL> PCL;

return;
```

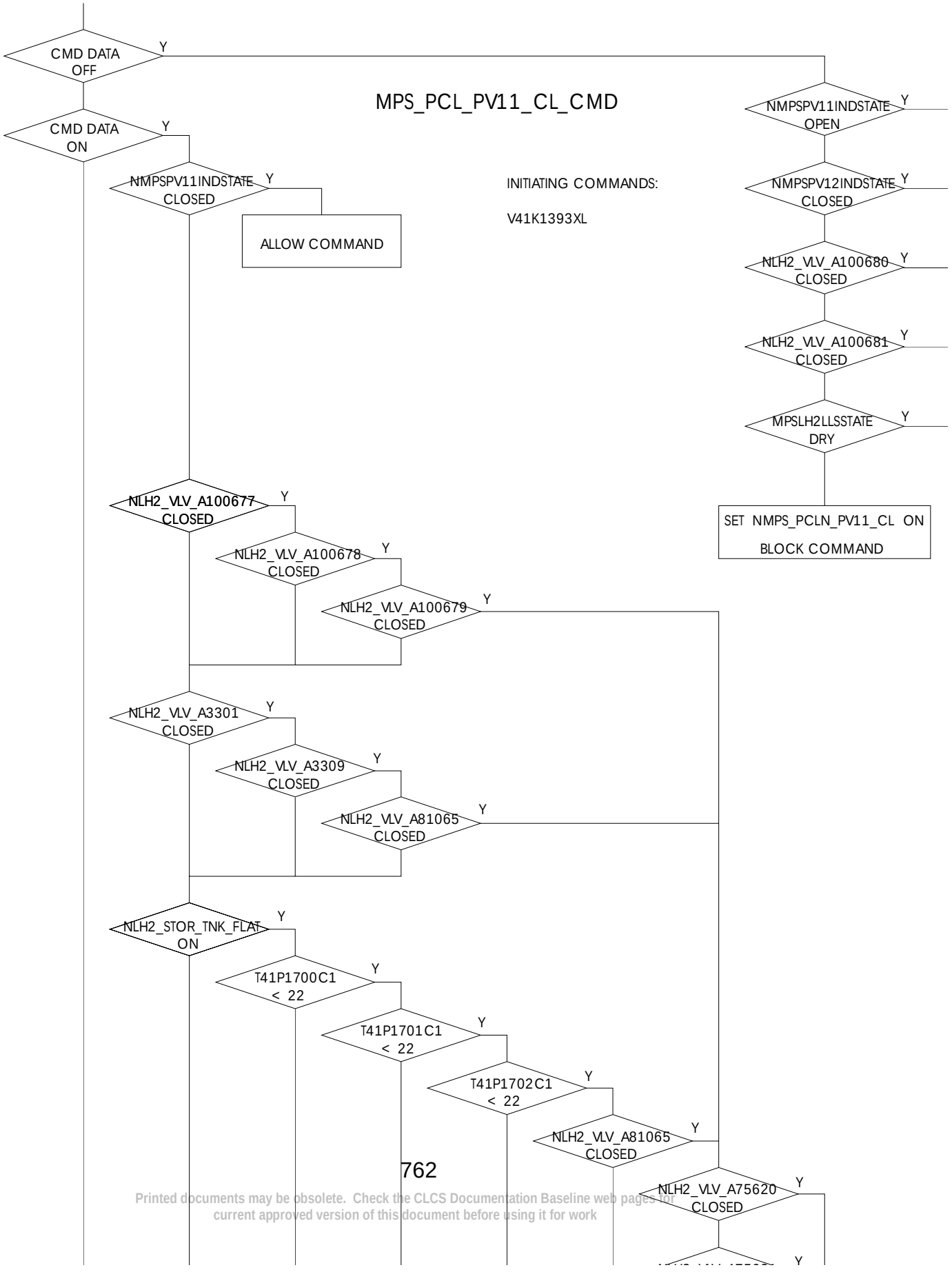

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.4.14.3. Algorithm

3.4.14.3.1. PCL Algorithm

3.4.14.3.1.1. Flowchart

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.4.14.3.1.2. Sequence

```
#include "mps_pcl_pv11_cl_cmd.h"
#include "SME.h"
#include "LH2.h"
#include "MPS.h"
#include "LO2.h"

static const char* seqName = "MPS_PCL_PV11_CL_CMD";
static const char* algorithmDescription = "Revision Date - 01/20/2000\n\
if (A) ||\n\
if Command_Data is ON &&\n\
  if (B || (((C && D && E) || (F && G && H) || (I && J && H)) && (K || L)&& (M || N))) ||\n\
if Command_Data is OFF &&\n\
  if (O || P || Q || R || S)\n\
where:   A  = NMPS_PCLO_PV11_CL - PCL Bypass ON\n\
          B  = PV11 - LH2 OUTBOARD F/D VALVE - CLOSED\n\
          C  = A100677 - LH2 MAIN FILL VALVE - CLOSED\n\
          D  = A100678 - LH2 AUX FILL VALVE - CLOSED\n\
          E  = A100679 - LH2 REPLENISH VALVE - CLOSED\n\
          F  = A3301 - LH2 XFER LINE VALVE - CLOSED\n\
          G = A3309 - LH2 XFER LINE CHILLDOWN VALVE - CLOSED\n\
          H = A81065 - LH2 FILL LINE PURGE VALVE - CLOSED\n\
          I   = LH2 STORAGE TANK PRESS - FLAT\n\
          J  = ET LH2 ULL PRESS < 22\n\
          K = A75620 - XFER LINE PURGE SOV - CLOSED\n\
          L  = A75621 - XFER LINE PURGE VALVE - CLOSED\n\
          M = A75567 - 2K HE RMT SOL SOV - CLOSED\n\
          N = A75586 - 2K HE RMT SOV - CLOSED\n\
          O = PV11 - LH2 OUTBOARD F/D VALVE - OPEN\n\
          P = A100680 - LH2 TSM DRAIN VALVE - CLOSED\n\
          Q = A100681 - LH2 TSM DRAIN SOV - CLOSED\n\
          R  = PV12 - LH2 INBOARD F/D VALVE - CLOSED\n\
          S =  ET LH2 LLS - DRY";

MPS_PCL_PV11_CL_CMD::MPS_PCL_PV11_CL_CMD(const char*
seqName):asv_pcsVPCL_DS_ON_OFF_(seqName)
{
  userInit();
}

MPS_PCL_PV11_CL_CMD::~MPS_PCL_PV11_CL_CMD()
{
}


void MPS_PCL_PV11_CL_CMD::userInit(void)
{
  initiatingFD(V41K1393XL);
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
int MPS_PCL_PV11_CL_CMD::userExecute(Discrete::ON_OFF dsVar)
{

 if  (Discrete::On == (NMPS_PCLO_PV11_CL.getValue()))
{
 return PCL_ALLOW_CMD;
{

 if (Discrete::On==dsVar)
{
  if (B() || (((C() && D() && E()) || (F() && G() && H()) || (I() && J() && H())) && (K() || L())&& (M() || N())))
  {
   return PCL_ALLOW_CMD;
  }
  else
  {
   NMPS_PCLN_PV11_CL.Set(Discrete::On);
   return PCL_BLOCK_CMD;
  }
}
if (Discrete::Off==dsVar)
}
if (O() || P() || Q() || R() || S())
  {
   return PCL_ALLOW_CMD;
  }
else
  {
   NMPS_PCLN_PV11_CL.Set(Discrete::On);
   return PCL_BLOCK_CMD;
  }
}

}



// PV11 - LH2 Outboard F/D Valve - CLOSED

int MPS_PCL_PV11_CL_CMD::B()
{
  int  valReturnB = PCL_VOTER_FAIL;

try
{
            if (NMPSPV11INDSTATE.getValue() == MPS_VALVE_IND_STATE::CLOSED)
     {
              valReturnB = PCL_VOTER_PASS;
             }
else
            {
       valReturnB = PCL_VOTER_FAIL;
            }
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
catch(...)
{
    valReturnB = PCL_VOTER_FAIL;
}

 return valReturnB;
}




// A100677 - LH2 Main Fill valve - CLOSED

int MPS_PCL_PV11_CL_CMD::C()
{
int  valReturnC = PCL_VOTER_FAIL;

try
{
          if (NLH2_VLV_A100677.getValue() == LH2_VALVE_STATE::CLOSED)
    {
            valReturnC = PCL_VOTER_PASS;
           }
else
          {
      valReturnC = PCL_VOTER_FAIL;
          }
}

catch(...)
{
    valReturnC = PCL_VOTER_FAIL;
}

return valReturnC;
}




// A100678 - LH2 Aux Fill valve - CLOSED

int MPS_PCL_PV11_CL_CMD::D()
{
int  valReturnD = PCL_VOTER_FAIL;

try
{
          if (NLH2_VLV_A100678.getValue() == LH2_VALVE_STATE::CLOSED)
    {
            valReturnD = PCL_VOTER_PASS;
           }
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
else
            {
       valReturnD = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnD = PCL_VOTER_FAIL;
}

return valReturnD;
}



// A100679 - LH2 Replenish Valve - CLOSED

int MPS_PCL_PV11_CL_CMD::E()
{
int  valReturnE = PCL_VOTER_FAIL;

try
{
            if (NLH2_VLV_A100679.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnE = PCL_VOTER_PASS;
             }
else
            {
       valReturnE = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnE = PCL_VOTER_FAIL;
}

return valReturnE;
}



// A3301 - LH2 Xfer Line Valve - CLOSED

int MPS_PCL_PV11_CL_CMD::F()
{
int  valReturnF = PCL_VOTER_FAIL;

try
{
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
            if (NLH2_VLV_A3301.getValue() == LH2_VALVE_STATE::CLOSED)
    {
              valReturnF = PCL_VOTER_PASS;
             }
else
            {
      valReturnF = PCL_VOTER_FAIL;
            }
}

catch(...)
{
    valReturnF = PCL_VOTER_FAIL;
}

return valReturnF;
}
```

```
// A3309 - LH2 Xfer Line Chilldown Valve - CLOSED

int MPS_PCL_PV11_CL_CMD::G()
{
int  valReturnG = PCL_VOTER_FAIL;

try
{
            if (NLH2_VLV_A3309.getValue() == LH2_VALVE_STATE::CLOSED)
    {
              valReturnG = PCL_VOTER_PASS;
             }
else
            {
      valReturnG = PCL_VOTER_FAIL;
            }
}

catch(...)
{
    valReturnG = PCL_VOTER_FAIL;
}

return valReturnG;
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
// A81065 - LH2 Fill Line Purge Valve - CLOSED

int MPS_PCL_PV11_CL_CMD::H()
{
int  valReturnH = PCL_VOTER_FAIL;

try
{
            if (NLH2_VLV_A81065.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnH = PCL_VOTER_PASS;
             }
else
            {
       valReturnH = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnH = PCL_VOTER_FAIL;
}

return valReturnH;
}



// LH2 STORAGE TANK PRESS - FLAT

int MPS_PCL_PV11_CL_CMD::I()
{
  int  valReturnI = PCL_VOTER_FAIL;

try
{
            if (NLH2_STOR_TNK_PRESS.getValue() == FLAT)
     {
              valReturnI = PCL_VOTER_PASS;
             }
else
            {
       valReturnI = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnI = PCL_VOTER_FAIL;
}

  return valReturnI;
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
// ET LH2 Ullage < 22 psia

int MPS_PCL_PV11_CL_CMD::J()
{
  int  valReturnJ = PCL_VOTER_FAIL;

try
{
            if (T41P1700C1.getvalue() < 22)
            {
               if (T41P1701C1.getvalue() < 22)
               {
                  if (T41P1701C1.getvalue() < 22)
                  {
                    valReturnJ = PCL_VOTER_PASS;
                    }
                  else
                    {
                  valReturnJ = PCL_VOTER_FAIL;
                    }
              {
             else
               {
              valReturnJ = PCL_VOTER_FAIL;
               }
          {
        else
           {
          valReturnJ = PCL_VOTER_FAIL;
          }
}

 catch(...)
    {
     valReturnJ = PCL_VOTER_FAIL;
    }

  return valReturnJ;
}




// A75620 - Xfer Line Purge SOV - CLOSED

int MPS_PCL_PV11_CL_CMD::K()
{
  int  valReturnK = PCL_VOTER_FAIL;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
try
{
            if (NLH2_VLV_A75620.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnK = PCL_VOTER_PASS;
             }
else
            {
       valReturnK = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnK = PCL_VOTER_FAIL;
}

return valReturnK;
}



// A75621 - Xfer Line Purge Valve - CLOSED

int MPS_PCL_PV11_CL_CMD::L()
{
  int  valReturnL = PCL_VOTER_FAIL;

try
{
            if (NLH2_VLV_A75621.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnL = PCL_VOTER_PASS;
             }
else
            {
       valReturnL = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnL = PCL_VOTER_FAIL;
}

  return valReturnL;
}



// A75567 - 2K He RMT SOL SOV - CLOSED
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
int MPS_PCL_PV11_CL_CMD::M()
{
  int  valReturnM = PCL_VOTER_FAIL;

 try
{
            if (NLH2_VLV_A75567.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnM = PCL_VOTER_PASS;
             }
else
            {
       valReturnM = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnM = PCL_VOTER_FAIL;
}

  return valReturnM;
}



// A75586 - 2K He RMT SOV - CLOSED

int MPS_PCL_PV11_CL_CMD::N()
{
  int  valReturnN = PCL_VOTER_FAIL;

try
{
            if (NLH2_VLV_A75586.getValue() == LH2_VLAVE_STATE::CLOSED)
     {
              valReturnN = PCL_VOTER_PASS;
             }
else
            {
       valReturnN = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnN = PCL_VOTER_FAIL;
}

  return valReturnN;
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
// PV11 - LH2 Outboard F/D Valve - OPEN

int MPS_PCL_PV11_CL_CMD::O()
{
  int  valReturnO = PCL_VOTER_FAIL;

try
{
            if (NMPSPV11INDSTATE.getValue() == MPS_VALVE_IND_STATE::OPEN)
     {
              valReturnO = PCL_VOTER_PASS;
             }
else
            {
       valReturnO = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnO = PCL_VOTER_FAIL;
}

  return valReturnO;
}




// A100680 - LH2 TSM Drain Valve - CLOSED

int MPS_PCL_PV11_CL_CMD::P()
{
int  valReturnP = PCL_VOTER_FAIL;

try
{
            if (NLH2_VLV_A100680.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnP = PCL_VOTER_PASS;
             }
else
            {
       valReturnP = PCL_VOTER_FAIL;
            }
}

catch(...)
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
{
    valReturnP = PCL_VOTER_FAIL;
}

return valReturnP;
}



// A100681 - LH2 TSM Drain SOV - CLOSED

int MPS_PCL_PV11_CL_CMD::Q()
{
int  valReturnQ = PCL_VOTER_FAIL;

try
{
            if (NLH2_VLV_A100681.getValue() == LH2_VALVE_STATE::CLOSED)
    {
              valReturnQ = PCL_VOTER_PASS;
             }
else
            {
       valReturnQ = PCL_VOTER_FAIL;
            }
}

catch(...)
{
    valReturnQ = PCL_VOTER_FAIL;
}

return valReturnQ;
}



// PV12 - LH2 Inboard F/D Valve - CLOSED

int MPS_PCL_PV11_CL_CMD::R()
{
  int  valReturnR = PCL_VOTER_FAIL;

try
{
            if (NMPSPV12INDSTATE.getValue() == MPS_VALVE_IND_STATE::CLOSED)
    {
              valReturnR = PCL_VOTER_PASS;
             }
else
            {
       valReturnR = PCL_VOTER_FAIL;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
            }
}

catch(...)
{
     valReturnR = PCL_VOTER_FAIL;
}

  return valReturnR;
}



// ET LH2 LLS - DRY

int MPS_PCL_PV11_CL_CMD::S()
{
  int  valReturnS = PCL_VOTER_FAIL;

try
{
             if (NMPSLH2LLSSTATE.getValue() == MPS_LLS_COMPOSITE_DRY)

            {
               valReturnS = PCL_VOTER_PASS;
              }
else
             {
       valReturnS = PCL_VOTER_FAIL;
             }
}

catch(...)
{
     valReturnS = PCL_VOTER_FAIL;
}

  return valReturnS;
}




MPS_PCL_PV11_CL_CMD mps_pcl_pv11_cl_cmd(seqName);
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.4.15. MPS_PCL_PV11_OP_CMD

3.4.15.1. Description

This PCL provides protection for the MPS LH2 Outboard Fill and Drain Valve Open Commands.
This class inherits from the class CAS_Control_Logic. The Algorithm will be implemented directly in C++ Code.

Reference - LPS Program GCH01/02

3.4.15.1.1. Instance

Instance Name
MPS_PCL_PV11_OP_CMD

3.4.15.1.2. PCL Initiating Commands

Command
V41K1391XL
V41K1391NL

3.4.15.1.3. Operations Index

Operation	**Description**
determineState()	This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE INHIBITED ERROR
activate()	This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.
inhibit()	This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.
getState()	This method is inherited from the base class - CAS_Control_Logic
resetCLNotification()	This method is inherited from the base class - CAS_Control_Logic

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.4.15.2. Operations

3.4.15.2.1. getState()

3.4.15.2.1.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.15.2.2. resetCLNotification()

3.4.15.2.2.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.15.2.3. determineState()

3.4.15.2.3.1. Descriptive Text

This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE, INHIBITED or ERROR.

3.4.15.2.3.2. Sequence

```
if  (<V41K1391XL>.PCLSTAT = active  AND
     <V41K1391NL>.PCLSTAT = active)
  {
  CLState = ACTIVE;
  }
else
  {
   if  (<V41K1391XL>.PCLSTAT = inhibited AND
        <V41K1391NL>.PCLSTAT = inhibited)
     {
     CLState = INHIBITED;
      }
    else
      {
     CLState = ERROR;
      }
   }
```

3.4.15.2.4. activate()

3.4.15.2.4.1. Descriptive Text

This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.15.2.4.2. Sequence

```
A FD <V41K1391XL> PCL;
A FD <V41K1391NL> PCL;

return;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.4.15.2.5. inhibit()

3.4.15.2.5.1. Descriptive Text

This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.15.2.5.2. Sequence

I FD <V41K1391XL> PCL;
I FD <V41K1391NL> PCL;

return;

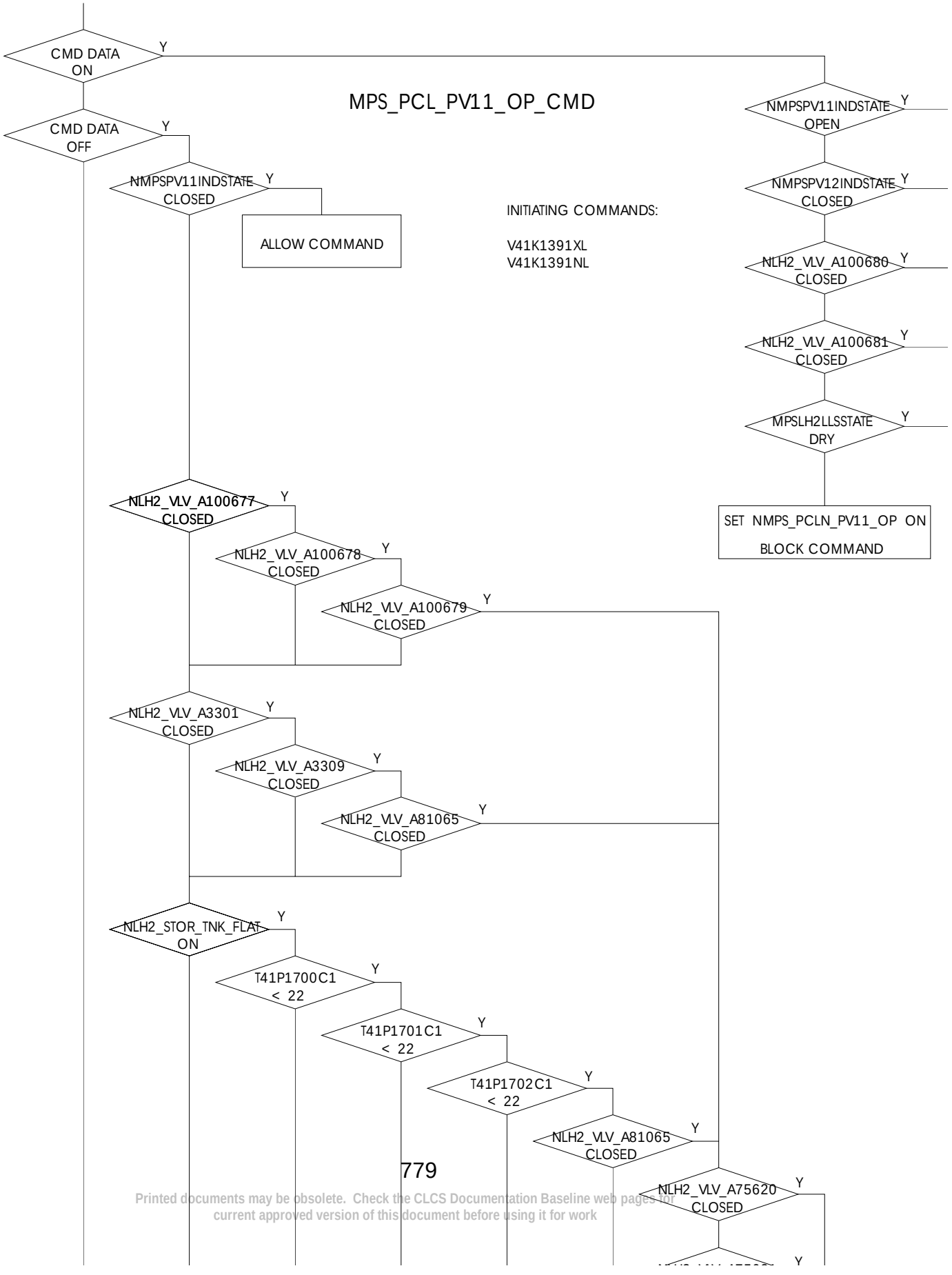
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.4.15.3. Algorithm

3.4.15.3.1. PCL Algorithm

3.4.15.3.1.1. Flowchart

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.4.15.3.1.2. Sequence

```
#include "mps_pcl_pv11_op_cmd.h"
#include "SME.h"
#include "LH2.h"
#include "MPS.h"
#include "LO2.h"

static const char* seqName = "MPS_PCL_PV11_OP_CMD";
static const char* algorithmDescription = "Revision Date - 01/20/2000\n\
if (A) ||\n\
if Command_Data is OFF &&\n\
  if (B || (((C && D && E) || (F && G && H) || (I && J && H)) && (K || L)&& (M || N))) ||\n\
if Command_Data is ON &&\n\
  if (O || P || Q || R || S)\n\
where:   A  = NMPS_PCLO_PV11_OP - PCL Bypass ON\n\
          B  = PV11 - LH2 OUTBOARD F/D VALVE - CLOSED\n\
          C  = A100677 - LH2 MAIN FILL VALVE - CLOSED\n\
          D  = A100678 - LH2 AUX FILL VALVE - CLOSED\n\
          E  = A100679 - LH2 REPLENISH VALVE - CLOSED\n\
          F  = A3301 - LH2 XFER LINE VALVE - CLOSED\n\
          G = A3309 - LH2 XFER LINE CHILLDOWN VALVE - CLOSED\n\
          H = A81065 - LH2 FILL LINE PURGE VALVE - CLOSED\n\
          I   = LH2 STORAGE TANK PRESS - FLAT\n\
          J  = ET LH2 ULL PRESS < 22\n\
          K = A75620 - XFER LINE PURGE SOV - CLOSED\n\
          L  = A75621 - XFER LINE PURGE VALVE - CLOSED\n\
          M = A75567 - 2K HE RMT SOL SOV - CLOSED\n\
          N = A75586 - 2K HE RMT SOV - CLOSED\n\
          O = PV11 - LH2 OUTBOARD F/D VALVE - OPEN\n\
          P = A100680 - LH2 TSM DRAIN VALVE - CLOSED\n\
          Q = A100681 - LH2 TSM DRAIN SOV - CLOSED\n\
          R  = PV12 - LH2 INBOARD F/D VALVE - CLOSED\n\
          S =  ET LH2 LLS - DRY";

MPS_PCL_PV11_OP_CMD::MPS_PCL_PV11_OP_CMD(const char*
seqName):asv_pcsVPCL_DS_ON_OFF_(seqName)
{
  userInit();
}

MPS_PCL_PV11_OP_CMD::~MPS_PCL_PV11_OP_CMD()
{
}


void MPS_PCL_PV11_OP_CMD::userInit(void)
{
  initiatingFD(V41K1391XL);
  initiatingFD(V41X1391NL);
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
int MPS_PCL_PV11_OP_CMD::userExecute(Discrete::ON_OFF dsVar)
{

 if (Discrete::On == (NMPS_PCLO_PV11_OP.getValue()))
{
 return PCL_ALLOW_CMD;
{

 if (Discrete::Off==dsVar)
{
  if (B() || (((C() && D() && E()) || (F() && G() && H()) || (I() && J() && H())) && (K() || L())&& (M() || N())))
  {
   return PCL_ALLOW_CMD;
  }
  else
  {
   NMPS_PCLN_PV11_OP.Set(Discrete::On);
   return PCL_BLOCK_CMD;
  }
}
if (Discrete::On==dsVar)
}
if (O() || P() || Q() || R() || S())
  {
   return PCL_ALLOW_CMD;
  }
else
  {
   NMPS_PCLN_PV11_OP.Set(Discrete::On);
   return PCL_BLOCK_CMD;
  }
}

}



// PV11 - LH2 Outboard F/D Valve - CLOSED

int MPS_PCL_PV11_OP_CMD::B()
{
  int  valReturnB = PCL_VOTER_FAIL;

try
{
            if (NMPSPV11INDSTATE.getValue() == MPS_VALVE_IND_STATE::CLOSED)
     {
              valReturnB = PCL_VOTER_PASS;
             }
else
            {
       valReturnB = PCL_VOTER_FAIL;
            }
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
}

catch(...)
{
    valReturnB = PCL_VOTER_FAIL;
}

 return valReturnB;
}




// A100677 - LH2 Main Fill valve - CLOSED

int MPS_PCL_PV11_OP_CMD::C()
{
int  valReturnC = PCL_VOTER_FAIL;

try
{
          if (NLH2_VLV_A100677.getValue() == LH2_VALVE_STATE::CLOSED)
    {
            valReturnC = PCL_VOTER_PASS;
           }
else
          {
      valReturnC = PCL_VOTER_FAIL;
          }
}

catch(...)
{
    valReturnC = PCL_VOTER_FAIL;
}

return valReturnC;
}




// A100678 - LH2 Aux Fill valve - CLOSED

int MPS_PCL_PV11_OP_CMD::D()
{
int  valReturnD = PCL_VOTER_FAIL;

try
{
          if (NLH2_VLV_A100678.getValue() == LH2_VALVE_STATE::CLOSED)
    {
            valReturnD = PCL_VOTER_PASS;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
            }
else
            {
       valReturnD = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnD = PCL_VOTER_FAIL;
}

return valReturnD;
}



// A100679 - LH2 Replenish Valve - CLOSED

int MPS_PCL_PV11_OP_CMD::E()
{
int  valReturnE = PCL_VOTER_FAIL;

try
{
            if (NLH2_VLV_A100679.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnE = PCL_VOTER_PASS;
             }
else
            {
       valReturnE = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnE = PCL_VOTER_FAIL;
}

return valReturnE;
}



// A3301 - LH2 Xfer Line Valve - CLOSED

int MPS_PCL_PV11_OP_CMD::F()
{
int  valReturnF = PCL_VOTER_FAIL;

try
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
{
            if (NLH2_VLV_A3301.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnF = PCL_VOTER_PASS;
             }
else
            {
       valReturnF = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnF = PCL_VOTER_FAIL;
}

return valReturnF;
}
```

```
// A3309 - LH2 Xfer Line Chilldown Valve - CLOSED

int MPS_PCL_PV11_OP_CMD::G()
{
int  valReturnG = PCL_VOTER_FAIL;

try
{
            if (NLH2_VLV_A3309.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnG = PCL_VOTER_PASS;
             }
else
            {
       valReturnG = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnG = PCL_VOTER_FAIL;
}

return valReturnG;
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
// A81065 - LH2 Fill Line Purge Valve - CLOSED

int MPS_PCL_PV11_OP_CMD::H()
{
int  valReturnH = PCL_VOTER_FAIL;

try
{
            if (NLH2_VLV_A81065.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnH = PCL_VOTER_PASS;
             }
else
            {
       valReturnH = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnH = PCL_VOTER_FAIL;
}

return valReturnH;
}



// LH2 STORAGE TANK PRESS - FLAT

int MPS_PCL_PV11_OP_CMD::I()
{
  int  valReturnI = PCL_VOTER_FAIL;

try
{
            if (NLH2_STOR_TNK_PRESS.getValue() == FLAT)
     {
              valReturnI = PCL_VOTER_PASS;
             }
else
            {
       valReturnI = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnI = PCL_VOTER_FAIL;
}

  return valReturnI;
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
// ET LH2 Ullage < 22 psia

int MPS_PCL_PV11_OP_CMD::J()
{
  int  valReturnJ = PCL_VOTER_FAIL;

try
{
            if (T41P1700C1.getvalue() < 22)
            {
               if (T41P1701C1.getvalue() < 22)
               {
                  if (T41P1701C1.getvalue() < 22)
                  {
                    valReturnJ = PCL_VOTER_PASS;
                    }
                  else
                    {
                  valReturnJ = PCL_VOTER_FAIL;
                    }
              {
             else
               {
              valReturnJ = PCL_VOTER_FAIL;
               }
          {
        else
           {
          valReturnJ = PCL_VOTER_FAIL;
          }
}

 catch(...)
    {
     valReturnJ = PCL_VOTER_FAIL;
    }

  return valReturnJ;
}




// A75620 - Xfer Line Purge SOV - CLOSED

int MPS_PCL_PV11_OP_CMD::K()
{
  int  valReturnK = PCL_VOTER_FAIL;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
try
{
            if (NLH2_VLV_A75620.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnK = PCL_VOTER_PASS;
             }
else
            {
       valReturnK = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnK = PCL_VOTER_FAIL;
}

return valReturnK;
}



// A75621 - Xfer Line Purge Valve - CLOSED

int MPS_PCL_PV11_OP_CMD::L()
{
  int  valReturnL = PCL_VOTER_FAIL;

try
{
            if (NLH2_VLV_A75621.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnL = PCL_VOTER_PASS;
             }
else
            {
       valReturnL = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnL = PCL_VOTER_FAIL;
}

  return valReturnL;
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
// A75567 - 2K He RMT SOL SOV - CLOSED

int MPS_PCL_PV11_OP_CMD::M()
{
  int  valReturnM = PCL_VOTER_FAIL;

 try
{
            if (NLH2_VLV_A75567.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnM = PCL_VOTER_PASS;
             }
else
            {
       valReturnM = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnM = PCL_VOTER_FAIL;
}

  return valReturnM;
}



// A75586 - 2K He RMT SOV - CLOSED

int MPS_PCL_PV11_OP_CMD::N()
{
  int  valReturnN = PCL_VOTER_FAIL;

try
{
            if (NLH2_VLV_A75586.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnN = PCL_VOTER_PASS;
             }
else
            {
       valReturnN = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnN = PCL_VOTER_FAIL;
}

  return valReturnN;
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
// PV11 - LH2 Outboard F/D Valve - OPEN

int MPS_PCL_PV11_OP_CMD::O()
{
  int  valReturnO = PCL_VOTER_FAIL;

try
{
            if (NMPSPV11INDSTATE.getValue() == MPS_VALVE_IND_STATE::OPEN)
     {
              valReturnO = PCL_VOTER_PASS;
             }
else
            {
       valReturnO = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnO = PCL_VOTER_FAIL;
}

  return valReturnO;
}




// A100680 - LH2 TSM Drain Valve - CLOSED

int MPS_PCL_PV11_OP_CMD::P()
{
int  valReturnP = PCL_VOTER_FAIL;

try
{
            if (NLH2_VLV_A100680.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnP = PCL_VOTER_PASS;
             }
else
            {
       valReturnP = PCL_VOTER_FAIL;
            }
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
catch(...)
{
    valReturnP = PCL_VOTER_FAIL;
}

return valReturnP;
}



// A100681 - LH2 TSM Drain SOV - CLOSED

int MPS_PCL_PV11_OP_CMD::Q()
{
int  valReturnQ = PCL_VOTER_FAIL;

try
{
            if (NLH2_VLV_A100681.getValue() == LH2_VALVE_STATE::CLOSED)
    {
              valReturnQ = PCL_VOTER_PASS;
             }
else
            {
       valReturnQ = PCL_VOTER_FAIL;
            }
}

catch(...)
{
    valReturnQ = PCL_VOTER_FAIL;
}

return valReturnQ;
}



// PV12 - LH2 Inboard F/D Valve - CLOSED

int MPS_PCL_PV11_OP_CMD::R()
{
  int  valReturnR = PCL_VOTER_FAIL;

try
{
            if (NMPSPV12INDSTATE.getValue() == LH2_VALVE_STATE::CLOSED)
    {
              valReturnR = PCL_VOTER_PASS;
             }
else
            {
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
       valReturnR = PCL_VOTER_FAIL;
            }
}

catch(...)
{
    valReturnR = PCL_VOTER_FAIL;
}

  return valReturnR;
}



// ET LH2 LLS - DRY

int MPS_PCL_PV11_OP_CMD::S()
{
  int  valReturnS = PCL_VOTER_FAIL;

try
{
            if (NMPSLH2LLSSTATE.getValue() == MPS_LLS_COMPOSITE_STATE::DRY)

           {
              valReturnS = PCL_VOTER_PASS;
             }

else
            {
       valReturnS = PCL_VOTER_FAIL;
            }
}

catch(...)
{
    valReturnS = PCL_VOTER_FAIL;
}

  return valReturnS;
}




MPS_PCL_PV11_OP_CMD mps_pcl_pv11_op_cmd(seqName);
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.4.16. MPS_PCL_PV12_CL_CMD

3.4.16.1. Description

This PCL provides protection for the MPS LH2 Inboard Fill and Drain Valve Close Commands.
This class inherits from the class CAS_Control_Logic. The Algorithm will be implemented directly in C++ Code.

Reference - LPS Program GCH03/04

3.4.16.1.1. Instance

Instance Name
MPS_PCL_PV12_CL_CMD

3.4.16.1.2. PCL Initiating Commands

Command
V41K1412XL

3.4.16.1.3. Operations Index

Operation	**Description**
determineState()	This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE INHIBITED ERROR
activate()	This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.
inhibit()	This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.
getState()	This method is inherited from the base class - CAS_Control_Logic
resetCLNotification()	This method is inherited from the base class - CAS_Control_Logic

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.4.16.2. Operations

3.4.16.2.1. getState()

3.4.16.2.1.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.16.2.2. resetCLNotification()

3.4.16.2.2.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.16.2.3. determineState()

3.4.16.2.3.1. Descriptive Text

This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE, INHIBITED or ERROR.

3.4.16.2.3.2. Sequence

```
if  (<V41K1412XL>.PCLSTAT = active)
  {
  CLState = ACTIVE;
  }
else
  {
   if  (<V41K1412XL>.PCLSTAT = inhibited)
     {
     CLState = INHIBITED;
      }
    else
      {
     CLState = ERROR;
      }
   }
```

3.4.16.2.4. activate()

3.4.16.2.4.1. Descriptive Text

This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.16.2.4.2. Sequence

```
A FD <V41K1412XL> PCL;

return;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.4.16.2.5. inhibit()

3.4.16.2.5.1. Descriptive Text

This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.16.2.5.2. Sequence

I FD <V41K1412XL> PCL;

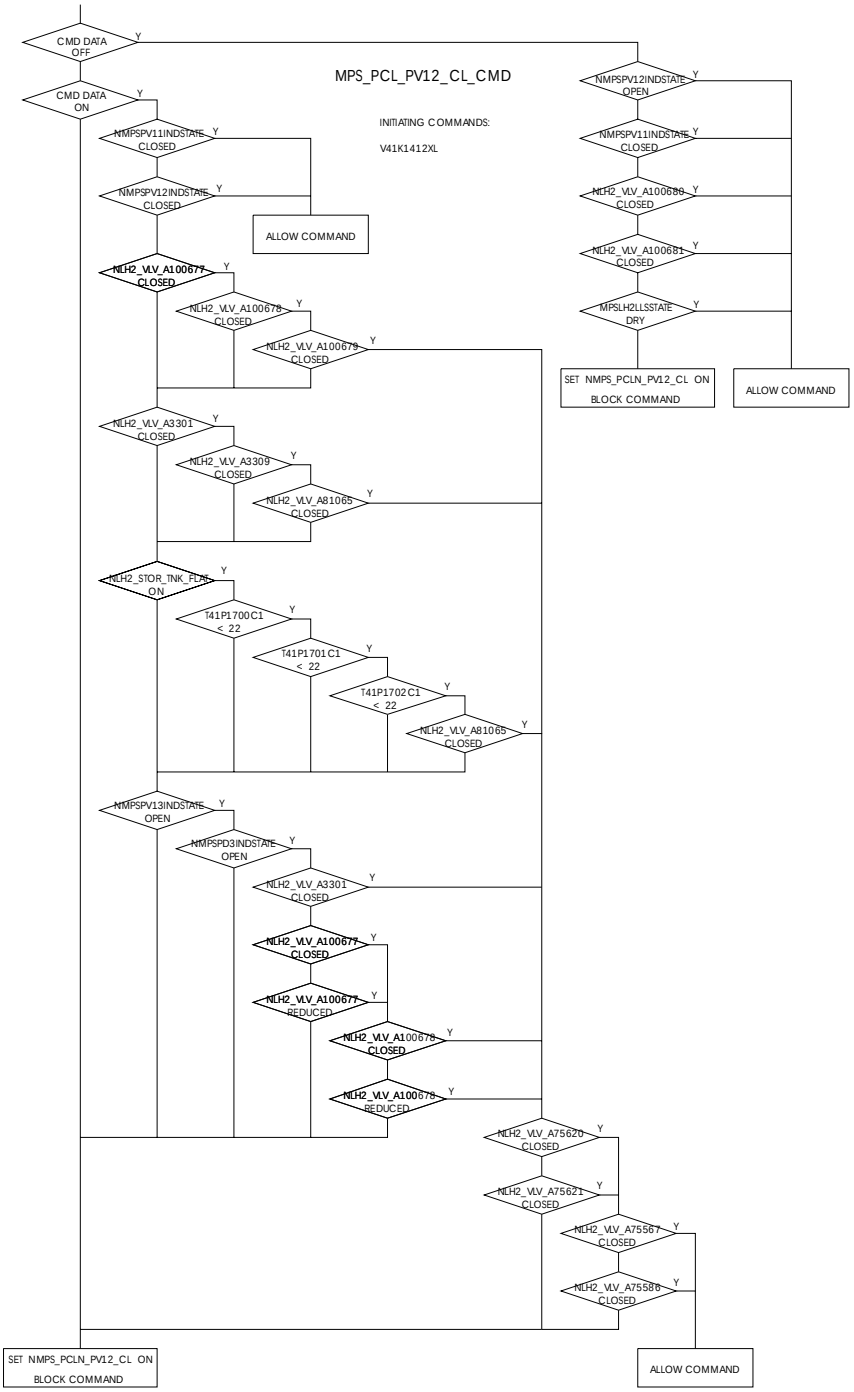
return;

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.4.16.3. Algorithm

3.4.16.3.1. PCL Algorithm

3.4.16.3.1.1. Flowchart



3.4.16.3.1.2. Sequence

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
#include "mps_pcl_pv12_cl_cmd.h"
#include "SME.h"
#include "LH2.h"
#include "MPS.h"
#include "LO2.h"

static const char* seqName = "MPS_PCL_PV12_CL_CMD";
static const char* algorithmDescription = "Revision Date - 01/20/2000\n\
if (A) ||\n\
if Command_Data is ON &&\n\
  if (B || C || (((D && E && F) || (G && H && I) || (J && K && I) || (L && G && M) || (L && N && O && M))
&& (P || Q)&& (R || S))) ||\n\
if Command_Data is OFF &&\n\
  if (T || U || V || W || X)\n\
where:   A  = NMPS_PCLO_PV12_CL - PCL Bypass ON\n\
         B  = PV11 - LH2 OUTBOARD F/D VALVE - CLOSED\n\
         C  = PV12 - LH2 INBOARD F/D VALVE - CLOSED\n\
         D  = A100677 - LH2 MAIN FILL VALVE - CLOSED\n\
         E  = A100678 - LH2 AUX FILL VALVE - CLOSED\n\
         F  = A100679 - LH2 REPLENISH VALVE - CLOSED\n\
         G  = A3301 - LH2 XFER LINE VALVE - CLOSED\n\
         H = A3309 - LH2 XFER LINE CHILLDOWN VALVE - CLOSED\n\
         I = A81065 - LH2 FILL LINE PURGE VALVE - CLOSED\n\
         J   = LH2 STORAGE TANK PRESS - FLAT\n\
         K  = ET LH2 ULL PRESS < 22\n\
         L  = PV13 - LH2 TOPPING VALVE - OPEN\n\
         M  = PD3 - LH2 RECIRC DISC VALVE - OPEN\n\
         N  = A100677 - LH2 MAIN FILL VALVE - CLOSED/REDUCED\n\
         O  = A100678 - LH2 AUX FILL VALVE - CLOSED/REDUCED\n\
         P = A75620 - XFER LINE PURGE SOV - CLOSED\n\
         Q  = A75621 - XFER LINE PURGE VALVE - CLOSED\n\
         R = A75567 - 2K HE RMT SOL SOV - CLOSED\n\
         S = A75586 - 2K HE RMT SOV - CLOSED\n\
         T = PV12 - LH2 INBOARD F/D VALVE - OPEN\n\
         U = A100680 - LH2 TSM DRAIN VALVE - CLOSED\n\
         V = A100681 - LH2 TSM DRAIN SOV - CLOSED\n\
         W  = PV11 - LH2 OUTBOARD F/D VALVE - CLOSED\n\
         X  =  ET LH2 LLS - DRY";

MPS_PCL_PV12_CL_CMD::MPS_PCL_PV12_CL_CMD(const char*
seqName):asv_pcsVPCL_DS_ON_OFF_(seqName)
{
  userInit();
}

MPS_PCL_PV12_CL_CMD::~MPS_PCL_PV12_CL_CMD()
{
}


void MPS_PCL_PV12_CL_CMD::userInit(void)
{
  initiatingFD(V41K1412XL);
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
int MPS_PCL_PV12_CL_CMD::userExecute(Discrete::ON_OFF dsVar)
{

 if (Discrete::On == (NMPS_PCLO_PV12_CL.getValue()))
{
 return PCL_ALLOW_CMD;
{

 if (Discrete::On==dsVar)
{
  if (B() || C() || (((D()&&E()&&F()) || (G()&&H()&&I()) || (J()&&K()&&I()) || (L()&&G()&&M()) ||
(L()&&N()&&O()&&M()))&&(K() || L())&&(M() || N())))
  {
   return PCL_ALLOW_CMD;
  }
  else
  {
   NMPS_PCLN_PV12_CL.Set(Discrete::On);
   return PCL_BLOCK_CMD;
  }
}
if (Discrete::Off==dsVar)
}
if (T() || U() || V() || W() || X())
  {
   return PCL_ALLOW_CMD;
  }
else
  {
   NMPS_PCLN_PV12_OP.Set(Discrete::On);
   return PCL_BLOCK_CMD;
  }
}

}



// PV11 - LH2 Outboard F/D Valve - CLOSED

int MPS_PCL_PV12_CL_CMD::B()
{
  int  valReturnB = PCL_VOTER_FAIL;

try
{
            if (NMPSPV11INDSTATE.getValue() == MPS_VALVE_IND_STATE::CLOSED)
     {
              valReturnB = PCL_VOTER_PASS;
             }
else
            {
       valReturnB = PCL_VOTER_FAIL;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
            }
}

catch(...)
{
    valReturnB = PCL_VOTER_FAIL;
}

 return valReturnB;
}



// PV12 - LH2 Inboard F/D Valve - CLOSED

int MPS_PCL_PV12_CL_CMD::C()
{
 int  valReturnC = PCL_VOTER_FAIL;

try
{
            if (NMPSPV12INDSTATE.getValue() == MPS_VALVE_IND_STATE::CLOSED)
    {
              valReturnC = PCL_VOTER_PASS;
             }
else
            {
      valReturnC = PCL_VOTER_FAIL;
            }
}

catch(...)
{
    valReturnC = PCL_VOTER_FAIL;
}

 return valReturnC;
}




// A100677 - LH2 Main Fill valve - CLOSED

int MPS_PCL_PV12_CL_CMD::D()
{
int  valReturnD = PCL_VOTER_FAIL;

try
{
            if (NLH2_VLV_A100677.getValue() == LH2_VALVE_STATE::CLOSED)
    {
              valReturnD = PCL_VOTER_PASS;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
            }
else
            {
       valReturnD = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnD = PCL_VOTER_FAIL;
}

return valReturnD;
}



// A100678 - LH2 Aux Fill valve - CLOSED

int MPS_PCL_PV12_CL_CMD::E()
{
int  valReturnE = PCL_VOTER_FAIL;

try
{
            if (NLH2_VLV_A100678.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnE = PCL_VOTER_PASS;
             }
else
            {
       valReturnE = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnE = PCL_VOTER_FAIL;
}

return valReturnE;
}



// A100679 - LH2 Replenish Valve - CLOSED

int MPS_PCL_PV12_CL_CMD::F()
{
int  valReturnF = PCL_VOTER_FAIL;

try
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
{
            if (NLH2_VLV_A100679.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnF = PCL_VOTER_PASS;
             }
else
            {
       valReturnF = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnF = PCL_VOTER_FAIL;
}

return valReturnF;
}



// A3301 - LH2 Xfer Line Valve - CLOSED

int MPS_PCL_PV12_CL_CMD::G()
{
int  valReturnG = PCL_VOTER_FAIL;

try
{
            if (NLH2_VLV_A3301.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnG = PCL_VOTER_PASS;
             }
else
            {
       valReturnG = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnG = PCL_VOTER_FAIL;
}

return valReturnG;
}



// A3309 - LH2 Xfer Line Chilldown Valve - CLOSED
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
int MPS_PCL_PV12_CL_CMD::H()
{
int  valReturnH = PCL_VOTER_FAIL;

try
{
            if (NLH2_VLV_A3309.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnH = PCL_VOTER_PASS;
             }
else
            {
       valReturnH = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnH = PCL_VOTER_FAIL;
}

return valReturnH;
}
```

```
// A81065 - LH2 Fill Line Purge Valve - CLOSED

int MPS_PCL_PV12_CL_CMD::I()
{
int  valReturnI = PCL_VOTER_FAIL;

try
{
            if (NLH2_VLV_A81065.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnI = PCL_VOTER_PASS;
             }
else
            {
       valReturnI = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnI = PCL_VOTER_FAIL;
}

return valReturnI;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
}



// LH2 STORAGE TANK PRESS - FLAT

int MPS_PCL_PV12_CL_CMD::J()
{
  int  valReturnJ = PCL_VOTER_FAIL;

try
{
            if (NLH2_STOR_TNK_PRESS.getValue() == FLAT)
     {
              valReturnJ = PCL_VOTER_PASS;
             }
else
            {
       valReturnJ = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnJ = PCL_VOTER_FAIL;
}

  return valReturnJ;
}



// ET LH2 Ullage < 22 psia

int MPS_PCL_PV12_CL_CMD::K()
{
  int  valReturnK = PCL_VOTER_FAIL;

try
{
            if (T41P1700C1.getvalue() < 22)
            {
               if (T41P1701C1.getvalue() < 22)
               {
                  if (T41P1701C1.getvalue() < 22)
                  {
                    valReturnK = PCL_VOTER_PASS;
                    }
                  else
                    {
                  valReturnK = PCL_VOTER_FAIL;
                    }
              {
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
            else
              {
             valReturnK = PCL_VOTER_FAIL;
              }
          {
        else
           {
          valReturnK = PCL_VOTER_FAIL;
          }
}

 catch(...)
    {
     valReturnK = PCL_VOTER_FAIL;
    }

  return valReturnK;
}



// PV13 - LH2 Topping Valve - OPEN

int MPS_PCL_PV12_CL_CMD::L()
{
  int  valReturnL = PCL_VOTER_FAIL;

try
{
            if (NMPSPV13INDSTATE.getValue() == MPS_VALVE_IND_STATE::OPEN)
     {
              valReturnL = PCL_VOTER_PASS;
             }
else
            {
       valReturnL = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnL = PCL_VOTER_FAIL;
}

  return valReturnL;
}



// PD3 - LH2 Recirc Disconnect Valve - OPEN

int MPS_PCL_PV12_CL_CMD::M()
{
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
  int  valReturnM = PCL_VOTER_FAIL;

try
{
            if (NMPSPD3INDSTATE.getValue() == MPS_VALVE_IND_STATE::OPEN)
     {
              valReturnM = PCL_VOTER_PASS;
             }
else
            {
       valReturnM = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnM = PCL_VOTER_FAIL;
}

  return valReturnM;
}




// A100677 - LH2 Main Fill valve - CLOSED/REDUCED

int MPS_PCL_PV12_CL_CMD::N()
{
int  valReturnN = PCL_VOTER_FAIL;

try
{
            if (NLH2_VLV_A100677.getValue() == LH2_VALVE_STATE::CLOSED)
              {
             valReturnN = PCL_VOTER_PASS;
             }
             else
                  {
                  if (NLH2_VLV_A100677.getValue() == LH2_VALVE_STATE::REDUCED)
                        {
                       valReturnN = PCL_VOTER_PASS;
                        }
                  else
                        {
                        valReturnN = PCL_VOTER_FAIL;
                       }
                   }

}

catch(...)
{
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
    valReturnN = PCL_VOTER_FAIL;
}

return valReturnN;
}



// A100678 - LH2 Aux Fill valve - CLOSED/REDUCED


int MPS_PCL_PV12_CL_CMD::O()
{
int  valReturnO = PCL_VOTER_FAIL;

try
{
            if (NLH2_VLV_A100678.getValue() == LH2_VALVE_STATE::CLOSED)
              {
              valReturnO = PCL_VOTER_PASS;
             }
             else
                  {
                  if (NLH2_VLV_A100678.getValue() == LH2_VALVE_STATE::REDUCED)
                        {
                       valReturnO = PCL_VOTER_PASS;
                        }
                  else
                        {
                        valReturnO = PCL_VOTER_FAIL;
                       }
                   }

}

catch(...)
{
    valReturnO = PCL_VOTER_FAIL;
}

return valReturnO;
}



// A75620 - Xfer Line Purge SOV - CLOSED

int MPS_PCL_PV12_CL_CMD::P()
{
 int  valReturnP = PCL_VOTER_FAIL;

try
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
{
            if (NLH2_VLV_A75620.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnP = PCL_VOTER_PASS;
             }
else
            {
       valReturnP = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnP = PCL_VOTER_FAIL;
}

return valReturnP;
}



// A75621 - Xfer Line Purge Valve - CLOSED

int MPS_PCL_PV12_CL_CMD::Q()
{
  int  valReturnQ = PCL_VOTER_FAIL;

try
{
            if (NLH2_VLV_A75621.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnQ = PCL_VOTER_PASS;
             }
else
            {
       valReturnQ = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnQ = PCL_VOTER_FAIL;
}

  return valReturnQ;
}




// A75567 - 2K He RMT SOL SOV - CLOSED
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
int MPS_PCL_PV12_CL_CMD::R()
{
  int  valReturnR = PCL_VOTER_FAIL;

 try
{
            if (NLH2_VLV_A75567.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnR = PCL_VOTER_PASS;
             }
else
            {
       valReturnR = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnR = PCL_VOTER_FAIL;
}

  return valReturnR;
}



// A75586 - 2K He RMT SOV - CLOSED

int MPS_PCL_PV12_CL_CMD::S()
{
  int  valReturnS = PCL_VOTER_FAIL;

try
{
            if (NLH2_VLV_A75586.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnS = PCL_VOTER_PASS;
             }
else
            {
       valReturnS = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnS = PCL_VOTER_FAIL;
}

  return valReturnS;
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
// PV12 - LH2 Inboard F/D Valve - OPEN

int MPS_PCL_PV12_CL_CMD::T()
{
  int  valReturnT = PCL_VOTER_FAIL;

try
{
            if (NMPSPV12INDSTATE.getValue() == MPS_VALVE_IND_STATE::OPEN)
     {
              valReturnT = PCL_VOTER_PASS;
             }
else
            {
       valReturnT = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnT = PCL_VOTER_FAIL;
}

  return valReturnT;
}




// A100680 - LH2 TSM Drain Valve - CLOSED

int MPS_PCL_PV12_CL_CMD::U()
{
int  valReturnU = PCL_VOTER_FAIL;

try
{
            if (NLH2_VLV_A100680.getValue() == MPS_VALVE_IND_STATE::CLOSED)
     {
              valReturnU = PCL_VOTER_PASS;
             }
else
            {
       valReturnU = PCL_VOTER_FAIL;
            }
}

catch(...)
{
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
    valReturnU = PCL_VOTER_FAIL;
}

return valReturnU;
}



// A100681 - LH2 TSM Drain SOV - CLOSED

int MPS_PCL_PV12_CL_CMD::V()
{
int  valReturnV = PCL_VOTER_FAIL;

try
{
          if (NLH2_VLV_A100681.getValue() == LH2_VALVE_STATE::CLOSED)
    {
            valReturnV = PCL_VOTER_PASS;
           }
else
          {
      valReturnV = PCL_VOTER_FAIL;
          }
}

catch(...)
{
    valReturnV = PCL_VOTER_FAIL;
}

return valReturnV;
}



// PV11 - LH2 Outboard F/D Valve - CLOSED

int MPS_PCL_PV12_CL_CMD::W()
{
 int  valReturnW = PCL_VOTER_FAIL;

try
{
          if (NMPSPV12INDSTATE.getValue() == MPS_VALVE_IND_STATE::CLOSED)
    {
            valReturnW = PCL_VOTER_PASS;
           }
else
          {
      valReturnW = PCL_VOTER_FAIL;
          }
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
}

catch(...)
{
     valReturnW = PCL_VOTER_FAIL;
}

  return valReturnW;
}



// ET LH2 LLS - DRY

int MPS_PCL_PV12_CL_CMD::X()
{
  int  valReturnX = PCL_VOTER_FAIL;

try
{
             if (NMPSLH2LLS.getValue() == MPS_LLS_COMPOSITE_STATE::DRY)

            {
               valReturnX = PCL_VOTER_PASS;
              }
else
              {
       valReturnX = PCL_VOTER_FAIL;
              }
}

catch(...)
{
     valReturnX = PCL_VOTER_FAIL;
}

  return valReturnX;
}




MPS_PCL_PV12_CL_CMD mps_pcl_pv12_cl_cmd(seqName);
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.4.17. MPS_PCL_PV12_OP_CMD

3.4.17.1. Description

This PCL provides protection for the MPS LH2 Inboard Fill and Drain Valve Open Commands.
This class inherits from the class CAS_Control_Logic. The Algorithm will be implemented directly in C++ Code.

Reference - LPS Program GCH03/04

3.4.17.1.1. Instance

Instance Name
MPS_PCL_PV12_OP_CMD

3.4.17.1.2. PCL Initiating Commands

Command
V41K1401XL
V41K1402XL
V41K1403NL

3.4.17.1.3. Operations Index

Operation	Description
determineState()	This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE INHIBITED ERROR
activate()	This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.
inhibit()	This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.
getState()	This method is inherited from the base class - CAS_Control_Logic
resetCLNotification()	This method is inherited from the base class - CAS_Control_Logic

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.4.17.2. Operations

3.4.17.2.1. getState()

3.4.17.2.1.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.17.2.2. resetCLNotification()

3.4.17.2.2.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.17.2.3. determineState()

3.4.17.2.3.1. Descriptive Text

This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE, INHIBITED or ERROR.

3.4.17.2.3.2. Sequence

```
if  (<V41K1401XL>.PCLSTAT = active  AND
     <V41K1402XL>.PCLSTAT = active  AND
     <V41K1403NL>.PCLSTAT = active)
  {
  CLState = ACTIVE;
  }
else
  {
   if  (<V41K1401XL>.PCLSTAT = inhibited AND
        <V41K1402XL>.PCLSTAT = inhibited  AND
        <V41K1403NL>.PCLSTAT = inhibited)
     {
     CLState = INHIBITED;
      }
    else
      {
     CLState = ERROR;
      }
   }
```

3.4.17.2.4. activate()

3.4.17.2.4.1. Descriptive Text

This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.17.2.4.2. Sequence

```
A FD <V41K1401XL> PCL;
A FD <V41K1402XL> PCL;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

A FD <V41K1403NL> PCL;

return;

3.4.17.2.5. inhibit()

3.4.17.2.5.1. Descriptive Text

This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.17.2.5.2. Sequence

I FD <V41K1401XL> PCL;
I FD <V41K1402XL> PCL;
I FD <V41K1403NL> PCL;

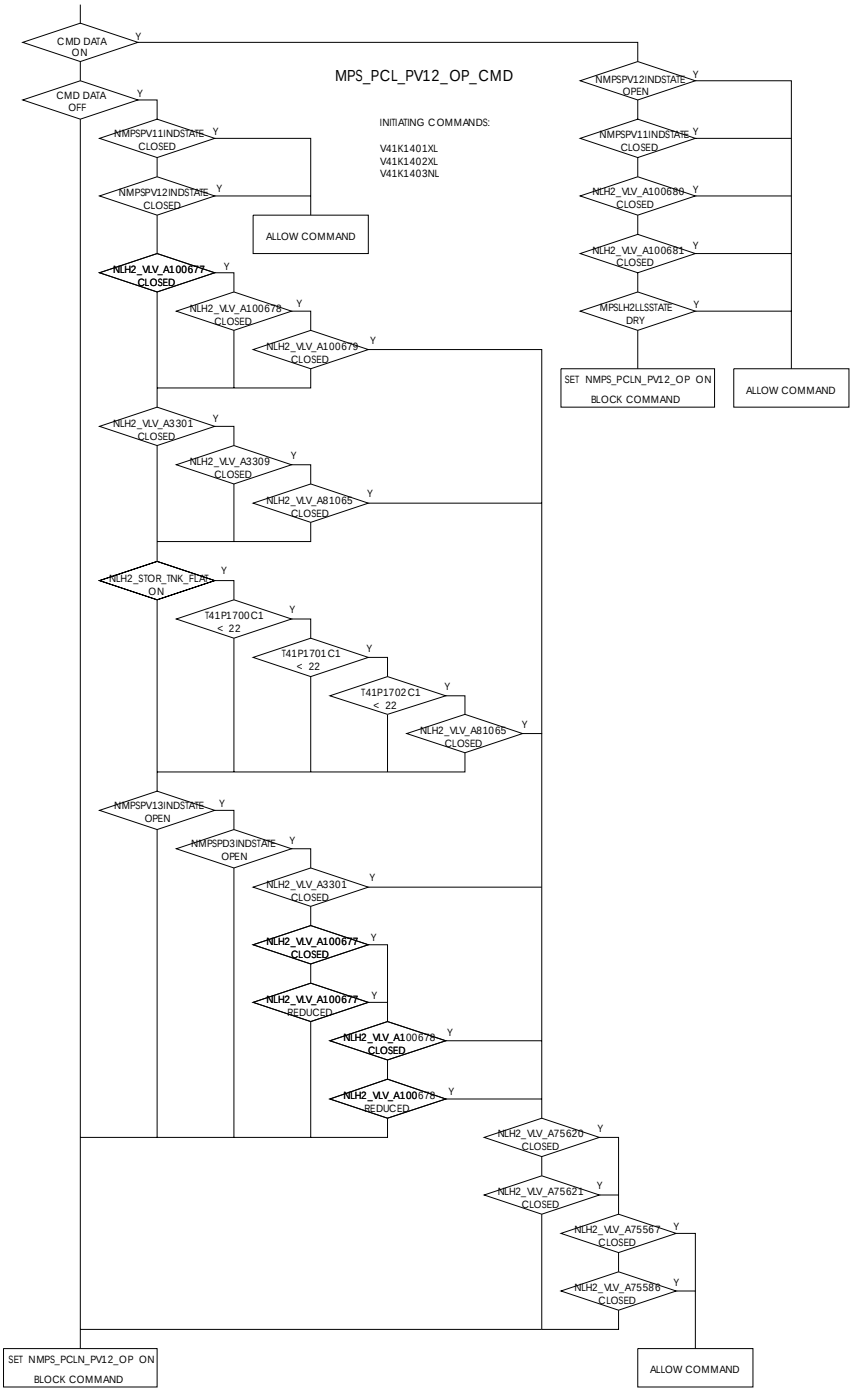
return;

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.4.17.3. Algorithm

3.4.17.3.1. PCL Algorithm

3.4.17.3.1.1. Flowchart



3.4.17.3.1.2. Sequence

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
#include "mps_pcl_pv12_op_cmd.h"
#include "SME.h"
#include "LH2.h"
#include "MPS.h"
#include "LO2.h"

static const char* seqName = "MPS_PCL_PV12_OP_CMD";
static const char* algorithmDescription = "Revision Date - 01/20/2000\n\
if (A) ||\n\
if Command_Data is OFF &&\n\
  if (B || C || (((D && E && F) || (G && H && I) || (J && K && I) || (L && G && M) || (L && N && O && M))
&& (P || Q)&& (R || S))) ||\n\
if Command_Data is ON &&\n\
  if (T || U || V || W || X)\n\
where:   A  = NMPS_PCLO_PV12_OP - PCL Bypass ON\n\
         B  = PV11 - LH2 OUTBOARD F/D VALVE - CLOSED\n\
         C  = PV12 - LH2 INBOARD F/D VALVE - CLOSED\n\
         D  = A100677 - LH2 MAIN FILL VALVE - CLOSED\n\
         E  = A100678 - LH2 AUX FILL VALVE - CLOSED\n\
         F  = A100679 - LH2 REPLENISH VALVE - CLOSED\n\
         G  = A3301 - LH2 XFER LINE VALVE - CLOSED\n\
         H = A3309 - LH2 XFER LINE CHILLDOWN VALVE - CLOSED\n\
         I = A81065 - LH2 FILL LINE PURGE VALVE - CLOSED\n\
         J   = LH2 STORAGE TANK PRESS - FLAT\n\
         K  = ET LH2 ULL PRESS < 22\n\
         L  = PV13 - LH2 TOPPING VALVE - OPEN\n\
         M  = PD3 - LH2 RECIRC DISC VALVE - OPEN\n\
         N  = A100677 - LH2 MAIN FILL VALVE - CLOSED/REDUCED\n\
         O  = A100678 - LH2 AUX FILL VALVE - CLOSED/REDUCED\n\
         P = A75620 - XFER LINE PURGE SOV - CLOSED\n\
         Q  = A75621 - XFER LINE PURGE VALVE - CLOSED\n\
         R = A75567 - 2K HE RMT SOL SOV - CLOSED\n\
         S = A75586 - 2K HE RMT SOV - CLOSED\n\
         T = PV12 - LH2 INBOARD F/D VALVE - OPEN\n\
         U = A100680 - LH2 TSM DRAIN VALVE - CLOSED\n\
         V = A100681 - LH2 TSM DRAIN SOV - CLOSED\n\
         W  = PV11 - LH2 OUTBOARD F/D VALVE - CLOSED\n\
         X  =  ET LH2 LLS - DRY";

MPS_PCL_PV12_OP_CMD::MPS_PCL_PV12_OP_CMD(const char*
seqName):asv_pcsVPCL_DS_ON_OFF_(seqName)
{
  userInit();
}

MPS_PCL_PV12_OP_CMD::~MPS_PCL_PV12_OP_CMD()
{
}


void MPS_PCL_PV12_OP_CMD::userInit(void)
{
  initiatingFD(V41K1401XL);
  initiatingFD(V41K1402XL);
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
  initiatingFD(V41K1403NL);
}

int MPS_PCL_PV12_OP_CMD::userExecute(Discrete::ON_OFF dsVar)
{

 if (Discrete::On == (NMPS_PCLO_PV12_OP.getValue()))
{
 return PCL_ALLOW_CMD;
{

 if (Discrete::Off==dsVar)
{
  if (B() || C() || (((D()&&E()&&F()) || (G()&&H()&&I()) || (J()&&K()&&I()) || (L()&&G()&&M()) ||
(L()&&N()&&O()&&M()))&&(K() || L())&&(M() || N())))
  {
   return PCL_ALLOW_CMD;
  }
  else
  {
   NMPS_PCLN_PV12_OP.Set(Discrete::On);
   return PCL_BLOCK_CMD;
  }
}
if (Discrete::On==dsVar)
}
if (T() || U() || V() || W() || X())
  {
   return PCL_ALLOW_CMD;
  }
else
  {
   NMPS_PCLN_PV12_OP.Set(Discrete::On);
   return PCL_BLOCK_CMD;
  }
}

}



// PV11 - LH2 Outboard F/D Valve - CLOSED

int MPS_PCL_PV12_OP_CMD::B()
{
  int  valReturnB = PCL_VOTER_FAIL;

try
{
            if (NMPSPV11INDSTATE.getValue() == MPS_VALVE_IND_STATE::CLOSED)
     {
              valReturnB = PCL_VOTER_PASS;
             }
else
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
            {
       valReturnB = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnB = PCL_VOTER_FAIL;
}

  return valReturnB;
}



// PV12 - LH2 Inboard F/D Valve - CLOSED

int MPS_PCL_PV12_OP_CMD::C()
{
  int  valReturnC = PCL_VOTER_FAIL;

try
{
            if (NMPSPV12INDSTATE.getValue() == MPS_VALVE_IND_STATE::CLOSED)
     {
              valReturnC = PCL_VOTER_PASS;
             }
else
            {
       valReturnC = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnC = PCL_VOTER_FAIL;
}

  return valReturnC;
}




// A100677 - LH2 Main Fill valve - CLOSED

int MPS_PCL_PV12_OP_CMD::D()
{
int  valReturnD = PCL_VOTER_FAIL;

try
{
            if (NLH2_VLV_A100677.getValue() == LH2_VALVE_STATE::CLOSED)
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
    {
            valReturnD = PCL_VOTER_PASS;
           }
else
           {
      valReturnD = PCL_VOTER_FAIL;
           }
}

catch(...)
{
    valReturnD = PCL_VOTER_FAIL;
}

return valReturnD;
}



// A100678 - LH2 Aux Fill valve - CLOSED

int MPS_PCL_PV12_OP_CMD::E()
{
int  valReturnE = PCL_VOTER_FAIL;

try
{
           if (NLH2_VLV_A100678.getValue() == LH2_VALVE_STATE::CLOSED)
    {
            valReturnE = PCL_VOTER_PASS;
           }
else
           {
      valReturnE = PCL_VOTER_FAIL;
           }
}

catch(...)
{
    valReturnE = PCL_VOTER_FAIL;
}

return valReturnE;
}



// A100679 - LH2 Replenish Valve - CLOSED

int MPS_PCL_PV12_OP_CMD::F()
{
int  valReturnF = PCL_VOTER_FAIL;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
try
{
            if (NLH2_VLV_A100679.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnF = PCL_VOTER_PASS;
             }
else
            {
       valReturnF = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnF = PCL_VOTER_FAIL;
}

return valReturnF;
}



// A3301 - LH2 Xfer Line Valve - CLOSED

int MPS_PCL_PV12_OP_CMD::G()
{
int  valReturnG = PCL_VOTER_FAIL;

try
{
            if (NLH2_VLV_A3301.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnG = PCL_VOTER_PASS;
             }
else
            {
       valReturnG = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnG = PCL_VOTER_FAIL;
}

return valReturnG;
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
// A3309 - LH2 Xfer Line Chilldown Valve - CLOSED

int MPS_PCL_PV12_OP_CMD::H()
{
int  valReturnH = PCL_VOTER_FAIL;

try
{
            if (NLH2_VLV_A3309.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnH = PCL_VOTER_PASS;
             }
else
            {
       valReturnH = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnH = PCL_VOTER_FAIL;
}

return valReturnH;
}
```

```
// A81065 - LH2 Fill Line Purge Valve - CLOSED

int MPS_PCL_PV12_OP_CMD::I()
{
int  valReturnI = PCL_VOTER_FAIL;

try
{
            if (NLH2_VLV_A81065.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnI = PCL_VOTER_PASS;
             }
else
            {
       valReturnI = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnI = PCL_VOTER_FAIL;
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
return valReturnI;
}



// LH2 STORAGE TANK PRESS - FLAT

int MPS_PCL_PV12_OP_CMD::J()
{
  int  valReturnJ = PCL_VOTER_FAIL;

try
{
            if (NLH2_STOR_TNK_PRESS.getValue() == FLAT)
     {
              valReturnJ = PCL_VOTER_PASS;
             }
else
            {
       valReturnJ = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnJ = PCL_VOTER_FAIL;
}

  return valReturnJ;
}



// ET LH2 Ullage < 22 psia

int MPS_PCL_PV12_OP_CMD::K()
{
  int  valReturnK = PCL_VOTER_FAIL;

try
{
            if (T41P1700C1.getvalue() < 22)
            {
               if (T41P1701C1.getvalue() < 22)
               {
                  if (T41P1701C1.getvalue() < 22)
                  {
                    valReturnK = PCL_VOTER_PASS;
                    }
                  else
                    {
                  valReturnK = PCL_VOTER_FAIL;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
                    }
                {
               else
                 {
                valReturnK = PCL_VOTER_FAIL;
                 }
            {
          else
             {
            valReturnK = PCL_VOTER_FAIL;
            }
}

 catch(...)
    {
     valReturnK = PCL_VOTER_FAIL;
    }

  return valReturnK;
}



// PV13 - LH2 Topping Valve - OPEN

int MPS_PCL_PV12_OP_CMD::L()
{
  int  valReturnL = PCL_VOTER_FAIL;

try
{
            if (NMPSPV13INDSTATE.getValue() == MPS_VALVE_IND_STATE::OPEN)
     {
              valReturnL = PCL_VOTER_PASS;
             }
else
            {
       valReturnL = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnL = PCL_VOTER_FAIL;
}

  return valReturnL;
}



// PD3 - LH2 Recirc Disconnect Valve - OPEN
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
int MPS_PCL_PV12_OP_CMD::M()
{
  int  valReturnM = PCL_VOTER_FAIL;

try
{
            if (NMPSPD3INDSTATE.getValue() == MPS_VALVE_IND_STATE::OPEN)
     {
              valReturnM = PCL_VOTER_PASS;
             }
else
            {
       valReturnM = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnM = PCL_VOTER_FAIL;
}

  return valReturnM;
}




// A100677 - LH2 Main Fill valve - CLOSED/REDUCED

int MPS_PCL_PV12_OP_CMD::N()
{
int  valReturnN = PCL_VOTER_FAIL;

try
{
            if (NLH2_VLV_A100677.getValue() == LH2_VALVE_STATE::CLOSED)
              {
              valReturnN = PCL_VOTER_PASS;
             }
              else
                   {
                   if (NLH2_VLV_A100677.getValue() == LH2_VALVE_STATE::REDUCED)
                         {
                        valReturnN = PCL_VOTER_PASS;
                         }
                   else
                         {
                         valReturnN = PCL_VOTER_FAIL;
                        }
                    }

}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
catch(...)
{
    valReturnN = PCL_VOTER_FAIL;
}

return valReturnN;
}



// A100678 - LH2 Aux Fill valve - CLOSED/REDUCED


int MPS_PCL_PV12_OP_CMD::O()
{
int  valReturnO = PCL_VOTER_FAIL;

try
{
            if (NLH2_VLV_A100678.getValue() == LH2_VALVE_STATE::CLOSED)
              {
             valReturnO = PCL_VOTER_PASS;
            }
             else
                  {
                  if (NLH2_VLV_A100678.getValue() == LH2_VALVE_STATE::REDUCED)
                        {
                       valReturnO = PCL_VOTER_PASS;
                        }
                  else
                        {
                        valReturnO = PCL_VOTER_FAIL;
                       }
                   }

}

catch(...)
{
    valReturnO = PCL_VOTER_FAIL;
}

return valReturnO;
}



// A75620 - Xfer Line Purge SOV - CLOSED

int MPS_PCL_PV12_OP_CMD::P()
{
  int  valReturnP = PCL_VOTER_FAIL;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
try
{
            if (NLH2_VLV_A75620.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnP = PCL_VOTER_PASS;
             }
else
            {
       valReturnP = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnP = PCL_VOTER_FAIL;
}

return valReturnP;
}



// A75621 - Xfer Line Purge Valve - CLOSED

int MPS_PCL_PV12_OP_CMD::Q()
{
  int  valReturnQ = PCL_VOTER_FAIL;

try
{
            if (NLH2_VLV_A75621.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnQ = PCL_VOTER_PASS;
             }
else
            {
       valReturnQ = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnQ = PCL_VOTER_FAIL;
}

  return valReturnQ;
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
// A75567 - 2K He RMT SOL SOV - CLOSED

int MPS_PCL_PV12_OP_CMD::R()
{
  int  valReturnR = PCL_VOTER_FAIL;

 try
{
            if (NLH2_VLV_A75567.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnR = PCL_VOTER_PASS;
             }
else
            {
       valReturnR = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnR = PCL_VOTER_FAIL;
}

  return valReturnR;
}



// A75586 - 2K He RMT SOV - CLOSED

int MPS_PCL_PV12_OP_CMD::S()
{
  int  valReturnS = PCL_VOTER_FAIL;

try
{
            if (NLH2_VLV_A75586.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnS = PCL_VOTER_PASS;
             }
else
            {
       valReturnS = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnS = PCL_VOTER_FAIL;
}

  return valReturnS;
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
// PV12 - LH2 Inboard F/D Valve - OPEN

int MPS_PCL_PV12_OP_CMD::T()
{
  int  valReturnT = PCL_VOTER_FAIL;

try
{
            if (NMPSPV12INDSTATE.getValue() == MPS_VALVE_IND_STATE::OPEN)
     {
              valReturnT = PCL_VOTER_PASS;
             }
else
            {
       valReturnT = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnT = PCL_VOTER_FAIL;
}

  return valReturnT;
}




// A100680 - LH2 TSM Drain Valve - CLOSED

int MPS_PCL_PV12_OP_CMD::U()
{
int  valReturnU = PCL_VOTER_FAIL;

try
{
            if (NLH2_VLV_A100680.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnU = PCL_VOTER_PASS;
             }
else
            {
       valReturnU = PCL_VOTER_FAIL;
            }
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
catch(...)
{
    valReturnU = PCL_VOTER_FAIL;
}

return valReturnU;
}



// A100681 - LH2 TSM Drain SOV - CLOSED

int MPS_PCL_PV12_OP_CMD::V()
{
int  valReturnV = PCL_VOTER_FAIL;

try
{
            if (NLH2_VLV_A100681.getValue() == LH2_VALVE_STATE::CLOSED)
    {
              valReturnV = PCL_VOTER_PASS;
             }
else
            {
       valReturnV = PCL_VOTER_FAIL;
            }
}

catch(...)
{
    valReturnV = PCL_VOTER_FAIL;
}

return valReturnV;
}



// PV11 - LH2 Outboard F/D Valve - CLOSED

int MPS_PCL_PV12_OP_CMD::W()
{
  int  valReturnW = PCL_VOTER_FAIL;

try
{
            if (NMPSPV12INDSTATE.getValue() == MPS_VALVE_IND_STATE::CLOSED)
    {
              valReturnW = PCL_VOTER_PASS;
             }
else
            {
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
       valReturnW = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnW = PCL_VOTER_FAIL;
}

  return valReturnW;
}



// ET LH2 LLS - DRY

int MPS_PCL_PV12_OP_CMD::X()
{
  int  valReturnX = PCL_VOTER_FAIL;

try
{
             if (NMPSLH2LLS.getValue() == MPS_LL2_COMPOSITE_STATE::DRY)

            {
               valReturnX = PCL_VOTER_PASS;
              }
else
             {
       valReturnX = PCL_VOTER_FAIL;
             }
}

catch(...)
{
     valReturnX = PCL_VOTER_FAIL;
}

  return valReturnX;
}




MPS_PCL_PV12_OP_CMD mps_pcl_pv12_op_cmd(seqName);
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.4.18. MPS_PCL_PV13_OP_CMD

3.4.18.1. Description

This PCL provides protection for the MPS LH2 Topping Valve Open Commands.
This class inherits from the class CAS_Control_Logic. The Algorithm will be implemented directly in C++ Code.

Reference - LPS Program GCH05

3.4.18.1.1. Instance

Instance Name
MPS_PCL_PV13_OP_CMD

3.4.18.1.2. PCL Initiating Commands

Command
V41K1411XL

3.4.18.1.3. Operations Index

Operation	**Description**
determineState()	This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE INHIBITED ERROR
activate()	This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.
inhibit()	This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.
getState()	This method is inherited from the base class - CAS_Control_Logic
resetCLNotification()	This method is inherited from the base class - CAS_Control_Logic

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.4.18.2. Operations

3.4.18.2.1. getState()

3.4.18.2.1.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.18.2.2. resetCLNotification()

3.4.18.2.2.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.18.2.3. determineState()

3.4.18.2.3.1. Descriptive Text

This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE, INHIBITED or ERROR.

3.4.18.2.3.2. Sequence

```
if  (<V41K1411XL>.PCLSTAT = active)
  {
  CLState = ACTIVE;
  }
else
  {
   if  (<V41K1411XL>.PCLSTAT = inhibited)
     {
     CLState = INHIBITED;
      }
    else
      {
     CLState = ERROR;
      }
   }
```

3.4.18.2.4. activate()

3.4.18.2.4.1. Descriptive Text

This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.18.2.4.2. Sequence

```
A FD <V41K1411XL> PCL;

return;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.4.18.2.5. inhibit()

3.4.18.2.5.1. Descriptive Text

This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.18.2.5.2. Sequence

I FD <V41K1411XL> PCL;

return;

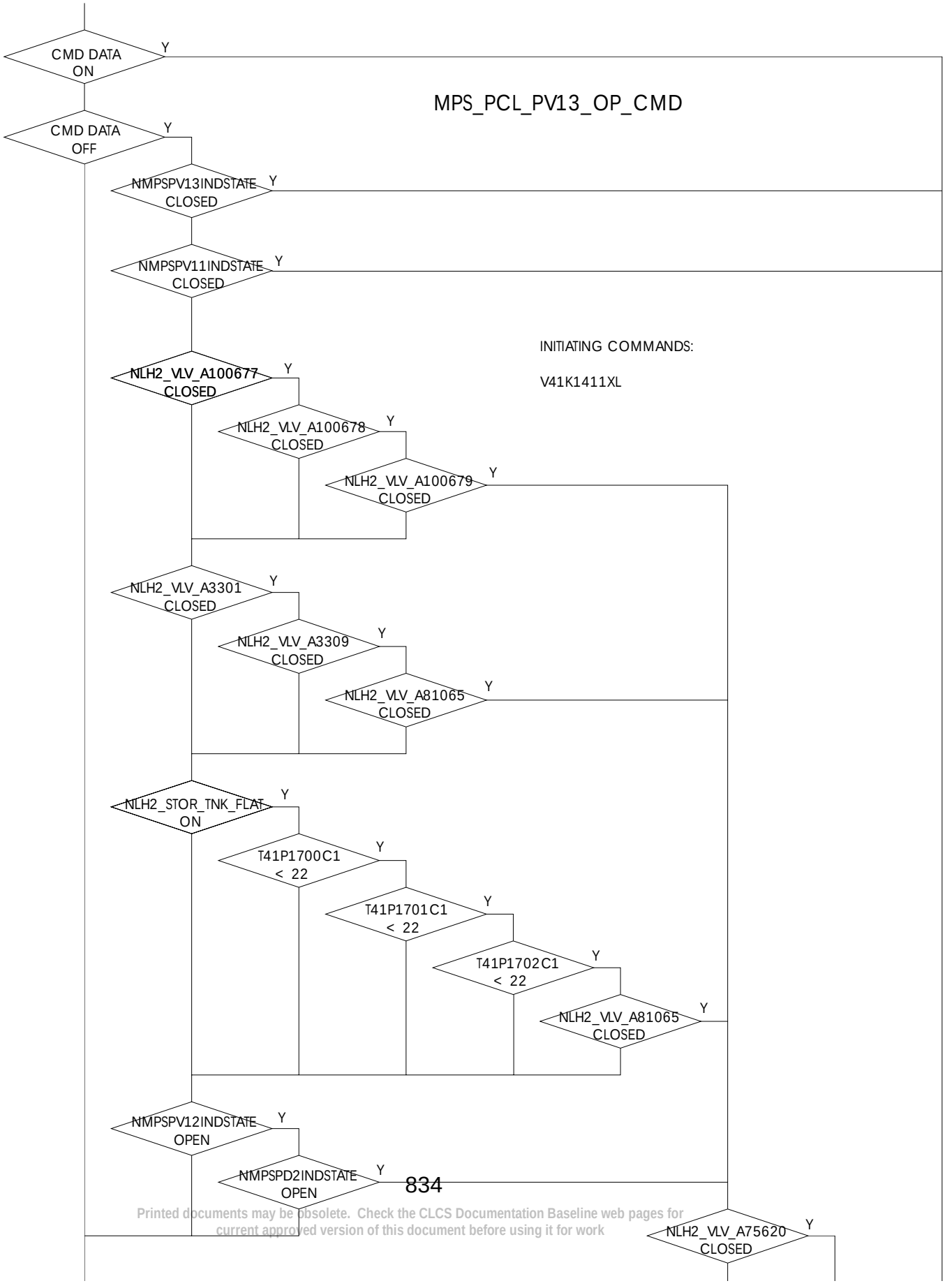
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.4.18.3. Algorithm

3.4.18.3.1. PCL Algorithm

3.4.18.3.1.1. Flowchart

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.4.18.3.1.2. Sequence

```
#include "mps_pcl_pv13_op_cmd.h"
#include "SME.h"
#include "LH2.h"
#include "MPS.h"
#include "LO2.h"

static const char* seqName = "MPS_PCL_PV13_OP_CMD";
static const char* algorithmDescription = "Revision Date - 01/20/2000\n\
if (A) ||\n\
if Command_Data is ON ||\n\
if Command_Data is OFF &&\n\
  if (B || C || (((D && E && F) || (G && H && I) || (J && K && I) || (L && M)) && (N || O)&& (P || Q)))\n\
where:   A  = NMPS_PCLO_PV13_OP - PCL Bypass ON\n\
         B  = PV13 - LH2 TOPPING VALVE - CLOSED\n\
         C  = PV11 - LH2 OUTBOARD F/D VALVE - CLOSED\n\
         D  = A100677 - LH2 MAIN FILL VALVE - CLOSED\n\
         E  = A100678 - LH2 AUX FILL VALVE - CLOSED\n\
         F  = A100679 - LH2 REPLENISH VALVE - CLOSED\n\
         G  = A3301 - LH2 XFER LINE VALVE - CLOSED\n\
         H = A3309 - LH2 XFER LINE CHILLDOWN VALVE - CLOSED\n\
         I = A81065 - LH2 FILL LINE PURGE VALVE - CLOSED\n\
         J   = LH2 STORAGE TANK PRESS - FLAT\n\
         K  = ET LH2 ULL PRESS < 22\n\
         L  = PV12 - LH2 INBOARD F/D VALVE - OPEN\n\
         M  = PD2 - LH2 ORB/ET FEED DISC VALVE - OPEN\n\
         N = A75620 - XFER LINE PURGE SOV - CLOSED\n\
         O  = A75621 - XFER LINE PURGE VALVE - CLOSED\n\
         P = A75567 - 2K HE RMT SOL SOV - CLOSED\n\
         Q = A75586 - 2K HE RMT SOV - CLOSED";

MPS_PCL_PV13_OP_CMD::MPS_PCL_PV13_OP_CMD(const char*
seqName):asv_pcsVPCL_DS_ON_OFF_(seqName)
{
  userInit();
}

MPS_PCL_PV13_OP_CMD::~MPS_PCL_PV13_OP_CMD()
{
}


void MPS_PCL_PV13_OP_CMD::userInit(void)
{
  initiatingFD(V41K1411XL);
}

int MPS_PCL_PV13_OP_CMD::userExecute(Discrete::ON_OFF dsVar)
{
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
 if (Discrete::On == (NMPS_PCLO_PV13_OP.getValue()))
{
 return PCL_ALLOW_CMD;
}

 if (Discrete::Off==dsVar)
{
  if (B() || C() || (((D()&&E()&&F()) || (G()&&H()&&I()) || (J()&&K()&&I()) || (L()&&M())) && (N() ||
O())&&(P() || Q())))
  {
   return PCL_ALLOW_CMD;
  }
  else
  {
   NMPS_PCLN_PV13_OP.Set(Discrete::On);
   return PCL_BLOCK_CMD;
  }
}

if (Discrete::On==dsVar)
 {
   return PCL_ALLOW_CMD;
 }
 else
  {
   NMPS_PCLN_PV13_OP.Set(Discrete::On);
   return PCL_BLOCK_CMD;
 }

}



// PV13 - LH2 Topping Valve - CLOSED

int MPS_PCL_PV13_OP_CMD::B()
{
  int  valReturnB = PCL_VOTER_FAIL;

try
{
            if (NMPSPV13INDSTATE.getValue() == MPS_VALVE_IND_STATE::CLOSED)
     {
              valReturnB = PCL_VOTER_PASS;
             }
else
            {
       valReturnB = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnB = PCL_VOTER_FAIL;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
}

  return valReturnB;
}



// PV11 - LH2 Outboard F/D Valve - CLOSED

int MPS_PCL_PV13_OP_CMD::C()
{
  int  valReturnC = PCL_VOTER_FAIL;

try
{
            if (NMPSPV11INDSTATE.getValue() == MPS_VALVE_IND_STATE::CLOSED)
     {
              valReturnC = PCL_VOTER_PASS;
             }
else
            {
       valReturnC = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnC = PCL_VOTER_FAIL;
}

  return valReturnC;
}




// A100677 - LH2 Main Fill valve - CLOSED

int MPS_PCL_PV13_OP_CMD::D()
{
int  valReturnD = PCL_VOTER_FAIL;

try
{
            if (NLH2_VLV_A100677.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnD = PCL_VOTER_PASS;
             }
else
            {
       valReturnD = PCL_VOTER_FAIL;
            }
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
catch(...)
{
    valReturnD = PCL_VOTER_FAIL;
}

return valReturnD;
}



// A100678 - LH2 Aux Fill valve - CLOSED

int MPS_PCL_PV13_OP_CMD::E()
{
int  valReturnE = PCL_VOTER_FAIL;

try
{
          if (NLH2_VLV_A100678.getValue() == LH2_VALVE_STATE::CLOSED)
    {
            valReturnE = PCL_VOTER_PASS;
           }
else
          {
      valReturnE = PCL_VOTER_FAIL;
          }
}

catch(...)
{
    valReturnE = PCL_VOTER_FAIL;
}

return valReturnE;
}



// A100679 - LH2 Replenish Valve - CLOSED

int MPS_PCL_PV13_OP_CMD::F()
{
int  valReturnF = PCL_VOTER_FAIL;

try
{
          if (NLH2_VLV_A100679.getValue() == LH2_VALVE_STATE::CLOSED)
    {
            valReturnF = PCL_VOTER_PASS;
           }
else
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
            {
       valReturnF = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnF = PCL_VOTER_FAIL;
}

return valReturnF;
}



// A3301 - LH2 Xfer Line Valve - CLOSED

int MPS_PCL_PV13_OP_CMD::G()
{
int  valReturnG = PCL_VOTER_FAIL;

try
{
            if (NLH2_VLV_A3301.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnG = PCL_VOTER_PASS;
             }
else
            {
       valReturnG = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnG = PCL_VOTER_FAIL;
}

return valReturnG;
}




// A3309 - LH2 Xfer Line Chilldown Valve - CLOSED

int MPS_PCL_PV13_OP_CMD::H()
{
int  valReturnH = PCL_VOTER_FAIL;

try
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
{
            if (NLH2_VLV_A3309.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnH = PCL_VOTER_PASS;
             }
else
            {
       valReturnH = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnH = PCL_VOTER_FAIL;
}

return valReturnH;
}




// A81065 - LH2 Fill Line Purge Valve - CLOSED

int MPS_PCL_PV13_OP_CMD::I()
{
int  valReturnI = PCL_VOTER_FAIL;

try
{
            if (NLH2_VLV_A81065.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnI = PCL_VOTER_PASS;
             }
else
            {
       valReturnI = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnI = PCL_VOTER_FAIL;
}

return valReturnI;
}



// LH2 STORAGE TANK PRESS - FLAT
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
int MPS_PCL_PV13_OP_CMD::J()
{
  int  valReturnJ = PCL_VOTER_FAIL;

try
{
            if (NLH2_STOR_TNK_PRESS.getValue() == FLAT)
     {
              valReturnJ = PCL_VOTER_PASS;
             }
else
            {
       valReturnJ = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnJ = PCL_VOTER_FAIL;
}

  return valReturnJ;
}



// ET LH2 Ullage < 22 psia

int MPS_PCL_PV13_OP_CMD::K()
{
  int  valReturnK = PCL_VOTER_FAIL;

try
{
            if (T41P1700C1.getvalue() < 22)
            {
               if (T41P1701C1.getvalue() < 22)
               {
                  if (T41P1701C1.getvalue() < 22)
                  {
                    valReturnK = PCL_VOTER_PASS;
                    }
                  else
                    {
                  valReturnK = PCL_VOTER_FAIL;
                    }
              {
              else
                {
               valReturnK = PCL_VOTER_FAIL;
                }
          {
         else
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
            {
           valReturnK = PCL_VOTER_FAIL;
           }
}

 catch(...)
    {
     valReturnK = PCL_VOTER_FAIL;
    }

  return valReturnK;
}



// PV12 - LH2 Inboard F/D Valve - OPEN

int MPS_PCL_PV13_OP_CMD::L()
{
  int  valReturnL = PCL_VOTER_FAIL;

try
{
           if (NMPSPV12INDSTATE.getValue() == MPS_VALVE_IND_STATE::OPEN)
     {
             valReturnL = PCL_VOTER_PASS;
            }
else
           {
       valReturnL = PCL_VOTER_FAIL;
           }
}

catch(...)
{
     valReturnL = PCL_VOTER_FAIL;
}

  return valReturnL;
}



// PD2 - LH2 ORB/ET Feed Disconnect Valve - OPEN

int MPS_PCL_PV13_OP_CMD::M()
{
  int  valReturnM = PCL_VOTER_FAIL;

try
{
           if (NMPSPD2INDSTATE.getValue() == MPS_VALVE_IND_STATE::OPEN)
     {
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
            valReturnM = PCL_VOTER_PASS;
           }
else
           {
      valReturnM = PCL_VOTER_FAIL;
           }
}

catch(...)
{
    valReturnM = PCL_VOTER_FAIL;
}

 return valReturnM;
}
```

```
// A75620 - Xfer Line Purge SOV - CLOSED

int MPS_PCL_PV13_OP_CMD::N()
{
 int  valReturnN = PCL_VOTER_FAIL;

try
{
           if (NLH2_VLV_A75620.getValue() == LH2_VALVE_STATE::CLOSED)
    {
            valReturnN = PCL_VOTER_PASS;
           }
else
           {
      valReturnN = PCL_VOTER_FAIL;
           }
}

catch(...)
{
    valReturnN = PCL_VOTER_FAIL;
}

return valReturnN;
}
```

```
// A75621 - Xfer Line Purge Valve - CLOSED

int MPS_PCL_PV13_OP_CMD::O()
{
 int  valReturnO = PCL_VOTER_FAIL;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
try
{
            if (NLH2_VLV_A75621.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnO = PCL_VOTER_PASS;
             }
else
            {
       valReturnO = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnO = PCL_VOTER_FAIL;
}

  return valReturnO;
}




// A75567 - 2K He RMT SOL SOV - CLOSED

int MPS_PCL_PV13_OP_CMD::P()
{
  int  valReturnP = PCL_VOTER_FAIL;

 try
{
            if (NLH2_VLV_A75567.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnP = PCL_VOTER_PASS;
             }
else
            {
       valReturnP = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnP = PCL_VOTER_FAIL;
}

  return valReturnP;
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
// A75586 - 2K He RMT SOV - CLOSED

int MPS_PCL_PV13_OP_CMD::Q()
{
  int  valReturnQ = PCL_VOTER_FAIL;

try
{
            if (NLH2_VLV_A75586.getValue() == LH2_VALVE_STATE::CLOSED)
     {
              valReturnQ = PCL_VOTER_PASS;
             }
else
            {
       valReturnQ = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnQ = PCL_VOTER_FAIL;
}

  return valReturnQ;
}
```

```
MPS_PCL_PV13_OP_CMD mps_pcl_pv13_op_cmd(seqName);
```

3.4.19. MPS_PCL_PD2_CL_CMD

3.4.19.1. Description

This PCL provides protection for the MPS LH2 ORB/ET Feed Disconnect Valve Close Command.
This class inherits from the class CAS_Control_Logic. The Algorithm will be implemented directly in C++ Code.

Reference - LPS Program GCH06

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.4.19.1.1. Instance

Instance Name
MPS_PCL_PD2_CL_CMD

3.4.19.1.2. PCL Initiating Commands

Command
V41K1416XL
V41K1417XL
V41K1418XL

3.4.19.1.3. Operations Index

Operation	Description
determineState()	This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE INHIBITED ERROR
activate()	This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.
inhibit()	This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.
getState()	This method is inherited from the base class - CAS_Control_Logic
resetCLNotification()	This method is inherited from the base class - CAS_Control_Logic

3.4.19.2. Operations

3.4.19.2.1. getState()

3.4.19.2.1.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.19.2.2. resetCLNotification()

3.4.19.2.2.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.4.19.2.3. determineState()

3.4.19.2.3.1. Descriptive Text

This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE, INHIBITED or ERROR.

3.4.19.2.3.2. Sequence

```
if  (<V41K1416XL>.PCLSTAT = active  AND
    <V41K1417XL>.PCLSTAT = active  AND
    <V41K1418XL>.PCLSTAT = active)
 {
 CLState = ACTIVE;
 }
else
 {
  if  (<V41K1416XL>.PCLSTAT = inhibited AND
      <V41K1417XL>.PCLSTAT = inhibited  AND
      <V41K1418XL>.PCLSTAT = inhibited)
   {
   CLState = INHIBITED;
    }
  else
    {
   CLState = ERROR;
    }
  }
```

3.4.19.2.4. activate()

3.4.19.2.4.1. Descriptive Text

This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.19.2.4.2. Sequence

```
A FD <V41K1416XL> PCL;
A FD <V41K1417XL> PCL;
A FD <V41K1418XL> PCL;

return;
```

3.4.19.2.5. inhibit()

3.4.19.2.5.1. Descriptive Text

This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.19.2.5.2. Sequence

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
I FD <V41K1416XL> PCL;
I FD <V41K1417XL> PCL;
I FD <V41K1418XL> PCL;

return;
```

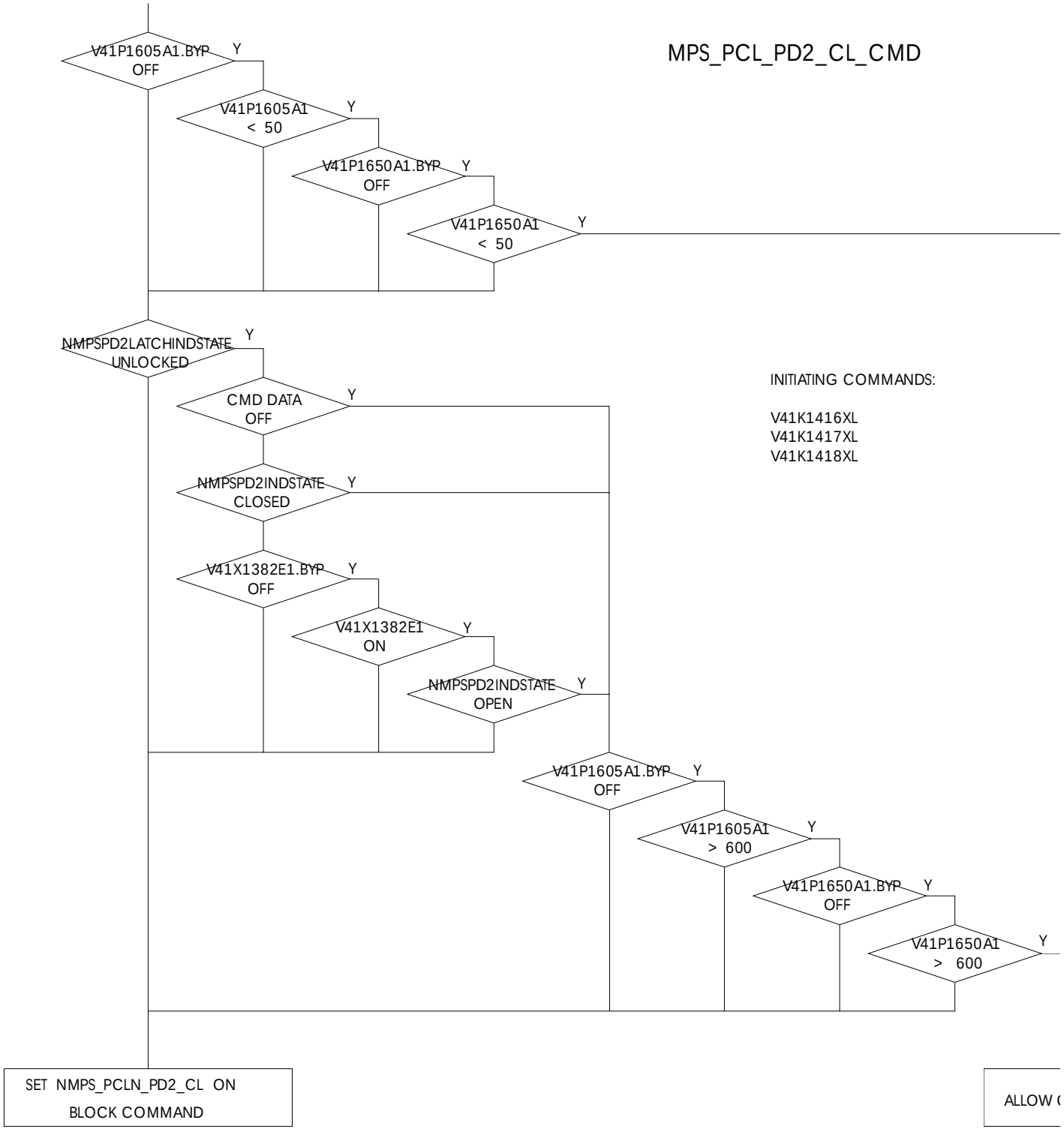

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.4.19.3. Algorithm

3.4.19.3.1. PCL Algorithm

3.4.19.3.1.1. Flowchart

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



3.4.19.3.1.2. Sequence

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
#include "mps_pcl_pd2_cl_cmd.h"
#include "SME.h"
#include "LO2.h"
#include "MPS.h"
#include "LH2.h"

static const char* seqName = "MPS_PCL_PD2_CL_CMD";
static const char* algorithmDescription = "Revision Date - 01/20/2000\n\
  if ((A && B) || (!B && C && D) || (!B && E && (H || (F && I) || G) && J && K))\n\
where:   A  = NMPS_PCLO_PD2_CL - PCL Emergency Override -  ON\n\
         B  = NLHK0017X - LH2 LOADING PREPS COMPLETE - ON\n\
         C  = MPS Pneu Reg Press < 50 psia\n\
         D  = MPS Pneu Accum Press < 50 psia\n\
         E  = PD2 - LH2 ORB/ET FEED DISC VALVE LATCH - UNLOCKED\n\
         F  = PD2 - LH2 ORB/ET FEED DISC VALVE OPEN POWER - ON\n\
         G  = PD2 - LH2 ORB/ET FEED DISC VALVE - CLOSED\n\
         H  = COMMAND DATA - OFF\n\
         I  = PD2 - LH2 ORB/ET FEED DISC VALVE - OPEN\n\
         J  = MPS Pneu Reg Press > 600 psia\n\
         K  = MPS Pneu Accum Press > 600 psia";

MPS_PCL_PD2_CL_CMD::MPS_PCL_PD2_CL_CMD(const char*
seqName):asv_pcsVPCL_DS_ON_OFF_(seqName)
{
  userInit();
}

MPS_PCL_PD2_CL_CMD::~MPS_PCL_PD2_CL_CMD()
{
}


void MPS_PCL_PD2_CL_CMD::userInit(void)
{
  initiatingFD(V41K1416XL);
  initiatingFD(V41K1417XL);
  initiatingFD(V41K1418XL);
}

int MPS_PCL_PD2_CL_CMD::userExecute(Discrete::ON_OFF dsVar)
{

if  ((A() && B()) || (!B() && C() && D()) || (!B() && E() && (H() || (F() && I()) || G()) && J() && K()))
   {
    return PCL_ALLOW_CMD;
   }
else
   {
    NMPS_PCLN_PD2_CL.Set(Discrete::On);
    return PCL_BLOCK_CMD;
   }

}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
// Function that determines the state of the indicator bypasses.

int MPS_PCL_PD2_CL_CMD::indicatorBypass(PD_ &pseudo_ind)
{
  if ( Discrete::On == pseudo_ind.getValue())
   return IND_BYPASS_ON;
 else
   return IND_BYPASS_OFF;
}



// Emergency Override ON

int MPS_PCL_PD2_CL_CMD::A()
{
  int  valReturnA = PCL_VOTER_FAIL;

    try
    {
      valReturnA =  (Discrete::On == (NMPS_PCLO_PD2_CL.getValue()));
    }

    catch(...)
    {
      valReturnA = PCL_VOTER_FAIL;
    }

  return valReturnA;
}



// LH2 Loading Preps Complete

int MPS_PCL_PD2_CL_CMD::B()
{
  int  valReturnB = PCL_VOTER_FAIL;

    try
    {
      valReturnB =  (Discrete::On == (NLHK0017X.getValue()));
    }

    catch(...)
    {
      valReturnB = PCL_VOTER_FAIL;
    }

  return valReturnB;
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
// MPS Pneu Reg Press < 50

int MPS_PCL_PD2_CL_CMD::C()
{
  int  valReturnC = PCL_VOTER_FAIL;

  if (indicatorBypass(N41P1605A1))
  {
    try
    {
      valReturnC =  (V41P1605A1.getValue() < 50);
    }

    catch(...)
    {
      valReturnC = PCL_VOTER_FAIL;
    }
  }

  else
  {
    valReturnC = PCL_VOTER_FAIL;
  }

  return valReturnC;
}




// MPS Pneu Accum Press < 50

int MPS_PCL_PD2_CL_CMD::D()
{
  int  valReturnD = PCL_VOTER_FAIL;

  if (indicatorBypass(N41P1650A1))
  {
    try
    {
      valReturnD =  (V41P1650A1.getValue() < 50);
    }

    catch(...)
    {
      valReturnD = PCL_VOTER_FAIL;
    }
  }

  else
  {
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
    valReturnD = PCL_VOTER_FAIL;
  }

  return valReturnD;
}



// PD2 - LH2 ORB/ET Feed Disconnect Latch - Unlocked

int MPS_PCL_PD2_CL_CMD::E()
{
  int  valReturnE = PCL_VOTER_FAIL;

try
{
            if (NMPSPD2LATCHINDSTATE.getValue() == MPS_ORB_LATCH_IND::UNLOCKED
           {
              valReturnE = PCL_VOTER_PASS;
             }
else
            {
       valReturnE = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnE = PCL_VOTER_FAIL;
}

  return valReturnE;
}







// PD2 OPEN POWER - ON

int MPS_PCL_PD2_CL_CMD::F()
{
  int  valReturnF = PCL_VOTER_FAIL;

  if (indicatorBypass(N41X1382E1))
  {
    try
    {
      valReturnF = (Discrete::On == (V41X1382E1.getValue()));
    }
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
   catch(...)
   {
    valReturnF = PCL_VOTER_FAIL;
   }
  }

  else
  {
   valReturnF = PCL_VOTER_FAIL;
  }

  return valReturnF;
}



//PD2 - LH2 ORB/ET Feed Disc Valve - CLOSED

int MPS_PCL_PD2_CL_CMD::G()
{
  int  valReturnG = PCL_VOTER_FAIL;

try
{
            if (NMPSPD2INDSTATE.getValue() == MPS_VALVE_IND_STATE::CLOSED)
     {
              valReturnG = PCL_VOTER_PASS;
             }
else
            {
       valReturnG = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnG = PCL_VOTER_FAIL;
}

  return valReturnG;
}




// COMMAND DATA - OFF

int MPS_PCL_PD2_CL_CMD::H()
{
  int  valReturnH = PCL_VOTER_FAIL;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
   try
   {
    valReturnH =  (Discrete::Off == dsVar);
   }

   catch(...)
   {
    valReturnH = PCL_VOTER_FAIL;
   }

  return valReturnH;
}



//PD2 - LH2 ORB/ET Feed Disc Valve - OPEN

int MPS_PCL_PD2_CL_CMD::I()
{
  int  valReturnI = PCL_VOTER_FAIL;

try
{
            if (NMPSPD2INDSTATE.getValue() == MPS_VALVE_IND_STATE::OPEN)
     {
              valReturnI = PCL_VOTER_PASS;
             }
else
            {
       valReturnI = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnI = PCL_VOTER_FAIL;
}

  return valReturnI;
}




// MPS Pneu Reg Press > 600

int MPS_PCL_PD2_CL_CMD::J()
{
  int  valReturnJ = PCL_VOTER_FAIL;

  if (indicatorBypass(N41P1605A1))
  {
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
   try
   {
    valReturnJ =  (V41P1605A1.getValue() > 600);
   }

   catch(...)
   {
    valReturnJ = PCL_VOTER_FAIL;
   }
  }

  else
  {
   valReturnJ = PCL_VOTER_FAIL;
  }

  return valReturnJ;
}




// MPS Pneu Accum Press > 600

int MPS_PCL_PD2_CL_CMD::K()
{
  int  valReturnK = PCL_VOTER_FAIL;

  if (indicatorBypass(N41P1650A1))
  {
   try
   {
    valReturnK =  (V41P1650A1.getValue() > 600);
   }

   catch(...)
   {
    valReturnK = PCL_VOTER_FAIL;
   }
  }

  else
  {
   valReturnK = PCL_VOTER_FAIL;
  }

  return valReturnK;
}


MPS_PCL_PD2_CL_CMD mps_pcl_pd2_cl_cmd(seqName);
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.4.20. MPS_PCL_PD2_OP_CMD

3.4.20.1. Description

This PCL provides protection for the MPS LH2 ORB/ET Feed Disconnect Valve Open Commands.
This class inherits from the class CAS_Control_Logic. The Algorithm will be implemented directly in C++ Code.

Reference - LPS Program GCH30

3.4.20.1.1. Instance

Instance Name
MPS_PCL_PD2_OP_CMD

3.4.20.1.2. PCL Initiating Commands

Command
V41K1413XL
V41K1414XL
V41K1415XL

3.4.20.1.3. Operations Index

Operation	**Description**
determineState()	This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE INHIBITED ERROR
activate()	This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.
inhibit()	This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.
getState()	This method is inherited from the base class - CAS_Control_Logic

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

resetCLNotification()	This method is inherited from the base class - CAS_Control_Logic

3.4.20.2. Operations

3.4.20.2.1. getState()

3.4.20.2.1.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.20.2.2. resetCLNotification()

3.4.20.2.2.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.20.2.3. determineState()

3.4.20.2.3.1. Descriptive Text

This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE, INHIBITED or ERROR.

3.4.20.2.3.2. Sequence

```
if  (<V41K1413XL>.PCLSTAT = active  AND
     <V41K1414XL>.PCLSTAT = active  AND
     <V41K1415XL>.PCLSTAT = active)
  {
  CLState = ACTIVE;
  }
else
  {
   if  (<V41K1413XL>.PCLSTAT = inhibited AND
        <V41K1414XL>.PCLSTAT = inhibited  AND
        <V41K1415XL>.PCLSTAT = inhibited)
     {
     CLState = INHIBITED;
      }
    else
      {
     CLState = ERROR;
      }
   }
```

3.4.20.2.4. activate()

3.4.20.2.4.1. Descriptive Text

This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.20.2.4.2. Sequence

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

A FD <V41K1413XL> PCL;
A FD <V41K1414XL> PCL;
A FD <V41K1415XL> PCL;

return;

3.4.20.2.5. inhibit()

3.4.20.2.5.1. Descriptive Text

This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.20.2.5.2. Sequence

I FD <V41K1413XL> PCL;
I FD <V41K1414XL> PCL;
I FD <V41K1415XL> PCL;

return;

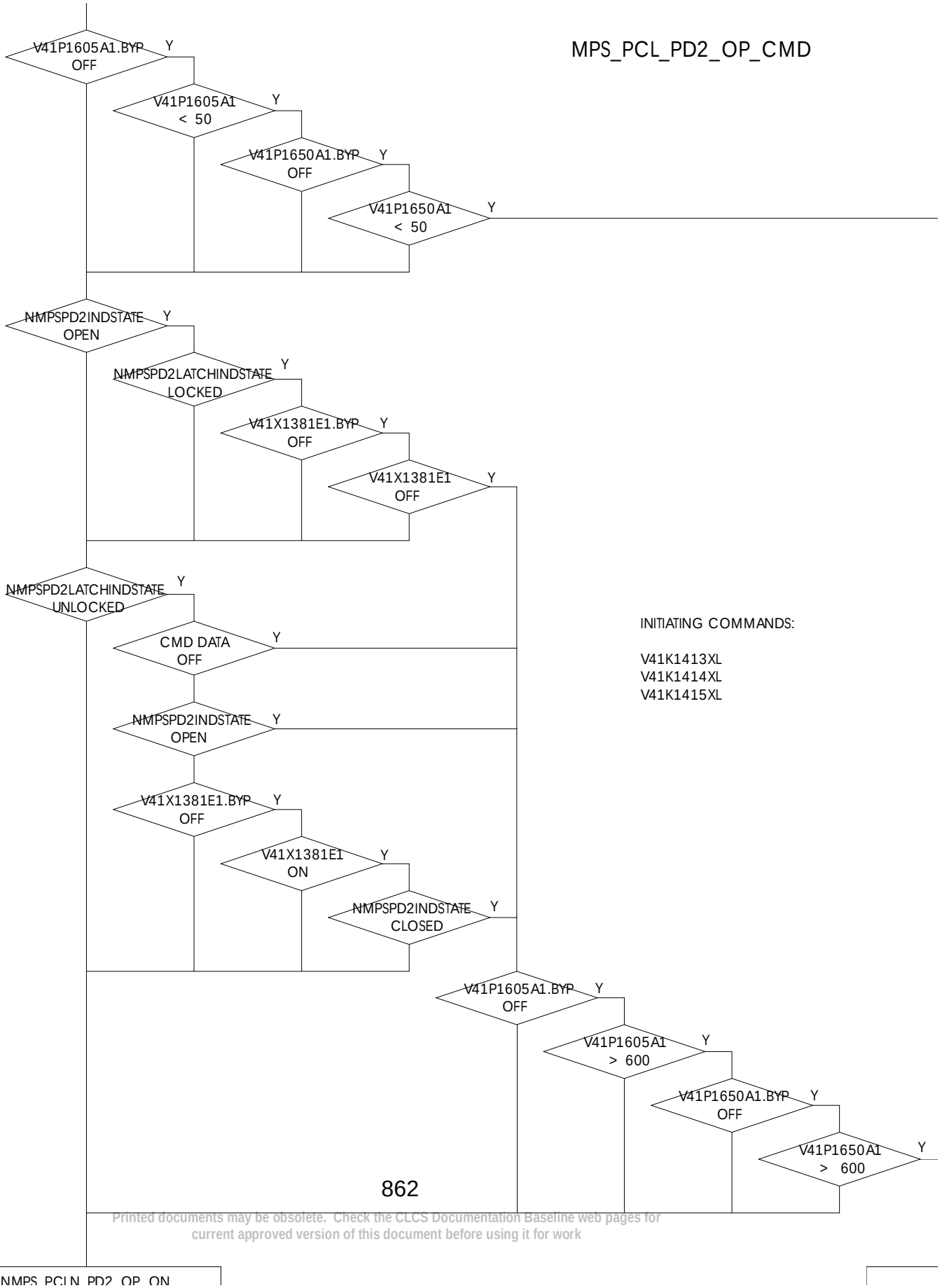
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.4.20.3. Algorithm

3.4.20.3.1. PCL Algorithm

3.4.20.3.1.1. Flowchart

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.4.20.3.1.2. Sequence

```
#include "mps_pcl_pd2_op_cmd.h"
#include "SME.h"
#include "LO2.h"
#include "MPS.h"
#include "LH2.h"

static const char* seqName = "MPS_PCL_PD2_OP_CMD";
static const char* algorithmDescription = "Revision Date - 01/20/2000\n\
  if ((A && B) || (!B && C && D) || (!B && E && F && G && J && K) || (!B && H && (L || (I && M) || E) &&
J && K))\n\
where:   A  = NMPS_PCLO_PD2_OP - PCL Emergency Override -  ON\n\
         B  = NLHK0017X - LH2 LOADING PREPS COMPLETE - ON\n\
         C  = MPS Pneu Reg Press < 50 psia\n\
         D  = MPS Pneu Accum Press < 50 psia\n\
         E  = PD2 - LH2 ORB/ET FEED DISC VALVE - OPEN\n\
         F  = PD2 - LH2 ORB/ET FEED DISC VALVE LATCH - LOCKED\n\
         G  = PD2 - LH2 ORB/ET FEED DISC VALVE CLOSE POWER - OFF\n\
         H  = PD2 - LH2 ORB/ET FEED DISC VALVE LATCH - UNLOCKED\n\
         I   = PD2 - LH2 ORB/ET FEED DISC VALVE CLOSE POWER - ON\n\
         J  = MPS Pneu Reg Press > 600 psia\n\
         K  = MPS Pneu Accum Press > 600 psia\n\
         L  = COMMAND DATA - OFF\n\
         M  = PD2 - LH2 ORB/ET FEED DISC VALVE - CLOSED";

MPS_PCL_PD2_OP_CMD::MPS_PCL_PD2_OP_CMD(const char*
seqName):asv_pcsVPCL_DS_ON_OFF_(seqName)
{
  userInit();
}

MPS_PCL_PD2_OP_CMD::~MPS_PCL_PD2_OP_CMD()
{
}


void MPS_PCL_PD2_OP_CMD::userInit(void)
{
  initiatingFD(V41K1413XL);
  initiatingFD(V41K1414XL);
  initiatingFD(V41K1415XL);
}

int MPS_PCL_PD2_OP_CMD::userExecute(Discrete::ON_OFF dsVar)
{

if  ((A() && B()) || (!B() && C() && D()) || (!B() && E() && F() && G() && J() && K()) || (!B() && H() &&
(L() || (I() && M()) || E()) && J() && K()))
  {
   return PCL_ALLOW_CMD;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
  }
else
  {

   NMPS_PCLN_PD2_OP.Set(Discrete::On);
   return PCL_BLOCK_CMD;
  }

}


// Function that determines the state of the indicator bypasses.

int MPS_PCL_PD2_OP_CMD::indicatorBypass(PD_ &pseudo_ind)
{
  if ( Discrete::On == pseudo_ind.getValue())
   return IND_BYPASS_ON;
 else
   return IND_BYPASS_OFF;
}



// Emergency Override ON

int MPS_PCL_PD2_OP_CMD::A()
{
  int  valReturnA = PCL_VOTER_FAIL;

    try
    {
      valReturnA =   (Discrete::On == (NMPS_PCLO_PD2_OP.getValue()));
    }

    catch(...)
    {
      valReturnA = PCL_VOTER_FAIL;
    }

  return valReturnA;
}



// LH2 Loading Preps Complete

int MPS_PCL_PD2_OP_CMD::B()
{
  int  valReturnB = PCL_VOTER_FAIL;

    try
    {
      valReturnB =  (Discrete::On == (NLHK0017X.getValue()));
    }
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
   catch(...)
   {
    valReturnB = PCL_VOTER_FAIL;
   }

 return valReturnB;
}



// MPS Pneu Reg Press < 50

int MPS_PCL_PD2_OP_CMD::C()
{
 int  valReturnC = PCL_VOTER_FAIL;

 if (indicatorBypass(N41P1605A1))
 {
  try
  {
   valReturnC =  (V41P1605A1.getValue() < 50);
  }

  catch(...)
  {
   valReturnC = PCL_VOTER_FAIL;
  }
 }

 else
 {
  valReturnC = PCL_VOTER_FAIL;
 }

 return valReturnC;
}



// MPS Pneu Accum Press < 50

int MPS_PCL_PD2_OP_CMD::D()
{
 int  valReturnD = PCL_VOTER_FAIL;

 if (indicatorBypass(N41P1650A1))
 {
  try
  {
   valReturnD =  (V41P1650A1.getValue() < 50);
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
    }

    catch(...)
    {
      valReturnD = PCL_VOTER_FAIL;
    }
  }

  else
  {
    valReturnD = PCL_VOTER_FAIL;
  }

  return valReturnD;
}




//PD2 - LO2 ORB/ET Feed Disc Valve - OPEN

int MPS_PCL_PD2_OP_CMD::E()
{
  int  valReturnE = PCL_VOTER_FAIL;

try
{
            if (NMPSPD2INDSTATE.getValue() == MPS_VALVE_IND_STATE::OPEN)
     {
              valReturnE = PCL_VOTER_PASS;
             }
else
            {
       valReturnE = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnE = PCL_VOTER_FAIL;
}

  return valReturnE;
}




// PD2 - LO2 ORB/ET Feed Disconnect Latch - Locked

int MPS_PCL_PD2_OP_CMD::F()
{
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
  int  valReturnF = PCL_VOTER_FAIL;

try
{
            if (NMPSPD2LATCHINDSTATE.getValue() == MPS_ORB_LATCH_IND::LOCKED)
           {
              valReturnF = PCL_VOTER_PASS;
             }
else
            {
       valReturnF = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnF = PCL_VOTER_FAIL;
}

  return valReturnF;
}




// PD2 CLOSE POWER - OFF

int MPS_PCL_PD2_OP_CMD::G()
{
  int  valReturnG = PCL_VOTER_FAIL;

  if (indicatorBypass(N41X1381E1))
  {
    try
    {
      valReturnG =  (Discrete::Off == (V41X1381E1.getValue()));
    }

    catch(...)
    {
      valReturnG = PCL_VOTER_FAIL;
    }
  }

  else
  {
    valReturnG = PCL_VOTER_FAIL;
  }

  return valReturnG;
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
// PD2 - LH2 ORB/ET Feed Disconnect Latch - Unlocked

int MPS_PCL_PD2_OP_CMD::H()
{
  int  valReturnH = PCL_VOTER_FAIL;

try
{
            if (NMPSPD2LATCHINDSTATE.getValue() == MPS_ORB_LATCH_IND::UNLOCKED)
           {
              valReturnH = PCL_VOTER_PASS;
             }
else
            {
       valReturnH = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnH = PCL_VOTER_FAIL;
}

  return valReturnH;
}




// PD2 CLOSE POWER - ON

int MPS_PCL_PD2_OP_CMD::I()
{
  int  valReturnI = PCL_VOTER_FAIL;

  if (indicatorBypass(N41X1381E1))
  {
    try
    {
      valReturnI =  (Discrete::On == (V41X1381E1.getValue()));
    }

    catch(...)
    {
      valReturnI = PCL_VOTER_FAIL;
    }
  }

  else
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
  {
    valReturnI = PCL_VOTER_FAIL;
  }

  return valReturnI;
}




// MPS Pneu Reg Press > 600

int MPS_PCL_PD2_OP_CMD::J()
{
  int  valReturnJ = PCL_VOTER_FAIL;

  if (indicatorBypass(N41P1605A1))
  {
    try
    {
      valReturnJ =  (V41P1605A1.getValue() > 600);
    }

    catch(...)
    {
      valReturnJ = PCL_VOTER_FAIL;
    }
  }

  else
  {
    valReturnJ = PCL_VOTER_FAIL;
  }

  return valReturnJ;
}




// MPS Pneu Accum Press > 600

int MPS_PCL_PD2_OP_CMD::K()
{
  int  valReturnK = PCL_VOTER_FAIL;

  if (indicatorBypass(N41P1650A1))
  {
    try
    {
      valReturnK =  (V41P1650A1.getValue() > 600);
    }
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
   catch(...)
   {
    valReturnK = PCL_VOTER_FAIL;
   }
  }

  else
  {
   valReturnK = PCL_VOTER_FAIL;
  }

  return valReturnK;
}




// COMMAND DATA - OFF

int MPS_PCL_PD2_OP_CMD::L()
{
  int  valReturnL = PCL_VOTER_FAIL;

   try
   {
    valReturnL =  (Discrete::Off == dsVar);
   }

   catch(...)
   {
    valReturnL = PCL_VOTER_FAIL;
   }

  return valReturnL;
}




//PD2 - LH2 ORB/ET Feed Disc Valve - CLOSED

int MPS_PCL_PD2_OP_CMD::M()
{
  int  valReturnM = PCL_VOTER_FAIL;

try
{
          if (NMPSPD2INDSTATE.getValue() == MPS_VALVE_IND_STATE::CLOSED)
    {
           valReturnM = PCL_VOTER_PASS;
          }
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
else
        {
    valReturnM = PCL_VOTER_FAIL;
        }
}

catch(...)
{
   valReturnM = PCL_VOTER_FAIL;
}

 return valReturnM;
}


MPS_PCL_PD2_OP_CMD mps_pcl_pd2_op_cmd(seqName);
```

3.4.21. MPS_PCL_PD2_LATCH_LK_CMD

3.4.21.1. Description

This PCL provides protection for the MPS LH2 ORB/ET Feed Disconnect Valve Latch Lock Command. This class inherits from the class CAS_Control_Logic. The Algorithm will be implemented directly in C++ Code.

Reference - LPS Program GCH37

3.4.21.1.1. Instance

Instance Name
MPS_PCL_PD2_LATCH_LK_CMD

3.4.21.1.2. PCL Initiating Commands

Command
V41K1981XL
V41K1982XL
V41K1983XL

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.4.21.1.3. Operations Index

Operation	Description
determineState()	This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE INHIBITED ERROR
activate()	This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.
inhibit()	This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.
getState()	This method is inherited from the base class - CAS_Control_Logic
resetCLNotification()	This method is inherited from the base class - CAS_Control_Logic

3.4.21.2. Operations

3.4.21.2.1. getState()

3.4.21.2.1.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.21.2.2. resetCLNotification()

3.4.21.2.2.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.21.2.3. determineState()

3.4.21.2.3.1. Descriptive Text

This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE, INHIBITED or ERROR.

3.4.21.2.3.2. Sequence

```
if  (<V41K1981XL>.PCLSTAT = active  AND
    <V41K1982XL>.PCLSTAT = active  AND
    <V41K1983XL>.PCLSTAT = active)
  {
  CLState = ACTIVE;
  }
else
  {
   if  (<V41K1981XL>.PCLSTAT = inhibited AND
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
    <V41K1982XL>.PCLSTAT = inhibited  AND
    <V41K1983XL>.PCLSTAT = inhibited)
  {
  CLState = INHIBITED;
   }
 else
   {
  CLState = ERROR;
   }
}
```

3.4.21.2.4. activate()

3.4.21.2.4.1. Descriptive Text

This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.21.2.4.2. Sequence

```
A FD <V41K1981XL> PCL;
A FD <V41K1982XL> PCL;
A FD <V41K1983XL> PCL;

return;
```

3.4.21.2.5. inhibit()

3.4.21.2.5.1. Descriptive Text

This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.21.2.5.2. Sequence

```
I FD <V41K1981XL> PCL;
I FD <V41K1982XL> PCL;
I FD <V41K1983XL> PCL;

return;
```

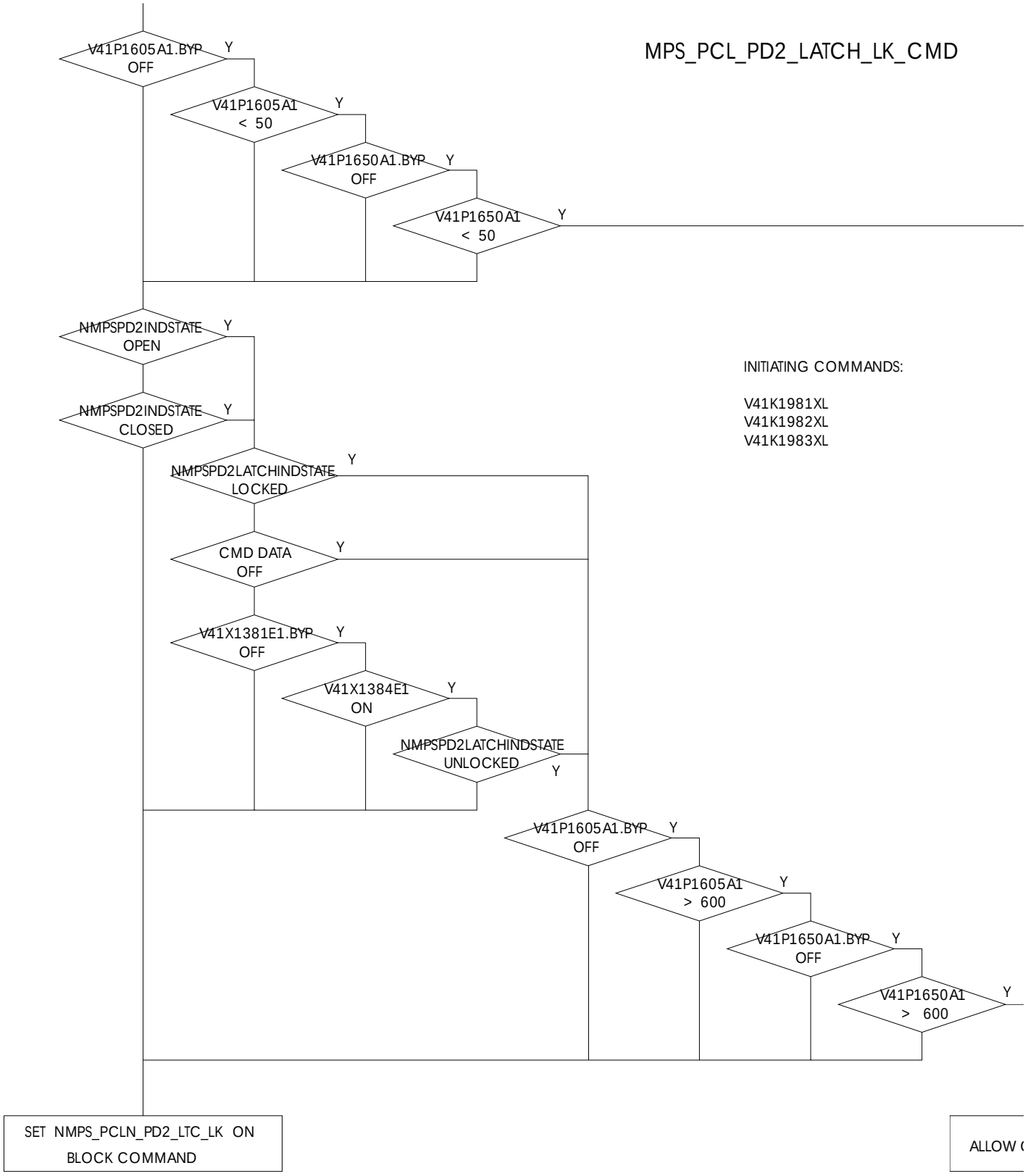

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.4.21.3. Algorithm

3.4.21.3.1. PCL Algorithm

3.4.21.3.1.1. Flowchart

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.4.21.3.1.2. Sequence

```
#include "mps_pcl_pd2_latch_lk_cmd.h"
#include "SME.h"
#include "LO2.h"
#include "MPS.h"
#include "LH2.h"

static const char* seqName = "MPS_PCL_PD2_LATCH_LK_CMD";
static const char* algorithmDescription = "Revision Date - 01/20/2000\n\
  if   (A  || (B && C) || ((D || E) && (G || (F && H) || I) && J && K))\n\
where:   A  = NMPS_PCLO_PD2_LTC_LK - PCL Bypass -  ON\n\
         B  = MPS Pneu Reg Press < 50 psia\n\
         C  = MPS Pneu Accum Press < 50 psia\n\
         D  = PD2 - LH2 ORB/ET FEED DISC VALVE - OPEN\n\
         E  = PD2 - LH2 ORB/ET FEED DISC VALVE - CLOSED\n\
         F  = PD2 - LH2 ORB/ET FEED DISC VALVE LATCH UNLOCK POWER - ON\n\
         G  = PD2 - LH2 ORB/ET FEED DISC VALVE LATCH - LOCKED\n\
         H  = PD2 - LH2 ORB/ET FEED DISC VALVE LATCH - UNLOCKED\n\
         I   = COMMAND DATA - OFF\n\
         J  = MPS Pneu Reg Press > 600 psia\n\
         K  = MPS Pneu Accum Press > 600 psia";

MPS_PCL_PD2_LATCH_LK_CMD::MPS_PCL_PD2_LATCH_LK_CMD
(const char* seqName):asv_pcsVPCL_DS_ON_OFF_(seqName)
{
  userInit();
}

MPS_PCL_PD2_LATCH_LK_CMD::~MPS_PCL_PD2_LATCH_LK_CMD()
{
}


void MPS_PCL_PD2_LATCH_LK_CMD::userInit(void)
{
  initiatingFD(V41K1981XL);
  initiatingFD(V41K1982XL);
  initiatingFD(V41K1983XL);
}

int MPS_PCL_PD2_LATCH_LK_CMD::userExecute(Discrete::ON_OFF dsVar)
{

if   (A() || (B() && C()) || ((D() || E()) && (G() || (F() && H()) || I()) && J() && K()))
  {
   return PCL_ALLOW_CMD;
  }
else
  {
   NMPS_PCLN_PD2_LTC_LK.Set(Discrete::On);
   return PCL_BLOCK_CMD;
  }
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
}


// Function that determines the state of the indicator bypasses.

int MPS_PCL_PD2_LATCH_LK_CMD::indicatorBypass(PD_ &pseudo_ind)
{
  if ( Discrete::On == pseudo_ind.getValue())
   return IND_BYPASS_ON;
 else
   return IND_BYPASS_OFF;
}



// PCL Bypass - ON

int MPS_PCL_PD2_LATCH_LK_CMD::A()
{
  int  valReturnA = PCL_VOTER_FAIL;

    try
    {
      valReturnA =  (Discrete::On == (NMPS_PCLO_PD2_LTC_LK.getValue()));
    }

    catch(...)
    {
      valReturnA = PCL_VOTER_FAIL;
    }

  return valReturnA;
}




// MPS Pneu Reg Press < 50

int MPS_PCL_PD2_LATCH_LK_CMD::B()
{
  int  valReturnB = PCL_VOTER_FAIL;

  if (indicatorBypass(N41P1605A1))
  {
    try
    {
      valReturnB =  (V41P1605A1.getValue() < 50);
    }

    catch(...)
    {
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
    valReturnB = PCL_VOTER_FAIL;
   }
  }

  else
  {
   valReturnB = PCL_VOTER_FAIL;
  }

  return valReturnB;
}




// MPS Pneu Accum Press < 50

int MPS_PCL_PD2_LATCH_LK_CMD::C()
{
  int  valReturnC = PCL_VOTER_FAIL;

  if (indicatorBypass(N41P1650A1))
  {
   try
   {
    valReturnC =  (V41P1650A1.getValue() < 50);
   }

   catch(...)
   {
    valReturnC = PCL_VOTER_FAIL;
   }
  }

  else
  {
   valReturnC = PCL_VOTER_FAIL;
  }

  return valReturnC;
}



//PD2 - LH2 ORB/ET Feed Disc Valve - OPEN

int MPS_PCL_PD2_LATCH_LK_CMD::D()
{
  int  valReturnD = PCL_VOTER_FAIL;

try
{
        if (NMPSPD2INDSTATE.getValue() == MPS_VALVE_IND_STATE::OPEN)
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
    {
            valReturnD = PCL_VOTER_PASS;
           }
else
           {
      valReturnD = PCL_VOTER_FAIL;
           }
}

catch(...)
{
    valReturnD = PCL_VOTER_FAIL;
}

 return valReturnD;
}



//PD2 - LH2 ORB/ET Feed Disc Valve - CLOSED

int MPS_PCL_PD2_LATCH_LK_CMD::E()
{
 int  valReturnE = PCL_VOTER_FAIL;

try
{
           if (NMPSPD2INDSTATE.getValue() == MPS_VALVE_IND_STATE::CLOSED)
    {
            valReturnE = PCL_VOTER_PASS;
           }
else
           {
      valReturnE = PCL_VOTER_FAIL;
           }
}

catch(...)
{
    valReturnE = PCL_VOTER_FAIL;
}

 return valReturnE;
}


// PD2 LATCH UNLOCK POWER - ON

int MPS_PCL_PD2_LATCH_LK_CMD::F()
{
 int  valReturnF = PCL_VOTER_FAIL;

 if (indicatorBypass(N41X1384E1))
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
 {
  try
  {
   valReturnF =  (Discrete::On == (V41X1384E1.getValue()));
  }

  catch(...)
  {
   valReturnF = PCL_VOTER_FAIL;
  }
 }

 else
 {
  valReturnF = PCL_VOTER_FAIL;
 }

 return valReturnF;
}



// PD2 - LH2 ORB/ET Feed Disconnect Latch - Locked

int MPS_PCL_PD2_LATCH_LK_CMD::G()
{
 int  valReturnG = PCL_VOTER_FAIL;

try
{
            if (NMPSPD2LATCHINDSTATE.getValue() == MPS_ORB_LATCH_IND::LOCKED)
           {
              valReturnG = PCL_VOTER_PASS;
            }
else
            {
      valReturnG = PCL_VOTER_FAIL;
            }
}

catch(...)
{
    valReturnG = PCL_VOTER_FAIL;
}

 return valReturnG;
}



// PD2 - LH2 ORB/ET Feed Disconnect Latch - Unlocked
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
int MPS_PCL_PD2_LATCH_LK_CMD::H()
{
  int  valReturnH = PCL_VOTER_FAIL;

try
{
            if (NMPSPD2LATCHINDSTATE.getValue() == MPS_ORB_LATCH_IND::UNLOCKED)
           {
              valReturnH = PCL_VOTER_PASS;
            }
else
            {
       valReturnH = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnH = PCL_VOTER_FAIL;
}

  return valReturnH;
}



// COMMAND DATA - OFF

int MPS_PCL_PD2_LATCH_LK_CMD::I()
{
  int  valReturnI = PCL_VOTER_FAIL;

    try
    {
      valReturnI =  (Discrete::Off == dsVar);
    }

    catch(...)
    {
      valReturnI = PCL_VOTER_FAIL;
    }

  return valReturnI;
}




// MPS Pneu Reg Press > 600

int MPS_PCL_PD2_LATCH_LK_CMD::J()
{
  int  valReturnJ = PCL_VOTER_FAIL;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
  if (indicatorBypass(N41P1605A1))
  {
    try
    {
      valReturnJ =  (V41P1605A1.getValue() > 600);
    }

    catch(...)
    {
      valReturnJ = PCL_VOTER_FAIL;
    }
  }

  else
  {
    valReturnJ = PCL_VOTER_FAIL;
  }

  return valReturnJ;
}




// MPS Pneu Accum Press > 600

int MPS_PCL_PD2_LATCH_LK_CMD::K()
{
  int  valReturnK = PCL_VOTER_FAIL;

  if (indicatorBypass(N41P1650A1))
  {
    try
    {
      valReturnK =  (V41P1650A1.getValue() > 600);
    }

    catch(...)
    {
      valReturnK = PCL_VOTER_FAIL;
    }
  }

  else
  {
    valReturnK = PCL_VOTER_FAIL;
  }

  return valReturnK;
}


MPS_PCL_PD2_LATCH_LK_CMD mps_pcl_pd2_latch_lk_cmd(seqName);
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.4.22. MPS_PCL_PD2_LATCH_UL_CMD

3.4.22.1. Description

This PCL provides protection for the MPS LH2 ORB/ET Feed Disconnect Valve Unlock Commands. This class inherits from the class CAS_Control_Logic. The Algorithm will be implemented directly in C++ Code.

Reference - LPS Program GCH38

3.4.22.1.1. Instance

Instance Name
MPS_PCL_PD2_LATCH_UL_CMD

3.4.22.1.2. PCL Initiating Commands

Command
V41K1984XL
V41K1985XL
V41K1986XL

3.4.22.1.3. Operations Index

Operation	**Description**
determineState()	This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE INHIBITED ERROR
activate()	This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.
inhibit()	This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

getState()	This method is inherited from the base class - CAS_Control_Logic
resetCLNotification()	This method is inherited from the base class - CAS_Control_Logic

3.4.22.2. Operations

3.4.22.2.1. getState()

3.4.22.2.1.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.22.2.2. resetCLNotification()

3.4.22.2.2.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.22.2.3. determineState()

3.4.22.2.3.1. Descriptive Text

This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE, INHIBITED or ERROR.

3.4.22.2.3.2. Sequence

```
if  (<V41K1984XL>.PCLSTAT = active  AND
    <V41K1985XL>.PCLSTAT = active  AND
    <V41K1986XL>.PCLSTAT = active)
  {
  CLState = ACTIVE;
  }
else
  {
   if  (<V41K1984XL>.PCLSTAT = inhibited AND
       <V41K1985XL>.PCLSTAT = inhibited  AND
       <V41K1986XL>.PCLSTAT = inhibited)
     {
     CLState = INHIBITED;
      }
    else
      {
     CLState = ERROR;
      }
   }
```

3.4.22.2.4. activate()

3.4.22.2.4.1. Descriptive Text

This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.4.22.2.4.2. Sequence

A FD <V41K1984XL> PCL;
A FD <V41K1985XL> PCL;
A FD <V41K1986XL> PCL;

return;

3.4.22.2.5. inhibit()

3.4.22.2.5.1. Descriptive Text

This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.22.2.5.2. Sequence

I FD <V41K1984XL> PCL;
I FD <V41K1985XL> PCL;
I FD <V41K1986XL> PCL;

return;

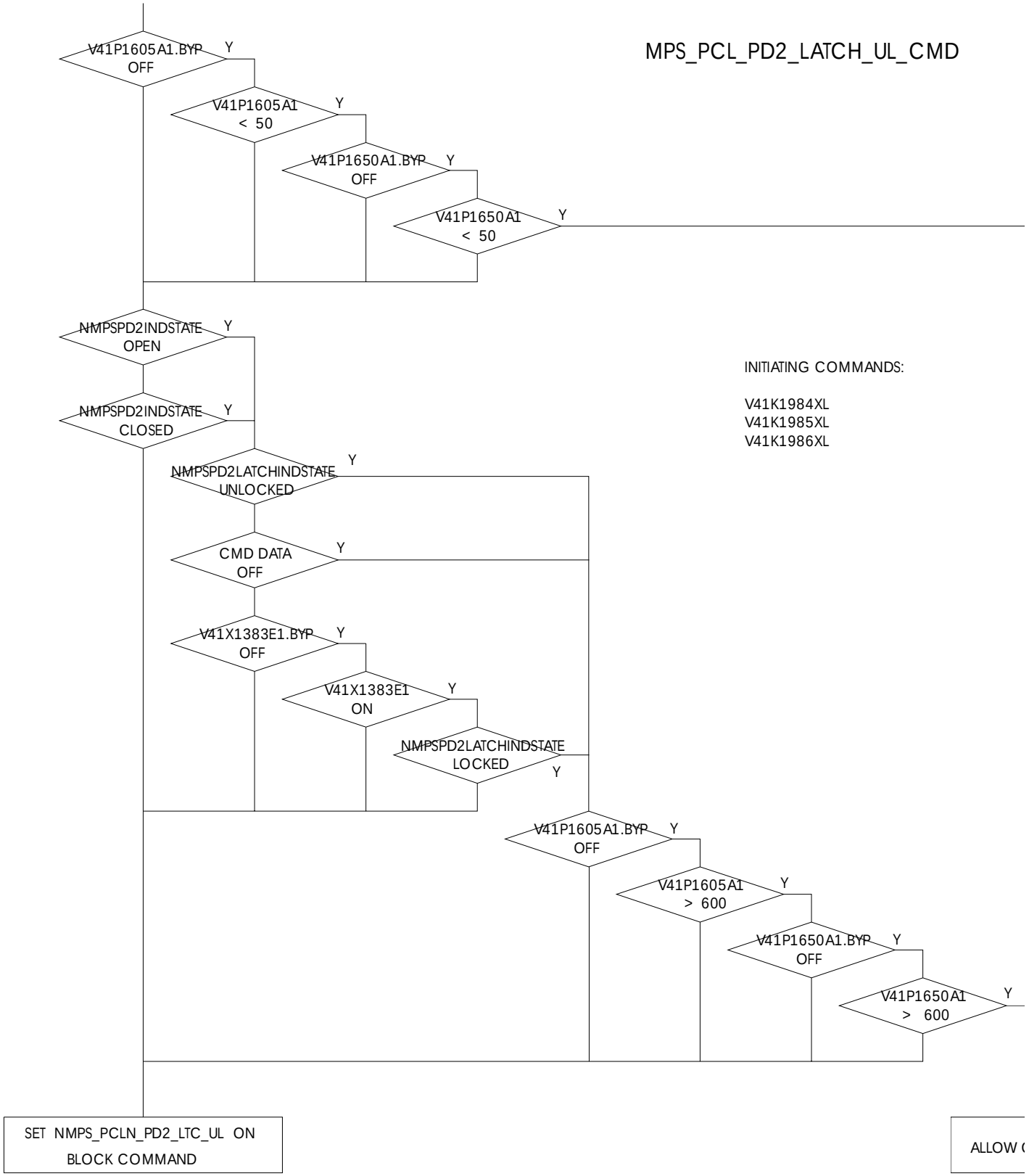
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.4.22.3. Algorithm

3.4.22.3.1. PCL Algorithm

3.4.22.3.1.1. Flowchart

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.4.22.3.1.2. Sequence

```
#include "mps_pcl_pd2_latch_ul_cmd.h"
#include "SME.h"
#include "LO2.h"
#include "MPS.h"
#include "LH2.h"

static const char* seqName = "MPS_PCL_PD2_LATCH_UL_CMD";
static const char* algorithmDescription = "Revision Datae - 01/20/2000\n\
  if   (A  || (B && C) || ((D || E) && (G || (F && H) || I) && J && K))\n\
where:   A  = NMPS_PCLO_PD2_LTC_UL - PCL Bypass -  ON\n\
           B  = MPS Pneu Reg Press < 50 psia\n\
           C  = MPS Pneu Accum Press < 50 psia\n\
           D  = PD2 - LH2 ORB/ET FEED DISC VALVE - OPEN\n\
           E  = PD2 - LH2 ORB/ET FEED DISC VALVE - CLOSED\n\
           F  = PD2 - LH2 ORB/ET FEED DISC VALVE LATCH LOCK POWER - ON\n\
           G  = PD2 - LH2 ORB/ET FEED DISC VALVE LATCH - UNLOCKED\n\
           H  = PD2 - LH2 ORB/ET FEED DISC VALVE LATCH - LOCKED\n\
           I   = COMMAND DATA - OFF\n\
           J  = MPS Pneu Reg Press > 600 psia\n\
           K  = MPS Pneu Accum Press > 600 psia";

MPS_PCL_PD2_LATCH_UL_CMD::MPS_PCL_PD2_LATCH_UL_CMD
(const char* seqName):asv_pcsVPCL_DS_ON_OFF_(seqName)
{
  userInit();
}

MPS_PCL_PD2_LATCH_UL_CMD::~MPS_PCL_PD2_LATCH_UL_CMD()
{
}


void MPS_PCL_PD2_LATCH_UL_CMD::userInit(void)
{
  initiatingFD(V41K1984XL);
  initiatingFD(V41K1985XL);
  initiatingFD(V41K1986XL);
}

int MPS_PCL_PD2_LATCH_UL_CMD::userExecute(Discrete::ON_OFF dsVar)
{

if   (A() || (B() && C()) || ((D() || E()) && (G() || (F() && H()) || I()) && J() && K()))
   {
    return PCL_ALLOW_CMD;
   }
else
   {
    NMPS_PCLN_PD2_LTC_UL.Set(Discrete::On);
    return PCL_BLOCK_CMD;
   }
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
}


// Function that determines the state of the indicator bypasses.

int MPS_PCL_PD2_LATCH_UL_CMD::indicatorBypass(PD_ &pseudo_ind)
{
  if ( Discrete::On == pseudo_ind.getValue())
   return IND_BYPASS_ON;
 else
   return IND_BYPASS_OFF;
}



// PCL Bypass - ON

int MPS_PCL_PD2_LATCH_UL_CMD::A()
{
  int  valReturnA = PCL_VOTER_FAIL;

    try
    {
      valReturnA =   (Discrete::On == (NMPS_PCLO_PD2_LTC_UL.getValue()));
    }

    catch(...)
    {
      valReturnA = PCL_VOTER_FAIL;
    }

  return valReturnA;
}




// MPS Pneu Reg Press < 50

int MPS_PCL_PD2_LATCH_UL_CMD::B()
{
  int  valReturnB = PCL_VOTER_FAIL;

  if (indicatorBypass(N41P1605A1))
  {
    try
    {
      valReturnB =  (V41P1605A1.getValue() < 50);
    }

    catch(...)
    {
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
    valReturnB = PCL_VOTER_FAIL;
   }
  }

  else
  {
   valReturnB = PCL_VOTER_FAIL;
  }

  return valReturnB;
}




// MPS Pneu Accum Press < 50

int MPS_PCL_PD2_LATCH_UL_CMD::C()
{
  int  valReturnC = PCL_VOTER_FAIL;

  if (indicatorBypass(N41P1650A1))
  {
   try
   {
    valReturnC =  (V41P1650A1.getValue() < 50);
   }

   catch(...)
   {
    valReturnC = PCL_VOTER_FAIL;
   }
  }

  else
  {
   valReturnC = PCL_VOTER_FAIL;
  }

  return valReturnC;
}



//PD2 - LH2 ORB/ET Feed Disc Valve - OPEN

int MPS_PCL_PD2_LATCH_UL_CMD::D()
{
  int  valReturnD = PCL_VOTER_FAIL;

try
{
        if (NMPSPD2INDSTATE.getValue() == MPS_VALVE_IND_STATE::OPEN)
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
    {
            valReturnD = PCL_VOTER_PASS;
           }
else
           {
      valReturnD = PCL_VOTER_FAIL;
           }
}

catch(...)
{
    valReturnD = PCL_VOTER_FAIL;
}

 return valReturnD;
}



//PD2 - LH2 ORB/ET Feed Disc Valve - CLOSED

int MPS_PCL_PD2_LATCH_UL_CMD::E()
{
 int  valReturnE = PCL_VOTER_FAIL;

try
{
           if (NMPSPD2INDSTATE.getValue() == MPS_VALVE_IND_STATE::CLOSED)
    {
            valReturnE = PCL_VOTER_PASS;
           }
else
           {
      valReturnE = PCL_VOTER_FAIL;
           }
}

catch(...)
{
    valReturnE = PCL_VOTER_FAIL;
}

 return valReturnE;
}


// PD2 LATCH LOCK POWER - ON

int MPS_PCL_PD2_LATCH_UL_CMD::F()
{
 int  valReturnF = PCL_VOTER_FAIL;

 if (indicatorBypass(N41X1383E1))
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
 {
  try
  {
   valReturnF = (Discrete::On == (V41X1383E1.getValue()));
  }

  catch(...)
  {
   valReturnF = PCL_VOTER_FAIL;
  }
 }

 else
 {
  valReturnF = PCL_VOTER_FAIL;
 }

 return valReturnF;
}



// PD2 - LH2 ORB/ET Feed Disconnect Latch - Unlocked

int MPS_PCL_PD2_LATCH_UL_CMD::G()
{
 int  valReturnG = PCL_VOTER_FAIL;

try
{
            if (NMPSPD2LATCHINDSTATE.getValue() == MPS_ORB_LATCH_IND::UNLOCKED)
           {
              valReturnG = PCL_VOTER_PASS;
            }
else
            {
      valReturnG = PCL_VOTER_FAIL;
            }
}

catch(...)
{
    valReturnG = PCL_VOTER_FAIL;
}

 return valReturnG;
}



// PD2 - LH2 ORB/ET Feed Disconnect Latch - Locked
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
int MPS_PCL_PD2_LATCH_UL_CMD::H()
{
  int  valReturnH = PCL_VOTER_FAIL;

try
{
            if (NMPSPD2LATCHINDSTATE.getValue() == MPS_ORB_LATCH_IND::LOCKED)
           {
              valReturnH = PCL_VOTER_PASS;
            }
else
            {
       valReturnH = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnH = PCL_VOTER_FAIL;
}

  return valReturnH;
}



// COMMAND DATA - OFF

int MPS_PCL_PD2_LATCH_UL_CMD::I()
{
  int  valReturnI = PCL_VOTER_FAIL;

    try
    {
      valReturnI =  (Discrete::Off == dsVar);
    }

    catch(...)
    {
      valReturnI = PCL_VOTER_FAIL;
    }

  return valReturnI;
}




// MPS Pneu Reg Press > 600

int MPS_PCL_PD2_LATCH_UL_CMD::J()
{
  int  valReturnJ = PCL_VOTER_FAIL;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
  if (indicatorBypass(N41P1605A1))
  {
    try
    {
      valReturnJ =  (V41P1605A1.getValue() > 600);
    }

    catch(...)
    {
      valReturnJ = PCL_VOTER_FAIL;
    }
  }

  else
  {
    valReturnJ = PCL_VOTER_FAIL;
  }

  return valReturnJ;
}




// MPS Pneu Accum Press > 600

int MPS_PCL_PD2_LATCH_UL_CMD::K()
{
  int  valReturnK = PCL_VOTER_FAIL;

  if (indicatorBypass(N41P1650A1))
  {
    try
    {
      valReturnK =  (V41P1650A1.getValue() > 600);
    }

    catch(...)
    {
      valReturnK = PCL_VOTER_FAIL;
    }
  }

  else
  {
    valReturnK = PCL_VOTER_FAIL;
  }

  return valReturnK;
}


MPS_PCL_PD2_LATCH_UL_CMD mps_pcl_pd2_latch_ul_cmd(seqName);
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.4.23. MPS_PCL_PD3_CL_CMD

3.4.23.1. Description

This PCL provides protection for the MPS LH2 ORB/ET Recirc Disconnect Valve Close Command. This class inherits from the class CAS_Control_Logic. The Algorithm will be implemented directly in C++ Code.

Reference - LPS Program GCH07

3.4.23.1.1. Instance

Instance Name
MPS_PCL_PD3_CL_CMD

3.4.23.1.2. PCL Initiating Commands

Command
V41K1422XL

3.4.23.1.3. Operations Index

Operation	**Description**
determineState()	This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE INHIBITED ERROR
activate()	This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.
inhibit()	This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.
getState()	This method is inherited from the base class - CAS_Control_Logic
resetCLNotification()	This method is inherited from the base class - CAS_Control_Logic

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.4.23.2. Operations

3.4.23.2.1. getState()

3.4.23.2.1.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.23.2.2. resetCLNotification()

3.4.23.2.2.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.23.2.3. determineState()

3.4.23.2.3.1. Descriptive Text

This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE, INHIBITED or ERROR.

3.4.23.2.3.2. Sequence

```
if  (<V41K1422XL>.PCLSTAT = active)
  {
  CLState = ACTIVE;
  }
else
  {
   if  (<V41K1422XL>.PCLSTAT = inhibited)
     {
     CLState = INHIBITED;
      }
    else
      {
     CLState = ERROR;
      }
   }
```

3.4.23.2.4. activate()

3.4.23.2.4.1. Descriptive Text

This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.23.2.4.2. Sequence

```
A FD <V41K1422XL> PCL;

return;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.4.23.2.5. inhibit()

3.4.23.2.5.1. Descriptive Text

This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.23.2.5.2. Sequence

I FD <V41K1422XL> PCL;

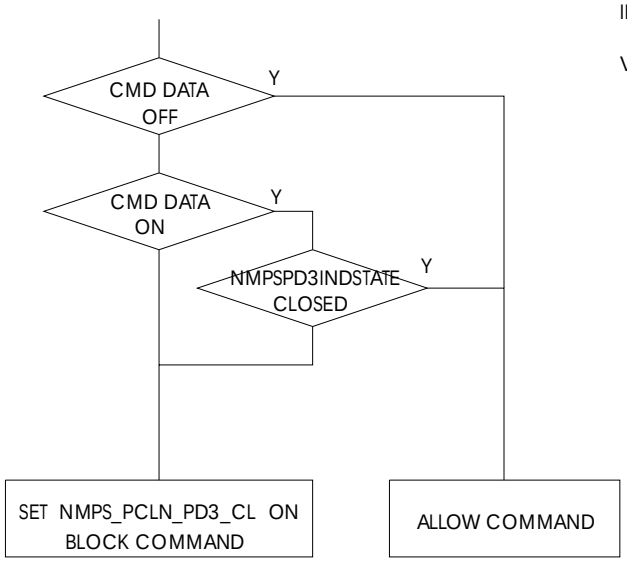
return;

3.4.23.3. Algorithm

3.4.23.3.1. PCL Algorithm

3.4.23.3.1.1. Flowchart

MPS_PCL_PD3_CL_CMD



3.4.23.3.1.2. Sequence

```
#include "mps_pcl_pd3_cl_cmd.h"
#include "SME.h"
#include "LO2.h"
#include "MPS.h"
#include "LH2.h"

static const char* seqName = "MPS_PCL_PD3_CL_CMD";
static const char* algorithmDescription = "Revision Date - 01/20/2000\n\
if (A  || B || (C && D))\n\
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
where:   A  = NMPS_PCLO_PD3_CL - PCL Bypass ON\n\
         B =  Command Data - OFF\n\
         C =  Command Data - ON\n\
         D  = PD3 - LH2 ORB/ET RECIRC DISC VALVE - CLOSED";

MPS_PCL_PD3_CL_CMD::MPS_PCL_PD3_CL_CMD(const char*
seqName):asv_pcsVPCL_DS_ON_OFF_(seqName)
{
  userInit();
}

MPS_PCL_PD3_CL_CMD::~MPS_PCL_PD3_CL_CMD()
{
}


void MPS_PCL_PD3_CL_CMD::userInit(void)
{
  initiatingFD(V41K1422XL);
}

int MPS_PCL_PD3_CL_CMD::userExecute(Discrete::ON_OFF dsVar)
{
  if (A() || B() || (C() && D()))
  {
   return PCL_ALLOW_CMD;
  }
  else
  {
   NMPS_PCLN_PD3_CL.Set(Discrete::On);
   return PCL_BLOCK_CMD;
  }
}




// PCL Override ON

int MPS_PCL_PD3_CL_CMD::A()
{
  int  valReturnA = PCL_VOTER_FAIL;

    try
    {
      valReturnA =  (Discrete::On == (NMPS_PCLO_PD3_CL.getValue()));
    }

    catch(...)
    {
      valReturnA = PCL_VOTER_FAIL;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
    }

  return valReturnA;
}




// COMMAND DATA - OFF

int MPS_PCL_PD3_CL_CMD::B()
{
  int  valReturnB = PCL_VOTER_FAIL;

    try
    {
      valReturnB =  (Discrete::Off == dsVar);
    }

    catch(...)
    {
      valReturnB = PCL_VOTER_FAIL;
    }

  return valReturnB;
}



// COMMAND DATA - ON

int MPS_PCL_PD3_CL_CMD::C()
{
  int  valReturnC = PCL_VOTER_FAIL;

    try
    {
      valReturnC =  (Discrete::On == dsVar);
    }

    catch(...)
    {
      valReturnC = PCL_VOTER_FAIL;
    }

  return valReturnC;
}



//PD3 - LH2 ORB/ET Recirc Disc Valve - CLOSED
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
int MPS_PCL_PD3_CL_CMD::D()
{
  int  valReturnD = PCL_VOTER_FAIL;

try
{
            if (NMPSPD3INDSTATE.getValue() == MPS_VALVE_IND_STATE::CLOSED)
     {
              valReturnD = PCL_VOTER_PASS;
             }
else
            {
       valReturnD = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnD = PCL_VOTER_FAIL;
}

  return valReturnD;
}
```

```
MPS_PCL_PD3_CL_CMD mps_pcl_pd3_cl_cmd(seqName);
```

3.4.24. MPS_PCL_PD3_OP_CMD

3.4.24.1. Description

This PCL provides protection for the MPS LH2 ORB/ET Recirc Disconnect Valve Open Command.
This class inherits from the class CAS_Control_Logic. The Algorithm will be implemented directly in C++ Code.

Reference - LPS Program GCH31

3.4.24.1.1. Instance

Instance Name
MPS_PCL_PD3_OP_CMD

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.4.24.1.2. PCL Initiating Commands

Command
V41K1421XL

3.4.24.1.3. Operations Index

Operation	Description
determineState()	This method will determine the enumerated state of the PCL sequence based on the associated command and will return either: ACTIVE INHIBITED ERROR
activate()	This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.
inhibit()	This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.
getState()	This method is inherited from the base class - CAS_Control_Logic
resetCLNotification()	This method is inherited from the base class - CAS_Control_Logic

3.4.24.2. Operations

3.4.24.2.1. getState()

3.4.24.2.1.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.24.2.2. resetCLNotification()

3.4.24.2.2.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.24.2.3. determineState()

3.4.24.2.3.1. Descriptive Text

This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE, INHIBITED or ERROR.

3.4.24.2.3.2. Sequence

if (<V41K1421XL>.PCLSTAT = active)

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
 {
 CLState = ACTIVE;
 }
else
 {
  if  (<V41K1421XL>.PCLSTAT = inhibited)
    {
    CLState = INHIBITED;
     }
   else
     {
    CLState = ERROR;
     }
  }
```

3.4.24.2.4. activate()

3.4.24.2.4.1. Descriptive Text

This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.24.2.4.2. Sequence

A FD <V41K1421XL> PCL;

return;

3.4.24.2.5. inhibit()

3.4.24.2.5.1. Descriptive Text

This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.24.2.5.2. Sequence

I FD <V41K1421XL> PCL;

return;

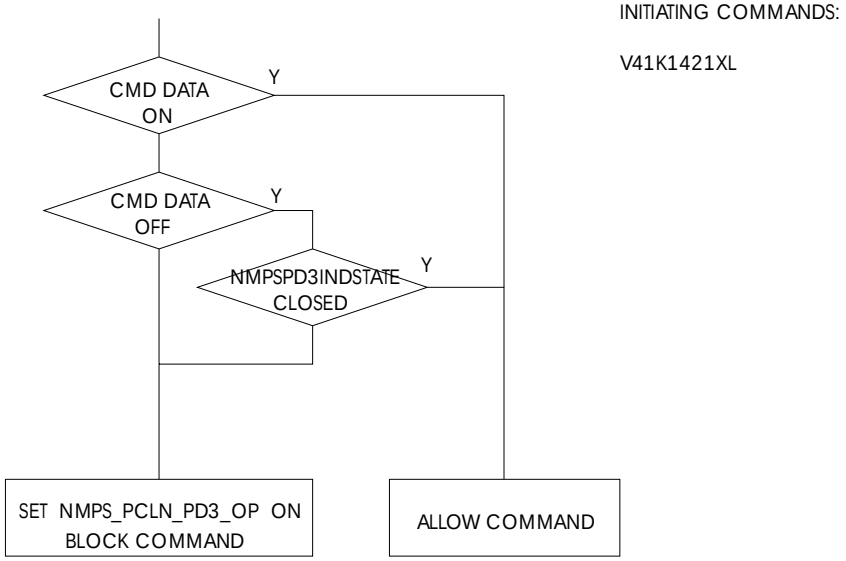
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.4.24.3. Algorithm

3.4.24.3.1. PCL Algorithm

3.4.24.3.1.1. Flowchart

MPS_PCL_PD3_OP_CMD



3.4.24.3.1.2. Sequence

```
#include "mps_pcl_pd3_op_cmd.h"
#include "SME.h"
#include "LO2.h"
#include "MPS.h"
#include "LH2.h"

static const char* seqName = "MPS_PCL_PD3_OP_CMD";
static const char* algorithmDescription = "Revision Date - 01/20/2000\n\
if (A  || B || (C && D))\n\
where:   A  = NMPS_PCLO_PD3_OP - PCL Bypass ON\n\
         B =  Command Data - ON\n\
         C =  Command Data - OFF\n\
         D  = PD3 - LH2 ORB/ET RECIRC DISC VALVE - CLOSED";

MPS_PCL_PD3_OP_CMD::MPS_PCL_PD3_OP_CMD(const char*
seqName):asv_pcsVPCL_DS_ON_OFF_(seqName)
{
  userInit();
}

MPS_PCL_PD3_OP_CMD::~MPS_PCL_PD3_OP_CMD()
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
{
}


void MPS_PCL_PD3_OP_CMD::userInit(void)
{
  initiatingFD(V41K1421XL);
}

int MPS_PCL_PD3_OP_CMD::userExecute(Discrete::ON_OFF dsVar)
{
  if (A() || B() || (C() && D()))
  {
   return PCL_ALLOW_CMD;
  }
  else
  {
  NMPS_PCLN_PD3_OP.Set(Discrete::On);
   return PCL_BLOCK_CMD;
  }
}




// PCL Override ON

int MPS_PCL_PD3_OP_CMD::A()
{
  int  valReturnA = PCL_VOTER_FAIL;

    try
    {
      valReturnA =  (Discrete::On == (NMPS_PCLO_PD3_OP.getValue()));
    }

    catch(...)
    {
      valReturnA = PCL_VOTER_FAIL;
    }

  return valReturnA;
}




// COMMAND DATA - ON

int MPS_PCL_PD3_OP_CMD::B()
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
{
  int  valReturnB = PCL_VOTER_FAIL;

    try
    {
      valReturnB =  (Discrete::On == dsVar);
    }

    catch(...)
    {
      valReturnB = PCL_VOTER_FAIL;
    }

  return valReturnB;
}



// COMMAND DATA - OFF

int MPS_PCL_PD3_OP_CMD::C()
{
  int  valReturnC = PCL_VOTER_FAIL;

    try
    {
      valReturnC =  (Discrete::Off == dsVar);
    }

    catch(...)
    {
      valReturnC = PCL_VOTER_FAIL;
    }

  return valReturnC;
}




//PD3 - LH2 ORB/ET Recirc Disc Valve - CLOSED

int MPS_PCL_PD3_OP_CMD::D()
{
  int  valReturnD = PCL_VOTER_FAIL;

try
{
            if (NMPSPD3INDSTATE.getValue() == MPS_VALVE_IND_STATE::CLOSED)
     {
              valReturnD = PCL_VOTER_PASS;
             }
else
            {
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
      valReturnD = PCL_VOTER_FAIL;
          }
}

catch(...)
{
    valReturnD = PCL_VOTER_FAIL;
}

 return valReturnD;
}
```

MPS_PCL_PD3_OP_CMD mps_pcl_pd3_op_cmd(seqName);

3.4.25. MPS_PCL_PV8_CL_CMD

3.4.25.1. Description

This PCL provides protection for the MPS LH2 Feedline Relief Isolation Valve Close Commands.
This class inherits from the class CAS_Control_Logic. The Algorithm will be implemented directly in C++ Code.

Reference - LPS Program GCH33

3.4.25.1.1. Instance

Instance Name
MPS_PCL_PV8_CL_CMD

3.4.25.1.2. PCL Initiating Commands

Command
V41K1447XL
V41K1448XL
V41K1450XL

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.4.25.1.3. Operations Index

Operation	Description
determineState()	This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE INHIBITED ERROR
activate()	This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.
inhibit()	This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.
getState()	This method is inherited from the base class - CAS_Control_Logic
resetCLNotification()	This method is inherited from the base class - CAS_Control_Logic

3.4.25.2. Operations

3.4.25.2.1. getState()

3.4.25.2.1.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.25.2.2. resetCLNotification()

3.4.25.2.2.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.25.2.3. determineState()

3.4.25.2.3.1. Descriptive Text

This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE, INHIBITED or ERROR.

3.4.25.2.3.2. Sequence

```
if  (<V41K1447XL>.PCLSTAT = active  AND
    <V41K1448XL>.PCLSTAT = active  AND
    <V41K1450XL>.PCLSTAT = active)
  {
  CLState = ACTIVE;
  }
else
  {
   if  (<V41K1447XL>.PCLSTAT = inhibited AND
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
    <V41K1448XL>.PCLSTAT = inhibited  AND
    <V41K1450XL>.PCLSTAT = inhibited)
  {
  CLState = INHIBITED;
   }
 else
   {
  CLState = ERROR;
   }
}
```

3.4.25.2.4. activate()

3.4.25.2.4.1. Descriptive Text

This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.25.2.4.2. Sequence

A FD <V41K1447XL> PCL;
A FD <V41K1448XL> PCL;
A FD <V41K1450XL> PCL;

return;

3.4.25.2.5. inhibit()

3.4.25.2.5.1. Descriptive Text

This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.25.2.5.2. Sequence

I FD <V41K1447XL> PCL;
I FD <V41K1448XL> PCL;
I FD <V41K1450XL> PCL;

return;

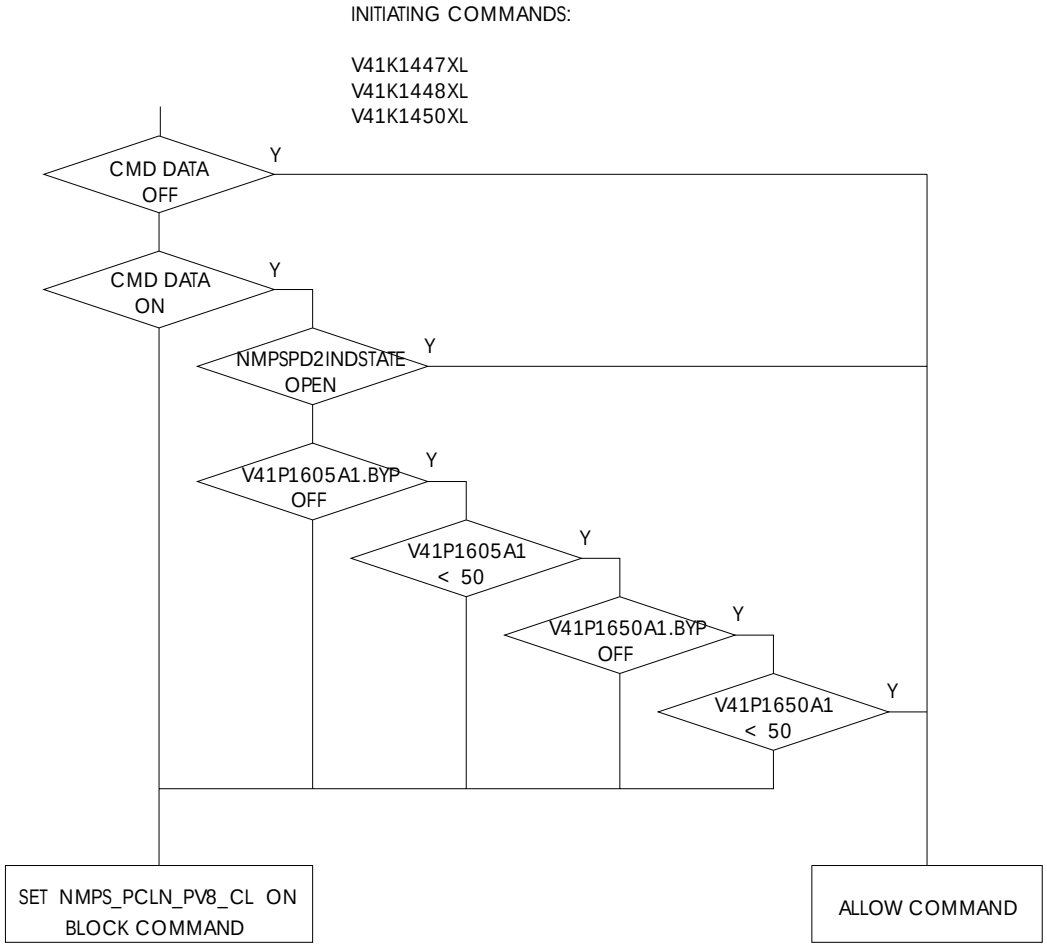
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.4.25.3. Algorithm

3.4.25.3.1. PCL Algorithm

3.4.25.3.1.1. Flowchart

MPS_PCL_PV8_CL_CMD



3.4.25.3.1.2. Sequence

```
#include "mps_pcl_pv8_cl_cmd.h"
#include "SME.h"
#include "LO2.h"
#include "MPS.h"
#include "LH2.h"

static const char* seqName = "MPS_PCL_PV8_CL_CMD";
static const char* algorithmDescription = "Revision Date - 01/20/2000\n\
if (A  || B || (C && (D || (E && F))))\n\
where:   A  = NMPS_PCLO_PV8_CL - PCL Bypass ON\n\
         B  = COMMAND DATA - OFF\n\
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
        C  = COMMAND DATA - ON\n\
        D  = PD2 - LH2 ORB/ET FEED DISC VALVE - OPEN\n\
        E  = MPS Pneu Reg Press < 50 psia\n\
        F  = MPS Pneu Accum Press < 50 psia";

MPS_PCL_PV8_CL_CMD::MPS_PCL_PV8_CL_CMD(const char*
seqName):asv_pcsVPCL_DS_ON_OFF_(seqName)
{
  userInit();
}

MPS_PCL_PV8_CL_CMD::~MPS_PCL_PV8_CL_CMD()
{
}


void MPS_PCL_PV8_CL_CMD::userInit(void)
{
  initiatingFD(V41K1447XL);
  initiatingFD(V41K1448XL);
  initiatingFD(V41K1450XL);

}

int MPS_PCL_PV8_CL_CMD::userExecute(Discrete::ON_OFF dsVar)
{
  if (A() || B() || (C() && D()))
  {
   return PCL_ALLOW_CMD;
  }
  else
  {
   NMPS_PCLN_PV8_CL.Set(Discrete::On);
   return PCL_BLOCK_CMD;
  }
}



// Function that determines the state of the indicator bypasses.

int MPS_PCL_PV8_CL_CMD::indicatorBypass(PD_ &pseudo_ind)
{
  if ( Discrete::On == pseudo_ind.getValue())
   return IND_BYPASS_ON;
 else
   return IND_BYPASS_OFF;
}




// PCL Override ON
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
int MPS_PCL_PV8_CL_CMD::A()
{
  int  valReturnA = PCL_VOTER_FAIL;

    try
    {
      valReturnA =  (Discrete::On == (NMPS_PCLO_PV8_CL.getValue()));
    }

    catch(...)
    {
      valReturnA = PCL_VOTER_FAIL;
    }

  return valReturnA;
}



// COMMAND DATA - OFF

int MPS_PCL_PV8_CL_CMD::B()
{
  int  valReturnB = PCL_VOTER_FAIL;

    try
    {
      valReturnB =  (Discrete::Off == dsVar);
    }

    catch(...)
    {
      valReturnB = PCL_VOTER_FAIL;
    }

  return valReturnB;
}



// COMMAND DATA - On

int MPS_PCL_PV8_CL_CMD::C()
{
  int  valReturnC = PCL_VOTER_FAIL;

    try
    {
      valReturnC =  (Discrete::On == dsVar);
    }
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
   catch(...)
   {
    valReturnC = PCL_VOTER_FAIL;
   }

 return valReturnC;
}



//PD2 - LH2 ORB/ET Feed Disc Valve - OPEN

int MPS_PCL_PV8_CL_CMD::D()
{
 int  valReturnD = PCL_VOTER_FAIL;

try
{
            if (NMPSPD2INDSTATE.getValue() == MPS_VALVE_IND_STATE::OPEN)
            {
              valReturnD = PCL_VOTER_PASS;
             }
else
            {
       valReturnD = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnD = PCL_VOTER_FAIL;
}

 return valReturnD;
}




// MPS Pneu Reg Press < 50

int MPS_PCL_PV8_CL_CMD::E()
{
 int  valReturnE = PCL_VOTER_FAIL;

 if (indicatorBypass(N41P1605A1))
 {
   try
   {
    valReturnE =  (V41P1605A1.getValue() < 50);
   }
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
   catch(...)
   {
    valReturnE = PCL_VOTER_FAIL;
   }
  }

  else
  {
   valReturnE = PCL_VOTER_FAIL;
  }

  return valReturnE;
}




// MPS Pneu Accum Press < 50

int MPS_PCL_PV8_CL_CMD::F()
{
  int  valReturnF = PCL_VOTER_FAIL;

  if (indicatorBypass(N41P1650A1))
  {
   try
   {
    valReturnF =  (V41P1650A1.getValue() < 50);
   }

   catch(...)
   {
    valReturnF = PCL_VOTER_FAIL;
   }
  }

  else
  {
   valReturnF = PCL_VOTER_FAIL;
  }

  return valReturnF;
}



MPS_PCL_PV8_CL_CMD mps_pcl_pv8_cl_cmd(seqName);
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.4.26. MPS_PCL_LV74_OP_CMD

3.4.26.1. Description

This PCL provides protection for the MPS LH2 RTLS Repress SOV #1.
This class inherits from the class CAS_Control_Logic. The Algorithm will be implemented directly in C++ Code.

Reference - LPS Program GCH34

3.4.26.1.1. Instance

Instance Name
MPS_PCL_LV74_OP_CMD

3.4.26.1.2. PCL Initiating Commands

Command
V41K1905XL
V41K1907XL
V41K1909XL

3.4.26.1.3. Operations Index

Operation	**Description**
determineState()	This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE INHIBITED ERROR
activate()	This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.
inhibit()	This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.
getState()	This method is inherited from the base class - CAS_Control_Logic
resetCLNotification()	This method is inherited from the base class - CAS_Control_Logic

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.4.26.2. Operations

3.4.26.2.1. getState()

3.4.26.2.1.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.26.2.2. resetCLNotification()

3.4.26.2.2.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.26.2.3. determineState()

3.4.26.2.3.1. Descriptive Text

This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE, INHIBITED or ERROR.

3.4.26.2.3.2. Sequence

```
if  (<V41K1905XL>.PCLSTAT = active  AND
     <V41K1907XL>.PCLSTAT = active  AND
     <V41K1909XL>.PCLSTAT = active)
  {
  CLState = ACTIVE;
  }
else
  {
   if  (<V41K1905XL>.PCLSTAT = inhibited AND
        <V41K1907XL>.PCLSTAT = inhibited  AND
        <V41K1909XL>.PCLSTAT = inhibited)
     {
     CLState = INHIBITED;
      }
    else
      {
     CLState = ERROR;
      }
   }
```

3.4.26.2.4. activate()

3.4.26.2.4.1. Descriptive Text

This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.26.2.4.2. Sequence

```
A FD <V41K1905XL> PCL;
A FD <V41K1907XL> PCL;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

A FD <V41K1909XL> PCL;

return;

3.4.26.2.5. inhibit()

3.4.26.2.5.1. Descriptive Text

This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.26.2.5.2. Sequence

I FD <V41K1905XL> PCL;
I FD <V41K1907XL> PCL;
I FD <V41K1909XL> PCL;

return;

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

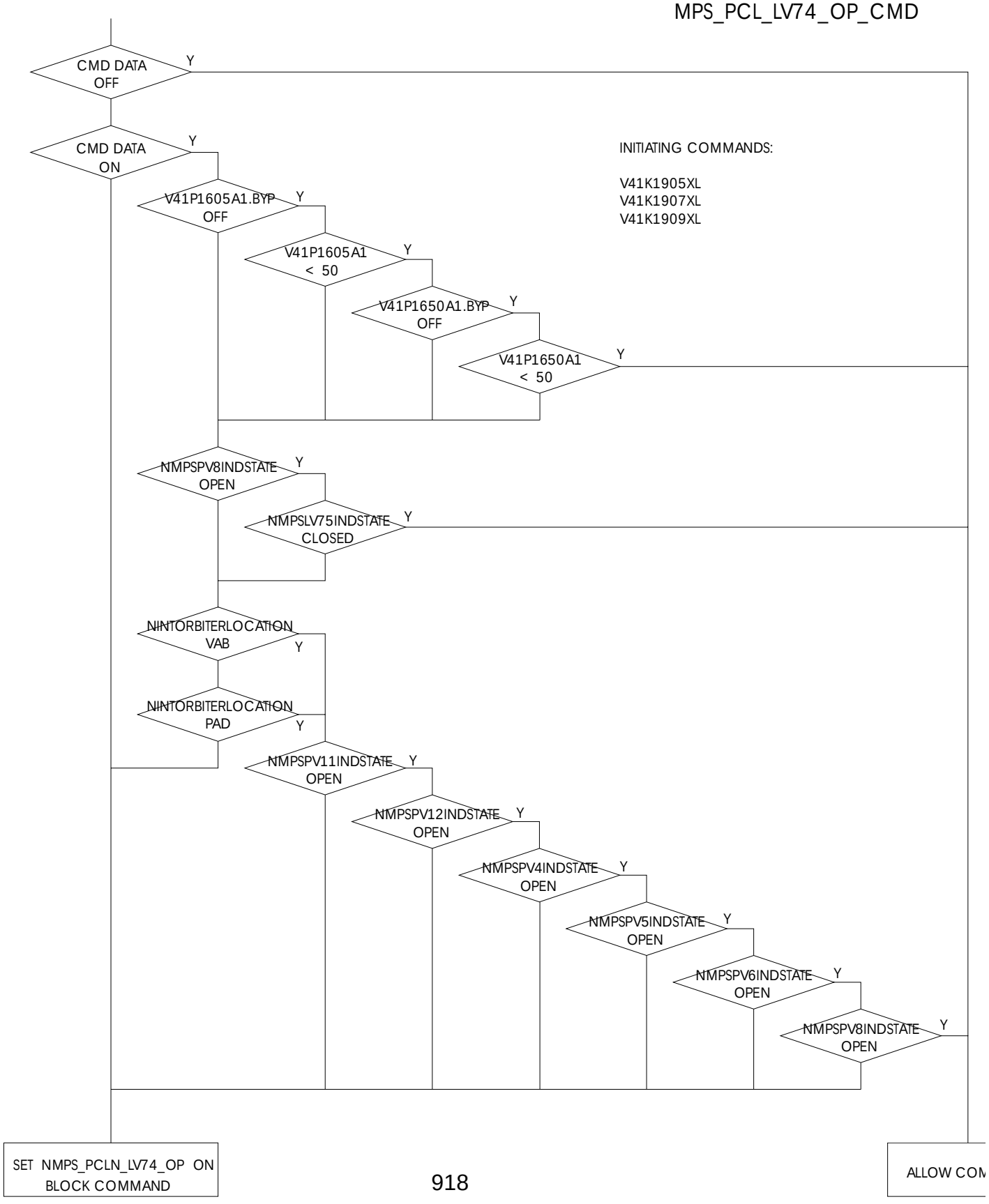
3.4.26.3. Algorithm

3.4.26.3.1. PCL Algorithm

3.4.26.3.1.1. Flowchart

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

MPS_PCL_LV74_OP_CMD



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.4.26.3.1.2. Sequence

```
#include "mps_pcl_lv74_op_cmd.h"
#include "SME.h"
#include "LO2.h"
#include "MPS.h"
#include "LH2.h"

static const char* seqName = "MPS_PCL_LV74_OP_CMD";
static const char* algorithmDescription = "Revision Date - 01/20/2000\n\
if (A  || B || (C && (D && E) || (F && G) || ((H || I) && J && K && L && M && N)))\n\
where:   A  = NMPS_PCLO_LV74_OP - PCL Bypass ON\n\
         B  = COMMAND DATA - OFF\n\
         C  = COMMAND DATA - ON\n\
         D  = MPS Pneu Reg Press < 50 psia\n\
         E  = MPS Pneu Accum Press < 50 psia\n\
         F  = PV8 - LH2 FDLN RELIEF SOV - OPEN\n\
         G  = LV75 - LH2 RTLS REPRESS SOV #2 - CLOSED\n\
         H  = NO3IS079E - ORB LOCATED AT VAB ON\n\
         I  = NO3IS080E - ORB LOCATED AT PAD ON\n\
         J  = PV12 - LH2 INBOARD F/D VALVE - OPEN\n\
         K  = PV11 -  LH2 OUTBOARD F/D VALVE - OPEN\n\
         L  = PV4 - LH2 E1 PREVALVE - OPEN\n\
         M  = PV5 - LH2 E2 PREVALVE - OPEN\n\
         N  = PV6 - LH2 E3 PREVALVE - OPEN";

MPS_PCL_LV74_OP_CMD::MPS_PCL_LV74_OP_CMD(const char*
seqName):asv_pcsVPCL_DS_ON_OFF_(seqName)
{
  userInit();
}

MPS_PCL_LV74_OP_CMD::~MPS_PCL_LV74_OP_CMD()
{
}


void MPS_PCL_LV74_OP_CMD::userInit(void)
{
  initiatingFD(V41K1905XL);
  initiatingFD(V41K1906XL);
  initiatingFD(V41K1907XL);

}

int MPS_PCL_LV74_OP_CMD::userExecute(Discrete::ON_OFF dsVar)
{

if (A()  || B() || (C() && (D() && E()) || (F() && G()) || ((H() || I()) && J() && K()) && L() && M() && N())))
    {
   return PCL_ALLOW_CMD;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
  }
  else
  {
   NMPS_PCLN_LV74_OP.Set(Discrete::On);
   return PCL_BLOCK_CMD;
  }
}



// Function that determines the state of the indicator bypasses.

int MPS_PCL_LV74_OP_CMD::indicatorBypass(PD_ &pseudo_ind)
{
  if ( Discrete::On == pseudo_ind.getValue())
   return IND_BYPASS_ON;
 else
   return IND_BYPASS_OFF;
}




// PCL Override ON

int MPS_PCL_LV74_OP_CMD::A()
{
  int  valReturnA = PCL_VOTER_FAIL;

    try
    {
      valReturnA =  (Discrete::On == (NMPS_PCLO_LV74_OP.getValue())) ;
    }

    catch(...)
    {
      valReturnA = PCL_VOTER_FAIL;
    }

  return valReturnA;
}




// COMMAND DATA - OFF

int MPS_PCL_LV74_OP_CMD::B()
{
  int  valReturnB = PCL_VOTER_FAIL;

    try
    {
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
    valReturnB =  (Discrete::Off == dsVar);
   }

   catch(...)
   {
    valReturnB = PCL_VOTER_FAIL;
   }

 return valReturnB;
}



// COMMAND DATA - On

int MPS_PCL_LV74_OP_CMD::C()
{
 int  valReturnC = PCL_VOTER_FAIL;

   try
   {
    valReturnC =  (Discrete::On == dsVar);
   }

   catch(...)
   {
    valReturnC = PCL_VOTER_FAIL;
   }

 return valReturnC;
}




// MPS Pneu Reg Press < 50

int MPS_PCL_LV74_OP_CMD::D()
{
 int  valReturnD = PCL_VOTER_FAIL;

 if (indicatorBypass(N41P1605A1))
 {
   try
   {
    valReturnD =  (V41P1605A1.getValue() < 50);
   }

   catch(...)
   {
    valReturnD = PCL_VOTER_FAIL;
   }
 }
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
  else
  {
    valReturnD = PCL_VOTER_FAIL;
  }

  return valReturnD;
}




// MPS Pneu Accum Press < 50

int MPS_PCL_LV74_OP_CMD::E()
{
  int  valReturnE = PCL_VOTER_FAIL;

  if (indicatorBypass(N41P1650A1))
  {
    try
    {
      valReturnE =  (V41P1650A1.getValue() < 50);
    }

    catch(...)
    {
      valReturnE = PCL_VOTER_FAIL;
    }
  }

  else
  {
    valReturnE = PCL_VOTER_FAIL;
  }

  return valReturnE;
}




//PV8 - LH2 FDLN Relief SOV - OPEN

int MPS_PCL_LV74_OP_CMD::F()
{
  int  valReturnF = PCL_VOTER_FAIL;

try
{
            if (NMPSPV8INDSTATE.getValue() == MPS_VALVE_IND_STATE::OPEN)
     {
              valReturnF = PCL_VOTER_PASS;
             }
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
else
            {
       valReturnF = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnF = PCL_VOTER_FAIL;
}

  return valReturnF;
}




//LV75 - LH2 RTLS REPRESS SOV #2 - CLOSED

int MPS_PCL_LV74_OP_CMD::G()
{
  int  valReturnG = PCL_VOTER_FAIL;

try
{
            if (NMPSLV74INDSTATE.getValue() == MPS_VALVE_IND_STATE::CLOSED)
     {
              valReturnG = PCL_VOTER_PASS;
             }
else
            {
       valReturnG = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnG = PCL_VOTER_FAIL;
}

  return valReturnG;
}




// ORB Location = VAB

int MPS_PCL_LV74_OP_CMD::H()
{
  int  valReturnH = PCL_VOTER_FAIL;

    try
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
    {
      valReturnH = (Discrete::On == (N03IS079E.getValue()));
    }

    catch(...)
    {
      valReturnH = PCL_VOTER_FAIL;
    }

  return valReturnH;
}




// ORB Location = PAD

int MPS_PCL_LV74_OP_CMD::I()
{
  int  valReturnI = PCL_VOTER_FAIL;

    try
    {
      valReturnI =    (Discrete::On == (N03IS080E.getValue()));
    }

    catch(...)
    {
      valReturnI = PCL_VOTER_FAIL;
    }

  return valReturnI;
}




// PV12 - LH2 Inboard F/D Valve - OPEN

int MPS_PCL_LV74_OP_CMD::J()
{
  int  valReturnJ = PCL_VOTER_FAIL;

try
{
            if (NMPSPV12INDSTATE.getValue() == MPS_VALVE_IND_STATE::OPEN)
     {
              valReturnJ = PCL_VOTER_PASS;
             }
else
            {
       valReturnJ = PCL_VOTER_FAIL;
            }
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
}

catch(...)
{
    valReturnJ = PCL_VOTER_FAIL;
}

 return valReturnJ;
}



//PV11 - LH2 Outboard F/D Valve - OPEN

int MPS_PCL_LV74_OP_CMD::K()
{
 int  valReturnK = PCL_VOTER_FAIL;

try
{
          if (NMPSPV11INDSTATE.getValue() == MPS_VALVE_IND_STATE::OPEN)
    {
            valReturnK = PCL_VOTER_PASS;
           }
else
          {
      valReturnK = PCL_VOTER_FAIL;
          }
}

catch(...)
{
    valReturnK = PCL_VOTER_FAIL;
}

 return valReturnK;
}



//PV4 - E#1LH2 PREVALVE - OPEN

int MPS_PCL_LV74_OP_CMD::L()
{
 int  valReturnL = PCL_VOTER_FAIL;

try
{
          if (NMPSPV4INDSTATE.getValue() == MPS_VALVE_IND_STATE::OPEN)
    {
            valReturnL = PCL_VOTER_PASS;
           }
else
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
            {
       valReturnL = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnL = PCL_VOTER_FAIL;
}

  return valReturnL;
}



//PV5 - E#2 LH2 PREVALVE - OPEN

int MPS_PCL_LV74_OP_CMD::M()
{
  int  valReturnM = PCL_VOTER_FAIL;

try
{
            if (NMPSPV5INDSTATE.getValue() == MPS_VALVE_IND_STATE::OPEN)
     {
              valReturnM = PCL_VOTER_PASS;
             }
else
            {
       valReturnM = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnM = PCL_VOTER_FAIL;
}

  return valReturnB;
}



//PV6 - E#3 LH2 PREVALVE - OPEN

int MPS_PCL_LV74_OP_CMD::N()
{
  int  valReturnN = PCL_VOTER_FAIL;

try
{
            if (NMPSPV6INDSTATE.getValue() == MPS_VALVE_IND_STATE::OPEN)
     {
              valReturnN = PCL_VOTER_PASS;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
            }
else
            {
       valReturnN = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnN = PCL_VOTER_FAIL;
}

  return valReturnN;
}
```

MPS_PCL_LV74_OP_CMD mps_pcl_lv74_op_cmd(seqName);

3.4.27. MPS_PCL_LV75_OP_CMD

3.4.27.1. Description

This PCL provides protection for the MPS LH2 RTLS Repress SOV#2.
This class inherits from the class CAS_Control_Logic. The Algorithm will be implemented directly in C++ Code.

Reference - LPS Program GCH35

3.4.27.1.1. Instance

Instance Name
MPS_PCL_LV75_OP_CMD

3.4.27.1.2. PCL Initiating Commands

Command
V41K1906XL
V41K1908XL

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

V41K1910XL

3.4.27.1.3. Operations Index

Operation	Description
determineState()	This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE INHIBITED ERROR
activate()	This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.
inhibit()	This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.
getState()	This method is inherited from the base class - CAS_Control_Logic
resetCLNotification()	This method is inherited from the base class - CAS_Control_Logic

3.4.27.2. Operations

3.4.27.2.1. getState()

3.4.27.2.1.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.27.2.2. resetCLNotification()

3.4.27.2.2.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.27.2.3. determineState()

3.4.27.2.3.1. Descriptive Text

This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE, INHIBITED or ERROR.

3.4.27.2.3.2. Sequence

```
if  (<V41K1906XL>.PCLSTAT = active  AND
    <V41K1908XL>.PCLSTAT = active  AND
    <V41K1910XL>.PCLSTAT = active)
 {
 CLState = ACTIVE;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
 }
else
 {
  if  (<V41K1906XL>.PCLSTAT = inhibited AND
       <V41K1908XL>.PCLSTAT = inhibited  AND
       <V41K1910XL>.PCLSTAT = inhibited)
    {
    CLState = INHIBITED;
     }
   else
     {
    CLState = ERROR;
     }
  }
```

3.4.27.2.4. activate()

3.4.27.2.4.1. Descriptive Text

This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.27.2.4.2. Sequence

```
A FD <V41K1906XL> PCL;
A FD <V41K1908XL> PCL;
A FD <V41K1910XL> PCL;

return;
```

3.4.27.2.5. inhibit()

3.4.27.2.5.1. Descriptive Text

This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.27.2.5.2. Sequence

```
I FD <V41K1906XL> PCL;
I FD <V41K1908XL> PCL;
I FD <V41K1910XL> PCL;

return;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

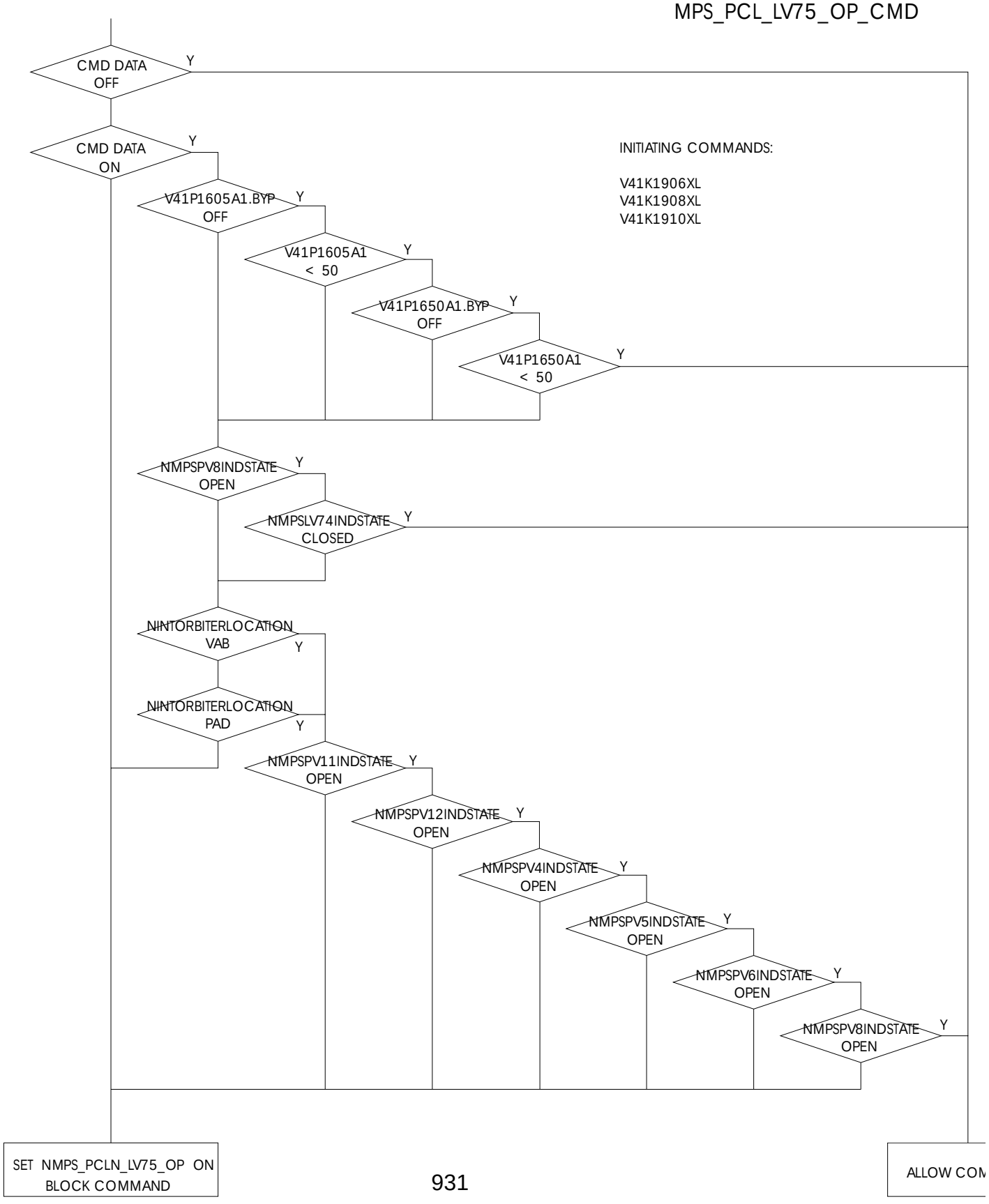
3.4.27.3. Algorithm

3.4.27.3.1. PCL Algorithm

3.4.27.3.1.1. Flowchart

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

MPS_PCL_LV75_OP_CMD



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.4.27.3.1.2. Sequence

```
#include "mps_pcl_lv75_op_cmd.h"
#include "SME.h"
#include "LO2.h"
#include "MPS.h"
#include "LH2.h"

static const char* seqName = "MPS_PCL_LV75_OP_CMD";
static const char* algorithmDescription = "Revision Date - 01/20/2000\n\
if (A  || B || (C && (D && E) || (F && G) || ((H || I) && J && K && L && M && N)))\n\
where:   A  = NMPS_PCLO_LV75_OP - PCL Bypass ON\n\
         B  = COMMAND DATA - OFF\n\
         C  = COMMAND DATA - ON\n\
         D  = MPS Pneu Reg Press < 50 psia\n\
         E  = MPS Pneu Accum Press < 50 psia\n\
         F  = PV8 - LH2 FDLN RELIEF SOV - OPEN\n\
         G  = LV74 - LH2 RTLS REPRESS SOV #1 - CLOSED\n\
         H  = NO3IS079E - ORB LOCATED AT VAB ON\n\
         I  = NO3IS080E - ORB LOCATED AT PAD ON\n\
         J  = PV12 - LH2 INBOARD F/D VALVE - OPEN\n\
         K  = PV11 -  LH2 OUTBOARD F/D VALVE - OPEN\n\
         L  = PV4 - LH2 E1 PREVALVE - OPEN\n\
         M  = PV5 - LH2 E2 PREVALVE - OPEN\n\
         N  = PV6 - LH2 E3 PREVALVE - OPEN";

MPS_PCL_LV75_OP_CMD::MPS_PCL_LV75_OP_CMD(const char*
seqName):asv_pcsVPCL_DS_ON_OFF_(seqName)
{
  userInit();
}

MPS_PCL_LV75_OP_CMD::~MPS_PCL_LV75_OP_CMD()
{
}


void MPS_PCL_LV75_OP_CMD::userInit(void)
{
  initiatingFD(V41K1906XL);
  initiatingFD(V41K1908XL);
  initiatingFD(V41K1910XL);

}

int MPS_PCL_LV75_OP_CMD::userExecute(Discrete::ON_OFF dsVar)
{

if (A()  || B() || (C() && (D() && E()) || (F() && G()) || ((H() || I()) && J() && K()) && L() && M() && N())))
    {
   return PCL_ALLOW_CMD;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
  }
  else
  {
   NMPS_PCLN_LV75_OP.Set(Discrete::On);
   return PCL_BLOCK_CMD;
  }
}



// Function that determines the state of the indicator bypasses.

int MPS_PCL_LV75_OP_CMD::indicatorBypass(PD_ &pseudo_ind)
{
  if ( Discrete::On == pseudo_ind.getValue())
   return IND_BYPASS_ON;
 else
   return IND_BYPASS_OFF;
}




// PCL Override ON

int MPS_PCL_LV75_OP_CMD::A()
{
  int  valReturnA = PCL_VOTER_FAIL;

    try
    {
      valReturnA =  (Discrete::On == (NMPS_PCLO_LV75_OP.getValue()));
    }

    catch(...)
    {
      valReturnA = PCL_VOTER_FAIL;
    }

  return valReturnA;
}




// COMMAND DATA - OFF

int MPS_PCL_LV75_OP_CMD::B()
{
  int  valReturnB = PCL_VOTER_FAIL;

    try
    {
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
    valReturnB =  (Discrete::Off == dsVar);
   }

   catch(...)
   {
    valReturnB = PCL_VOTER_FAIL;
   }

 return valReturnB;
}



// COMMAND DATA - On

int MPS_PCL_LV75_OP_CMD::C()
{
 int  valReturnC = PCL_VOTER_FAIL;

   try
   {
    valReturnC =  (Discrete::On == dsVar);
   }

   catch(...)
   {
    valReturnC = PCL_VOTER_FAIL;
   }

 return valReturnC;
}




// MPS Pneu Reg Press < 50

int MPS_PCL_LV75_OP_CMD::D()
{
 int  valReturnD = PCL_VOTER_FAIL;

 if (indicatorBypass(N41P1605A1))
 {
   try
   {
    valReturnD =  (V41P1605A1.getValue() < 50);
   }

   catch(...)
   {
    valReturnD = PCL_VOTER_FAIL;
   }
 }
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
  else
  {
    valReturnD = PCL_VOTER_FAIL;
  }

  return valReturnD;
}




// MPS Pneu Accum Press < 50

int MPS_PCL_LV75_OP_CMD::E()
{
  int  valReturnE = PCL_VOTER_FAIL;

  if (indicatorBypass(N41P1650A1))
  {
    try
    {
      valReturnE =  (V41P1650A1.getValue() < 50);
    }

    catch(...)
    {
      valReturnE = PCL_VOTER_FAIL;
    }
  }

  else
  {
    valReturnE = PCL_VOTER_FAIL;
  }

  return valReturnE;
}




//PV8 - LH2 FDLN Relief SOV - OPEN

int MPS_PCL_LV75_OP_CMD::F()
{
  int  valReturnF = PCL_VOTER_FAIL;

try
{
            if (NMPSPV8INDSTATE.getValue() == MPS_VALVE_IND_STATE::OPEN)
     {
              valReturnF = PCL_VOTER_PASS;
             }
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
else
            {
       valReturnF = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnF = PCL_VOTER_FAIL;
}

  return valReturnF;
}




//LV74 - LH2 RTLS REPRESS SOV #1 - CLOSED

int MPS_PCL_LV75_OP_CMD::G()
{
  int  valReturnG = PCL_VOTER_FAIL;

try
{
            if (NMPSLV74INDSTATE.getValue() == MPS_VALVE_IND_STATE::CLOSED)
     {
              valReturnG = PCL_VOTER_PASS;
             }
else
            {
       valReturnG = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnG = PCL_VOTER_FAIL;
}

  return valReturnG;
}




// ORB Location = VAB

int MPS_PCL_LV75_OP_CMD::H()
{
  int  valReturnH = PCL_VOTER_FAIL;

    try
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
    {
      valReturnH = (Discrete::On == (N03IS079E.getValue()));
    }

    catch(...)
    {
      valReturnH = PCL_VOTER_FAIL;
    }

  return valReturnH;
}




// ORB Location = PAD

int MPS_PCL_LV75_OP_CMD::I()
{
  int  valReturnI = PCL_VOTER_FAIL;

    try
    {
      valReturnI =  (Discrete::On == (N03IS080E.getValue()));
    }

    catch(...)
    {
      valReturnI = PCL_VOTER_FAIL;
    }

  return valReturnI;
}




// PV12 - LH2 Inboard F/D Valve - OPEN

int MPS_PCL_LV75_OP_CMD::J()
{
  int  valReturnJ = PCL_VOTER_FAIL;

try
{
            if (NMPSPV12INDSTATE.getValue() == MPS_VALVE_IND_STATE::OPEN)
     {
              valReturnJ = PCL_VOTER_PASS;
             }
else
            {
       valReturnJ = PCL_VOTER_FAIL;
            }
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
}

catch(...)
{
    valReturnJ = PCL_VOTER_FAIL;
}

 return valReturnJ;
}



//PV11 - LH2 Outboard F/D Valve - OPEN

int MPS_PCL_LV75_OP_CMD::K()
{
 int  valReturnK = PCL_VOTER_FAIL;

try
{
          if (NMPSPV11INDSTATE.getValue() == MPS_VALVE_IND_STATE::OPEN)
    {
            valReturnK = PCL_VOTER_PASS;
           }
else
          {
      valReturnK = PCL_VOTER_FAIL;
          }
}

catch(...)
{
    valReturnK = PCL_VOTER_FAIL;
}

 return valReturnK;
}




//PV4 - E#1LH2 PREVALVE - OPEN

int MPS_PCL_LV75_OP_CMD::L()
{
 int  valReturnL = PCL_VOTER_FAIL;

try
{
          if (NMPSPV4INDSTATE.getValue() == MPS_VALVE_IND_STATE::OPEN)
    {
            valReturnL = PCL_VOTER_PASS;
           }
else
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
            {
       valReturnL = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnL = PCL_VOTER_FAIL;
}

  return valReturnL;
}



//PV5 - E#2 LH2 PREVALVE - OPEN

int MPS_PCL_LV75_OP_CMD::M()
{
  int  valReturnM = PCL_VOTER_FAIL;

try
{
            if (NMPSPV5INDSTATE.getValue() == MPS_VALVE_IND_STATE::OPEN)
     {
              valReturnM = PCL_VOTER_PASS;
             }
else
            {
       valReturnM = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnM = PCL_VOTER_FAIL;
}

  return valReturnB;
}



//PV6 - E#3 LH2 PREVALVE - OPEN

int MPS_PCL_LV75_OP_CMD::N()
{
  int  valReturnN = PCL_VOTER_FAIL;

try
{
            if (NMPSPV6INDSTATE.getValue() == MPS_VALVE_IND_STATE::OPEN)
     {
              valReturnN = PCL_VOTER_PASS;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
            }
else
            {
       valReturnN = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnN = PCL_VOTER_FAIL;
}

  return valReturnN;
}
```

MPS_PCL_LV75_OP_CMD mps_pcl_lv75_op_cmd(seqName);

3.4.28. MPS_PCL_RECIRC_OP_CMD

3.4.28.1. Description

This PCL provides protection for the MPS LH2 Recirc Valves.
This class inherits from the class CAS_Control_Logic. The Algorithm will be implemented directly in C++ Code.

Reference - LPS Program GCH36

3.4.28.1.1. Instance

Instance Name
MPS_PCL_RECIRC_OP_CMD

3.4.28.1.2. PCL Initiating Commands

Command
V41K1111NL

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.4.28.1.3. Operations Index

Operation	Description
determineState()	This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE INHIBITED ERROR
activate()	This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.
inhibit()	This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.
getState()	This method is inherited from the base class - CAS_Control_Logic
resetCLNotification()	This method is inherited from the base class - CAS_Control_Logic

3.4.28.2. Operations

3.4.28.2.1. getState()

3.4.28.2.1.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.28.2.2. resetCLNotification()

3.4.28.2.2.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.28.2.3. determineState()

3.4.28.2.3.1. Descriptive Text

This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE, INHIBITED or ERROR.

3.4.28.2.3.2. Sequence

```
if  (<V41K1111NL>.PCLSTAT = active)
  {
  CLState = ACTIVE;
  }
else
  {
   if  (<V41K1111NL>.PCLSTAT = inhibted)
     {
     CLState = INHIBITED;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
    }
  else
    {
   CLState = ERROR;
    }
 }
```

3.4.28.2.4. activate()

3.4.28.2.4.1. Descriptive Text

This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.28.2.4.2. Sequence

A FD <V41K1111NL> PCL;

return;

3.4.28.2.5. inhibit()

3.4.28.2.5.1. Descriptive Text

This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.28.2.5.2. Sequence

I FD <V41K1111NL> PCL;

return;

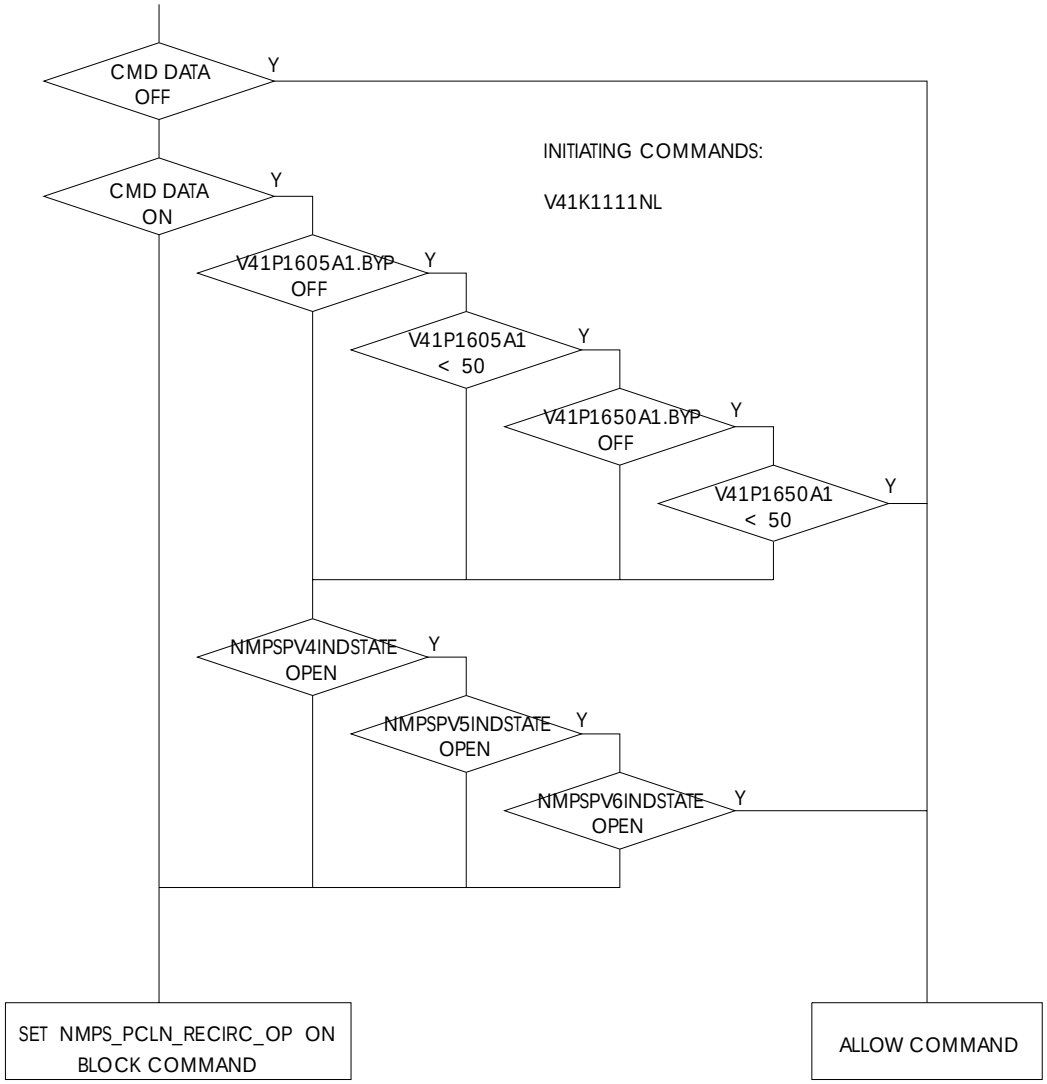
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.4.28.3. Algorithm

3.4.28.3.1. PCL Algorithm

3.4.28.3.1.1. Flowchart

MPS_PCL_RECIRC_OP_CMD



3.4.28.3.1.2. Sequence

```
#include "mps_pcl_recirc_op_cmd.h"
#include "SME.h"
#include "LO2.h"
#include "MPS.h"
#include "LH2.h"

static const char* seqName = "MPS_PCL_RECIRC_OP_CMD";
static const char* algorithmDescription = "Revision Date - 01/20/2000\n\
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
if (A  || B || (C && ((D && E && F) || (G && H))))\n\
where:   A  = NMPS_PCLO_RECIRC_OP - PCL Bypass ON\n\
         B  = COMMAND DATA - OFF\n\
         C  = COMMAND DATA - ON\n\
         D  = PV4 - LH2 E1 PREVALVE - OPEN\n\
         E  = PV5 - LH2 E2 PREVALVE - OPEN\n\
         F  = PV6 - LH2 E3 PREVALVE - OPEN\n\
         G  = MPS Pneu Reg Press < 50 psia\n\
         H  = MPS Pneu Accum Press < 50 psia";

MPS_PCL_RECIRC_OP_CMD::MPS_PCL_RECIRC_OP_CMD(const char*
seqName):asv_pcsVPCL_DS_ON_OFF_(seqName)
{
  userInit();
}

MPS_PCL_RECIRC_OP_CMD::~MPS_PCL_RECIRC_OP_CMD()
{
}


void MPS_PCL_RECIRC_OP_CMD::userInit(void)
{
  initiatingFD(V41K1111NL);
}

int MPS_PCL_RECIRC_OP_CMD::userExecute(Discrete::ON_OFF dsVar)
{

if (A()  || B() || (C() && ((D() && E() && F()) || (G() && H()))))
    {
   return PCL_ALLOW_CMD;
  }
  else
  {
   NMPS_PCLN_RECIRC_OP.Set(Discrete::On);
   return PCL_BLOCK_CMD;
  }
}



// Function that determines the state of the indicator bypasses.

int MPS_PCL_RECIRC_OP_CMD::indicatorBypass(PD_ &pseudo_ind)
{
  if ( Discrete::On == pseudo_ind.getValue())
   return IND_BYPASS_ON;
 else
   return IND_BYPASS_OFF;
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
// PCL Override ON

int MPS_PCL_RECIRC_OP_CMD::A()
{
  int  valReturnA = PCL_VOTER_FAIL;

    try
    {
      valReturnA =  (Discrete::On == (NMPS_PCLO_RECIRC_OP.getValue()));
    }

    catch(...)
    {
      valReturnA = PCL_VOTER_FAIL;
    }

  return valReturnA;
}



// COMMAND DATA - OFF

int MPS_PCL_RECIRC_OP_CMD::B()
{
  int  valReturnB = PCL_VOTER_FAIL;

    try
    {
      valReturnB =  (Discrete::Off == dsVar);
    }

    catch(...)
    {
      valReturnB = PCL_VOTER_FAIL;
    }

  return valReturnB;
}



// COMMAND DATA - On

int MPS_PCL_RECIRC_OP_CMD::C()
{
  int  valReturnC = PCL_VOTER_FAIL;

    try
    {
      valReturnC =  (Discrete::On == dsVar);
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
    }

   catch(...)
   {
    valReturnC = PCL_VOTER_FAIL;
   }

 return valReturnC;
}




//PV4 - E#1LH2 PREVALVE - OPEN

int MPS_PCL_RECIRC_OP_CMD::D()
{
 int  valReturnD = PCL_VOTER_FAIL;

try
{
            if (NMPSPV4INDSTATE.getValue() == MPS_VALVE_IND_STATE::OPEN)
     {
              valReturnD = PCL_VOTER_PASS;
             }
else
            {
       valReturnD = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnD = PCL_VOTER_FAIL;
}

 return valReturnD;
}



//PV5 - E#2 LH2 PREVALVE - OPEN

int MPS_PCL_RECIRC_OP_CMD::E()
{
 int  valReturnE = PCL_VOTER_FAIL;

try
{
            if (NMPSPV5INDSTATE.getValue() == MPS_VALVE_IND_STATE::OPEN)
     {
              valReturnE = PCL_VOTER_PASS;
             }
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
else
            {
       valReturnE = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnE = PCL_VOTER_FAIL;
}

  return valReturnE;
}



//PV6 - E#3 LH2 PREVALVE - OPEN

int MPS_PCL_RECIRC_OP_CMD::F()
{
  int  valReturnF = PCL_VOTER_FAIL;

try
{
            if (NMPSPV6INDSTATE.getValue() == MPS_VALVE_IND_STATE::OPEN)
     {
              valReturnF = PCL_VOTER_PASS;
             }
else
            {
       valReturnF = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnF = PCL_VOTER_FAIL;
}

  return valReturnF;
}



// MPS Pneu Reg Press < 50

int MPS_PCL_RECIRC_OP_CMD::G()
{
  int  valReturnG = PCL_VOTER_FAIL;

  if (indicatorBypass(N41P1605A1))
  {
    try
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
    {
     valReturnG =  (V41P1605A1.getValue() < 50);
    }

    catch(...)
    {
     valReturnG = PCL_VOTER_FAIL;
    }
  }

  else
  {
    valReturnG = PCL_VOTER_FAIL;
  }

  return valReturnG;
}




// MPS Pneu Accum Press < 50

int MPS_PCL_RECIRC_OP_CMD::H()
{
  int  valReturnH = PCL_VOTER_FAIL;

  if (indicatorBypass(N41P1650A1))
  {
    try
    {
     valReturnH =  (V41P1650A1.getValue() < 50);
    }

    catch(...)
    {
     valReturnH = PCL_VOTER_FAIL;
    }
  }

  else
  {
    valReturnH = PCL_VOTER_FAIL;
  }

  return valReturnH;
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

MPS_PCL_RECIRC_OP_CMD mps_pcl_recirc_op_cmd(seqName);

3.4.29. MPS_PCL_RTLS_DUMP_OP_CMD

3.4.29.1. Description

This PCL provides protection for the MPS LH2 RTLS DUMP Valves (PV17/PV18) Open Commands to prevent inadvertant opening of the valves during LH2 Load or Drain.
This class inherits from the class CAS_Control_Logic. The Algorithm will be implemented directly in C++ Code.

3.4.29.1.1. Instance

Instance Name
MPS_PCL_RTLS_DUMP_OP_CMD

3.4.29.1.2. PCL Initiating Commands

Command
V41K1923XL
V41K1924XL
V41K1925XL
V41K1913XL
V41K1914XL
V41K1915XL

3.4.29.1.3. Operations Index

Operation	**Description**
determineState()	This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE INHIBITED ERROR

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

activate()	This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.
inhibit()	This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.
getState()	This method is inherited from the base class - CAS_Control_Logic
resetCLNotification()	This method is inherited from the base class - CAS_Control_Logic

3.4.29.2. Operations

3.4.29.2.1. getState()

3.4.29.2.1.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.29.2.2. resetCLNotification()

3.4.29.2.2.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.29.2.3. determineState()

3.4.29.2.3.1. Descriptive Text

This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE, INHIBITED or ERROR.

3.4.29.2.3.2. Sequence

```
if  (<V41K1923XL>.PCLSTAT = active  AND
    <V41K1924XL>.PCLSTAT = active  AND
    <V41K1925XL>.PCLSTAT = active  AND
    <V41K1913XL>.PCLSTAT = active  AND
    <V41K1914XL>.PCLSTAT = active  AND
    <V41K1915XL>.PCLSTAT = active)
  {
  CLState = ACTIVE;
  }
else
  {
   if  (<V41K1923XL>.PCLSTAT = inhibited AND
       <V41K1924XL>.PCLSTAT = inhibited  AND
       <V41K1925XL>.PCLSTAT = inhibited  AND
       <V41K1913XL>.PCLSTAT = inhibited AND
       <V41K1914XL>.PCLSTAT = inhibited  AND
       <V41K1915XL>.PCLSTAT = inhibited)
    {
    CLState = INHIBITED;
     }
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
   else
     {
    CLState = ERROR;
     }
  }
```

3.4.29.2.4. activate()

3.4.29.2.4.1. Descriptive Text

This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.29.2.4.2. Sequence

A FD <V41K1923XL> PCL;
A FD <V41K1924XL> PCL;
A FD <V41K1925XL> PCL;
A FD <V41K1913XL> PCL;
A FD <V41K1914XL> PCL;
A FD <V41K1915XL> PCL;

return;

3.4.29.2.5. inhibit()

3.4.29.2.5.1. Descriptive Text

This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.29.2.5.2. Sequence

I FD <V41K1923XL> PCL;
I FD <V41K1924XL> PCL;
I FD <V41K1925XL> PCL;
I FD <V41K1913XL> PCL;
I FD <V41K1914XL> PCL;
I FD <V41K1915XL> PCL;

return;

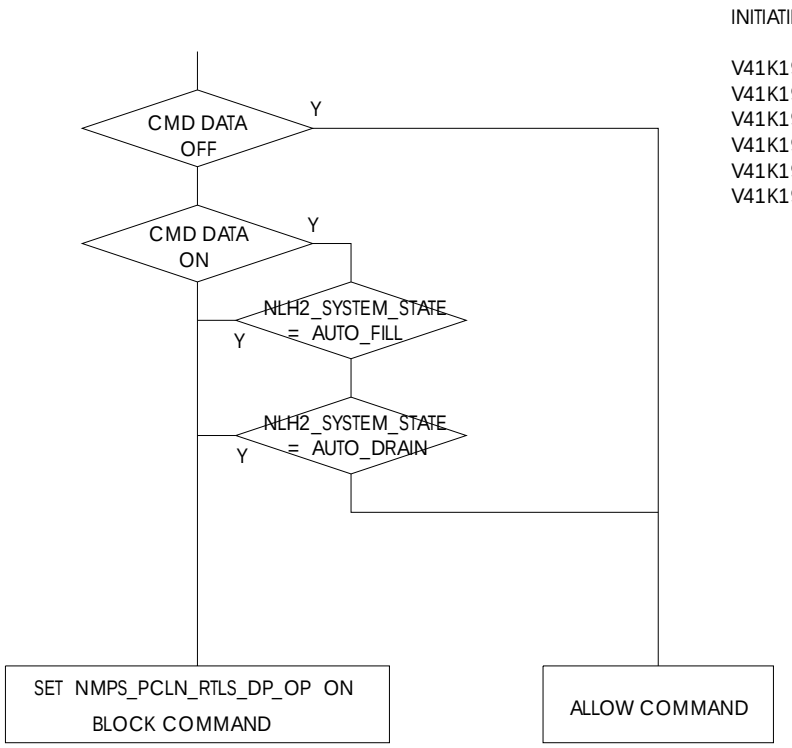
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.4.29.3. Algorithm

3.4.29.3.1. PCL Algorithm

3.4.29.3.1.1. Flowchart

MPS_PCL_RTLS_DUMP_OP_CMD



3.4.29.3.1.2. Sequence

```
#include "mps_pcl_rtls_dump_op_cmd.h"
#include "SME.h"
#include "LO2.h"
#include "MPS.h"
#include "LH2.h"

static const char* seqName = "MPS_PCL_RTLS_DUMP_OP_CMD";
static const char* algorithmDescription = "Revision Date - 01/20/2000\n\
if (A  || B || (C && !D && !E))\n\
where:   A  = NMPS_PCLO_RTLS_DP_OP - PCL Bypass ON\n\
         B  = COMMAND DATA - OFF\n\
         C  = COMMAND DATA - ON\n\
         D  = LH2 LOAD IN PROGRESS\n\
         E  = LH2 DRAIN IN PROGRESS";
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
MPS_PCL_RTLS_DUMP_OP_CMD::MPS_PCL_RTLS_DUMP_OP_CMD(const char*
seqName):asv_pcsVPCL_DS_ON_OFF_(seqName)
{
  userInit();
}

MPS_PCL_RTLS_DUMP_OP_CMD::~MPS_PCL_RTLS_DUMP_OP_CMD()
{
}


void MPS_PCL_RTLS_DUMP_OP_CMD::userInit(void)
{
  initiatingFD(V41K1923XL);
  initiatingFD(V41K1924XL);
  initiatingFD(V41K1925XL);
  initiatingFD(V41K1913XL);
  initiatingFD(V41K1914XL);
  initiatingFD(V41K1915XL);
}

int MPS_PCL_RTLS_DUMP_OP_CMD::userExecute(Discrete::ON_OFF dsVar)
{
  if (A()  || B() || (C() && !D() && !E()))
  {
   return PCL_ALLOW_CMD;
  }
  else
  {
   NMPS_PCLN_RTLS_DP_OP.Set(Discrete::On);
   return PCL_BLOCK_CMD;
  }
}




// PCL Override ON

int MPS_PCL_RTLS_DUMP_OP_CMD::A()
{
  int  valReturnA = PCL_VOTER_FAIL;

    try
    {
      valReturnA =   (Discrete::On == (NMPS_PCLO_RTLS_DP_OP.getValue()));
    }

    catch(...)
    {
      valReturnA = PCL_VOTER_FAIL;
    }
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
  return valReturnA;
}



// COMMAND DATA - OFF

int MPS_PCL_RTLS_DUMP_OP_CMD::B()
{
  int  valReturnB = PCL_VOTER_FAIL;

    try
    {
      valReturnB =  (Discrete::Off == dsVar);
    }

    catch(...)
    {
      valReturnB = PCL_VOTER_FAIL;
    }

  return valReturnB;
}



// COMMAND DATA - ON

int MPS_PCL_RTLS_DUMP_OP_CMD::C()
{
  int  valReturnC = PCL_VOTER_FAIL;

    try
    {
      valReturnC =  (Discrete::On == dsVar);
    }

    catch(...)
    {
      valReturnC = PCL_VOTER_FAIL;
    }

  return valReturnC;
}



// LH2 LOAD IN PROGRESS

int MPS_PCL_RTLS_DUMP_OP_CMD::D()
{
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
  int  valReturnD = PCL_VOTER_FAIL;

    try
    {
     valReturnD = (Discrete::On == (NLHK0012X.getValue()));
    }

    catch(...)
    {
     valReturnD = PCL_VOTER_FAIL;
    }

  return valReturnD;
}



// LH2 DRAIN IN PROGRESS

int MPS_PCL_RTLS_DUMP_OP_CMD::E()
{
  int  valReturnE = PCL_VOTER_FAIL;

    try
    {
     valReturnE = (Discrete::On == (NLHK0015X.getValue()));
    }

    catch(...)
    {
     valReturnE = PCL_VOTER_FAIL;
    }

  return valReturnE;
}



MPS_PCL_RTLS_DUMP_OP_CMD mps_pcl_rtls_dump_op_cmd(seqName);
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.4.30. MPS_PCL_PT_SENS_PWR_CMD

3.4.30.1. Description

This PCL provides protection for the MPS/ET Point Sensor Power On Commands to prevent inadvertant power off of the ET during LH2 Load or Drain.
This class inherits from the class CAS_Control_Logic. The Algorithm will be implemented directly in C++ Code.

3.4.30.1.1. Instance

Instance Name
MPS_PCL_PT_SENS_PWR_CMD

3.4.30.1.2. PCL Initiating Commands

Command
V41K0075XL
V41K0076XL
V41K0077XL

3.4.30.1.3. Operations Index

Operation	**Description**
determineState()	This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE INHIBITED ERROR
activate()	This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.
inhibit()	This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.
getState()	This method is inherited from the base class - CAS_Control_Logic
resetCLNotification()	This method is inherited from the base class - CAS_Control_Logic

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.4.30.2. Operations

3.4.30.2.1. getState()

3.4.30.2.1.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.30.2.2. resetCLNotification()

3.4.30.2.2.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.30.2.3. determineState()

3.4.30.2.3.1. Descriptive Text

This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE, INHIBITED or ERROR.

3.4.30.2.3.2. Sequence

```
if  (<V41K0075XL>.PCLSTAT = active  AND
     <V41K0076XL>.PCLSTAT = active  AND
     <V41K0077XL>.PCLSTAT = active)
  {
  CLState = ACTIVE;
  }
else
  {
   if  (<V41K0075XL>.PCLSTAT = inhibited AND
        <V41K0076XL>.PCLSTAT = inhibited  AND
        <V41K0077XL>.PCLSTAT = inhibited)
     {
     CLState = INHIBITED;
      }
    else
      {
     CLState = ERROR;
      }
   }
```

3.4.30.2.4. activate()

3.4.30.2.4.1. Descriptive Text

This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.30.2.4.2. Sequence

```
A FD <V41K0075XL> PCL;
A FD <V41K0076XL> PCL;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

A FD <V41K0077XL> PCL;

return;

3.4.30.2.5. inhibit()

3.4.30.2.5.1. Descriptive Text

This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.30.2.5.2. Sequence

I FD <V41K0075XL> PCL;
I FD <V41K0076XL> PCL;
I FD <V41K0077XL> PCL;

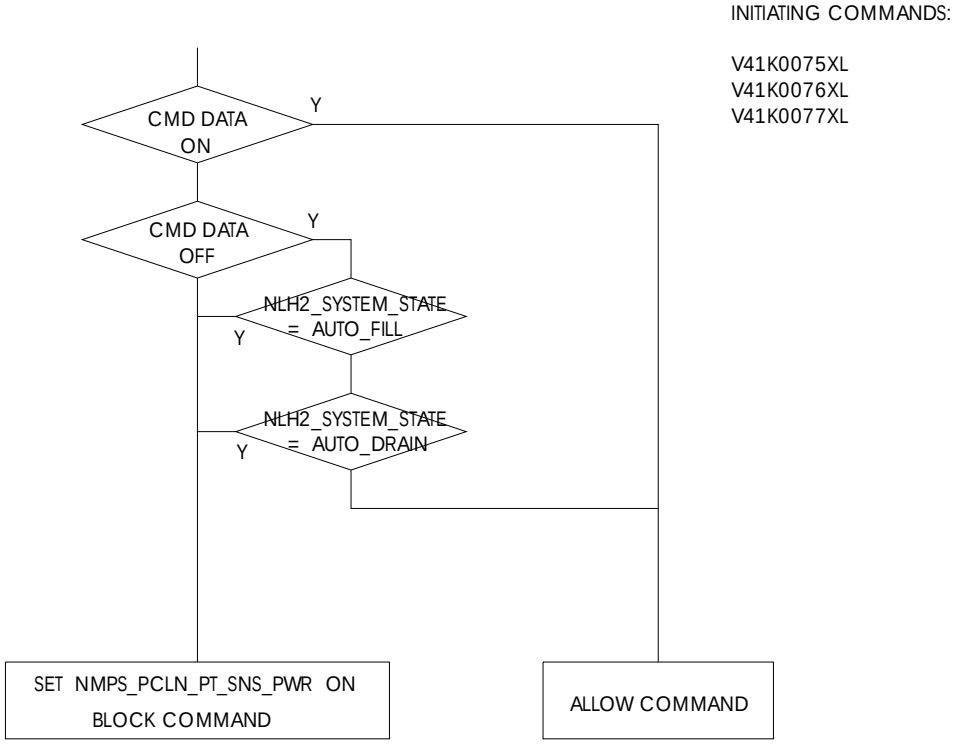
return;

3.4.30.3. Algorithm

3.4.30.3.1. PCL Algorithm

3.4.30.3.1.1. Flowchart

MPS_PCL_PT_SENS_PWR_CMD



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.4.30.3.1.2. Sequence

```
#include "mps_pcl_pt_sens_pwr_cmd.h"
#include "SME.h"
#include "LO2.h"
#include "MPS.h"
#include "LH2.h"

static const char* seqName = "MPS_PCL_PT_SENS_PWR_CMD";
static const char* algorithmDescription = "Revision Date - 01/20/2000\n\
if (A  || B || (C && !D && !E))\n\
where:   A  = NMPS_PCLO_PT_SNS_PWR - PCL Bypass ON\n\
         B  = COMMAND DATA - ON\n\
         C  = COMMAND DATA - OFF\n\
         D  = LH2 LOAD IN PROGRESS\n\
         E  = LH2 DRAIN IN PROGRESS";

MPS_PCL_PT_SENS_PWR_CMD::MPS_PCL_PT_SENS_PWR_CMD(const char*
seqName):asv_pcsVPCL_DS_ON_OFF_(seqName)
{
  userInit();
}

MPS_PCL_PT_SENS_PWR_CMD::~MPS_PCL_PT_SENS_PWR_CMD()
{
}


void MPS_PCL_PT_SENS_PWR_CMD::userInit(void)
{
  initiatingFD(V41K0075XL);
  initiatingFD(V41K0076XL);
  initiatingFD(V41K0077XL);
}

int MPS_PCL_PT_SENS_PWR_CMD::userExecute(Discrete::ON_OFF dsVar)
{
  if (A()  || B() || (C() && !D() && !E()))
  {
   return PCL_ALLOW_CMD;
  }
  else
  {
   NMPS_PCLN_PT_SNS_PWR.Set(Discrete::On);
   return PCL_BLOCK_CMD;
  }
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
// PCL Override ON

int MPS_PCL_PT_SENS_PWR_CMD::A()
{
  int  valReturnA = PCL_VOTER_FAIL;

    try
    {
      valReturnA =  (Discrete::On == (NMPS_PCLO_PT_SNS_PWR.getValue()));
    }

    catch(...)
    {
      valReturnA = PCL_VOTER_FAIL;
    }

  return valReturnA;
}



// COMMAND DATA - ON

int MPS_PCL_PT_SENS_PWR_CMD::B()
{
  int  valReturnB = PCL_VOTER_FAIL;

    try
    {
      valReturnB =  (Discrete::On == dsVar);
    }

    catch(...)
    {
      valReturnB = PCL_VOTER_FAIL;
    }

  return valReturnB;
}



// COMMAND DATA - OFF

int MPS_PCL_PT_SENS_PWR_CMD::C()
{
  int  valReturnC = PCL_VOTER_FAIL;

    try
    {
      valReturnC =  (Discrete::Off == dsVar);
    }
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
    catch(...)
    {
      valReturnC = PCL_VOTER_FAIL;
    }

  return valReturnC;
}



// LH2 LOAD IN PROGRESS

int MPS_PCL_PT_SENS_PWR_CMD::D()
{
  int  valReturnD = PCL_VOTER_FAIL;

    try
    {
      valReturnD =  (Discrete::On == (NLHK0012X.getValue()));
    }

    catch(...)
    {
      valReturnD = PCL_VOTER_FAIL;
    }

  return valReturnD;
}



// LH2 DRAIN IN PROGRESS

int MPS_PCL_PT_SENS_PWR_CMD::E()
{
  int  valReturnE = PCL_VOTER_FAIL;

    try
    {
      valReturnE =  (Discrete::On == (NLHK0015X.getValue()));
    }

    catch(...)
    {
      valReturnE = PCL_VOTER_FAIL;
    }

  return valReturnE;
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

MPS_PCL_PT_SENS_PWR_CMD mps_pcl_pt_sens_pwr_cmd(seqName);

3.4.31. MPS_PCL_FCV1_CO_CMD

3.4.31.1. Description

This PCL provides protection for the MPS LH2 FCV #1 Checkout HI Commands.
This class inherits from the class CAS_Control_Logic. The Algorithm will be implemented directly in C++ Code.

Reference - LPS Program GCH39

3.4.31.1.1. Instance

Instance Name
MPS_PCL_FCV1_CO_CMD

3.4.31.1.2. PCL Initiating Commands

Command
V41K0052NL

3.4.31.1.3. Operations Index

Operation	**Description**
determineState()	This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE INHIBITED ERROR
activate()	This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.
inhibit()	This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.
getState()	This method is inherited from the base class - CAS_Control_Logic
resetCLNotification()	This method is inherited from the base class - CAS_Control_Logic

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.4.31.2. Operations

3.4.31.2.1. getState()

3.4.31.2.1.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.31.2.2. resetCLNotification()

3.4.31.2.2.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.31.2.3. determineState()

3.4.31.2.3.1. Descriptive Text

This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE, INHIBITED or ERROR.

3.4.31.2.3.2. Sequence

```
if  (<V41K0052NL>.PCLSTAT = active)
  {
  CLState = ACTIVE;
  }
else
  {
   if  (<V41K0052NL>.PCLSTAT = inhibited)
     {
     CLState = INHIBITED;
      }
    else
      {
     CLState = ERROR;
      }
   }
```

3.4.31.2.4. activate()

3.4.31.2.4.1. Descriptive Text

This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.31.2.4.2. Sequence

```
A FD <V41K0052NL> PCL;

return;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.4.31.2.5. inhibit()

3.4.31.2.5.1. Descriptive Text

This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.31.2.5.2. Sequence

I FD <V41K0052NL> PCL;

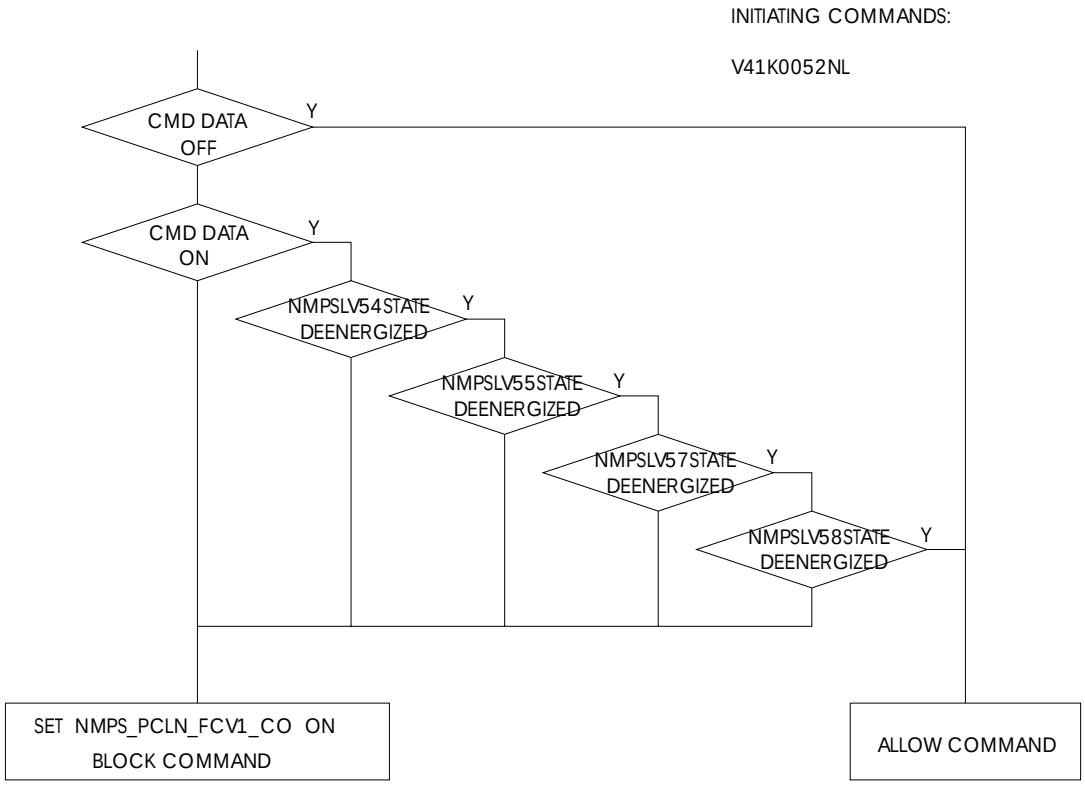
return;

3.4.31.3. Algorithm

3.4.31.3.1. PCL Algorithm

3.4.31.3.1.1. Flowchart

MPS_PCL_FCV1_CO_CMD



3.4.31.3.1.2. Sequence

#include "mps_pcl_fcv1_co_cmd.h"
#include "SME.h"
#include "LO2.h"

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
#include "MPS.h"
#include "LH2.h"

static const char* seqName = "MPS_PCL_FCV1_CO_CMD";
static const char* algorithmDescription = "Revision Date - 01/20/2000\n\
if (A || B || (C && (D && E && F && G)))\n\
where:   A  = NMPS_PCLO_FCV1_CO - PCL Bypass ON\n\
         B  = COMMAND DATA - OFF\n\
         C  = COMMAND DATA - ON\n\
         D  = LV54 - E2 GO2 FCV  - DEENERGIZED\n\
         E  = LV55 - E3 GO2 FCV  - DEENERGIZED\n\
         F  = LV57 - E2 GH2 FCV  - DEENERGIZED\n\
         G  = LV58 - E3 GH2 FCV  - DEENERGIZED";

MPS_PCL_FCV1_CO_CMD::MPS_PCL_FCV1_CO_CMD(const char*
seqName):asv_pcsVPCL_DS_ON_OFF_(seqName)
{
  userInit();
}

MPS_PCL_FCV1_CO_CMD::~MPS_PCL_FCV1_CO_CMD()
{
}


void MPS_PCL_FCV1_CO_CMD::userInit(void)
{
  initiatingFD(V41K0052NL);
}

int MPS_PCL_FCV1_CO_CMD::userExecute(Discrete::ON_OFF dsVar)
{

if (A() || B() || (C() && (D() && E() && F() && G())))
    {
   return PCL_ALLOW_CMD;
  }
  else
  {
   NMPS_PCLN_FCV1_CO.Set(Discrete::On);
   return PCL_BLOCK_CMD;
  }
}




// PCL Override ON

int MPS_PCL_FCV1_CO_CMD::A()
{
  int  valReturnA = PCL_VOTER_FAIL;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
   try
   {
    valReturnA =  (Discrete::On == (NMPS_PCLO_FCV1_CO.getValue())) ;
   }

   catch(...)
   {
    valReturnA = PCL_VOTER_FAIL;
   }

  return valReturnA;
}



// COMMAND DATA - OFF

int MPS_PCL_FCV1_CO_CMD::B()
{
  int  valReturnB = PCL_VOTER_FAIL;

   try
   {
    valReturnB =  (Discrete::Off == dsVar);
   }

   catch(...)
   {
    valReturnB = PCL_VOTER_FAIL;
   }

  return valReturnB;
}



// COMMAND DATA - ON

int MPS_PCL_FCV1_CO_CMD::C()
{
  int  valReturnC = PCL_VOTER_FAIL;

   try
   {
    valReturnC =  (Discrete::On == dsVar);
   }

   catch(...)
   {
    valReturnC = PCL_VOTER_FAIL;
   }

  return valReturnC;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
}



//LV54 - E#2 GO2 FCV - DEENERGIZED

int MPS_PCL_FCV1_CO_CMD::D()
{
  int  valReturnD = PCL_VOTER_FAIL;

try
{
            if (NMPSLV54STATE.getValue() == MPS_ORB_SOL_STATE::DEENERGIZED)
     {
              valReturnD = PCL_VOTER_PASS;
             }
else
            {
       valReturnD = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnD = PCL_VOTER_FAIL;
}

  return valReturnD;
}


//LV55 - E#3 GO2 FCV - DEENERGIZED

int MPS_PCL_FCV1_CO_CMD::E()
{
  int  valReturnE = PCL_VOTER_FAIL;

try
{
            if (NMPSLV55STATE.getValue() ==  MPS_ORB_SOL_STATE::DEENERGIZED)
     {
              valReturnE = PCL_VOTER_PASS;
             }
else
            {
       valReturnE = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnE = PCL_VOTER_FAIL;
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
  return valReturnE;
}



//LV57 - E#2 GH2 FCV - DEENERGIZED

int MPS_PCL_FCV1_CO_CMD::F()
{
  int  valReturnF = PCL_VOTER_FAIL;

try
{
            if (NMPSLV57STATE.getValue() == MPS_ORB_SOL_STATE::DEENERGIZED)
     {
              valReturnF = PCL_VOTER_PASS;
             }
else
            {
       valReturnF = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnF = PCL_VOTER_FAIL;
}

  return valReturnF;
}



//LV58 - E#3 GH2 FCV - DEENERGIZED

int MPS_PCL_FCV_CO_CMD::G()
{
  int  valReturnG = PCL_VOTER_FAIL;

try
{
            if (NMPSLV58STATE.getValue() ==  MPS_ORB_SOL_STATE::DEENERGIZED)
     {
              valReturnG = PCL_VOTER_PASS;
             }
else
            {
       valReturnG = PCL_VOTER_FAIL;
            }
}

catch(...)
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
{
    valReturnG = PCL_VOTER_FAIL;
}

 return valReturnG;
}
```

MPS_PCL_FCV1_CO_CMD mps_pcl_fcv1_co_cmd(seqName);

3.4.32. MPS_PCL_FCV2_CO_CMD

3.4.32.1. Description

This PCL provides protection for the MPS LH2 FCV #2 Checkout HI Commands.
This class inherits from the class CAS_Control_Logic. The Algorithm will be implemented directly in C++ Code.

Reference - LPS Program GCH39

3.4.32.1.1. Instance

Instance Name
MPS_PCL_FCV2_CO_CMD

3.4.32.1.2. PCL Initiating Commands

Command
V41K0053NL

3.4.32.1.3. Operations Index

Operation	**Description**
determineState()	This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE INHIBITED ERROR

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

activate()	This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.
inhibit()	This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.
getState()	This method is inherited from the base class - CAS_Control_Logic
resetCLNotification()	This method is inherited from the base class - CAS_Control_Logic

3.4.32.2. Operations

3.4.32.2.1. getState()

3.4.32.2.1.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.32.2.2. resetCLNotification()

3.4.32.2.2.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.32.2.3. determineState()

3.4.32.2.3.1. Descriptive Text

This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE, INHIBITED or ERROR.

3.4.32.2.3.2. Sequence

```
if  (<V41K0053NL>.PCLSTAT = active)
  {
  CLState = ACTIVE;
  }
else
  {
   if  (<V41K0053NL>.PCLSTAT = inhibited)
     {
     CLState = INHIBITED;
      }
    else
      {
     CLState = ERROR;
      }
   }
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.4.32.2.4. activate()

3.4.32.2.4.1. Descriptive Text

This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.32.2.4.2. Sequence

A FD <V41K0053NL> PCL;

return;

3.4.32.2.5. inhibit()

3.4.32.2.5.1. Descriptive Text

This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.32.2.5.2. Sequence

I FD <V41K0053NL> PCL;

return;

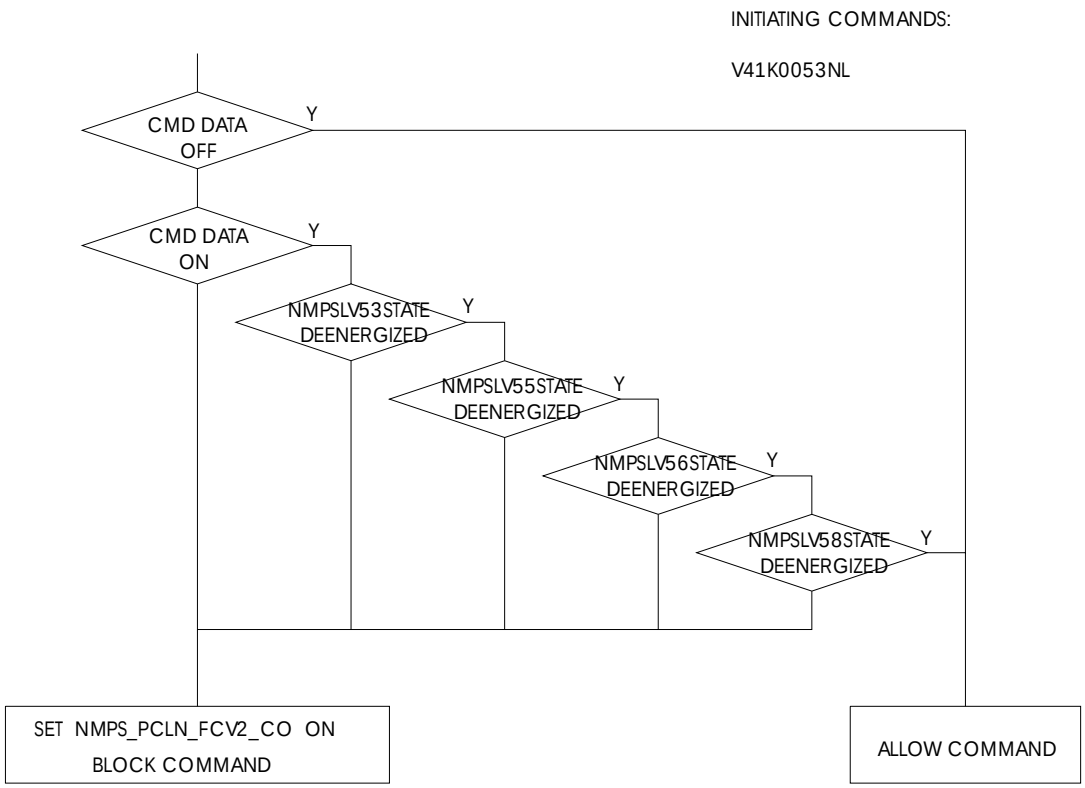
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.4.32.3. Algorithm

3.4.32.3.1. PCL Algorithm

3.4.32.3.1.1. Flowchart

MPS_PCL_FCV2_CO_CMD



3.4.32.3.1.2. Sequence

```
#include "mps_pcl_fcv2_co_cmd.h"
#include "SME.h"
#include "LO2.h"
#include "MPS.h"
#include "LH2.h"

static const char* seqName = "MPS_PCL_FCV2_CO_CMD";
static const char* algorithmDescription = "Revision Date - 01/20/2000\n\
if (A || B || (C && (D && E && F && G)))\n\
where:   A  = NMPS_PCLO_FCV2_CO - PCL Bypass ON\n\
         B  = COMMAND DATA - OFF\n\
         C  = COMMAND DATA - ON\n\
         D  = LV53 - E1 GO2 FCV  - DEENERGIZED\n\
         E  = LV55 - E3 GO2 FCV  - DEENERGIZED\n\
         F  = LV56 - E1 GH2 FCV  - DEENERGIZED\n\
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
          G = LV58 - E3 GH2 FCV  - DEENERGIZED";

MPS_PCL_FCV2_CO_CMD::MPS_PCL_FCV2_CO_CMD(const char*
seqName):asv_pcsVPCL_DS_ON_OFF_(seqName)
{
  userInit();
}

MPS_PCL_FCV2_CO_CMD::~MPS_PCL_FCV2_CO_CMD()
{
}


void MPS_PCL_FCV2_CO_CMD::userInit(void)
{
  initiatingFD(V41K0053NL);
}

int MPS_PCL_FCV2_CO_CMD::userExecute(Discrete::ON_OFF dsVar)
{

if (A() || B() || (C() && (D() && E() && F() && G())))
    {
   return PCL_ALLOW_CMD;
  }
  else
  {
   NMPS_PCLN_FCV2_CO.Set(Discrete::On);
   return PCL_BLOCK_CMD;
  }
}




// PCL Override ON

int MPS_PCL_FCV2_CO_CMD::A()
{
  int  valReturnA = PCL_VOTER_FAIL;

    try
    {
      valReturnA =  (Discrete::On == (NMPS_PCLO_FCV2_CO.getValue())) ;
    }

    catch(...)
    {
      valReturnA = PCL_VOTER_FAIL;
    }

  return valReturnA;
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
// COMMAND DATA - OFF

int MPS_PCL_FCV2_CO_CMD::B()
{
  int  valReturnB = PCL_VOTER_FAIL;

    try
    {
      valReturnB =  (Discrete::Off == dsVar);
    }

    catch(...)
    {
      valReturnB = PCL_VOTER_FAIL;
    }

  return valReturnB;
}



// COMMAND DATA - ON

int MPS_PCL_FCV2_CO_CMD::C()
{
  int  valReturnC = PCL_VOTER_FAIL;

    try
    {
      valReturnC =  (Discrete::On == dsVar);
    }

    catch(...)
    {
      valReturnC = PCL_VOTER_FAIL;
    }

  return valReturnC;
}



//LV53 - E#1 GO2 FCV - DEENERGIZED

int MPS_PCL_FCV2_CO_CMD::D()
{
  int  valReturnD = PCL_VOTER_FAIL;

try
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
{
            if (NMPSLV53STATE.getValue() == MPS_ORB_SOL_STATE::DEENERGIZED)
     {
              valReturnD = PCL_VOTER_PASS;
             }
else
            {
       valReturnD = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnD = PCL_VOTER_FAIL;
}

  return valReturnD;
}




//LV55 - E#3 GO2 FCV - DEENERGIZED

int MPS_PCL_FCV2_CO_CMD::E()
{
  int  valReturnE = PCL_VOTER_FAIL;

try
{
            if (NMPSLV55STATE.getValue() == MPS_ORB_SOL_STATE::DEENERGIZED)
     {
              valReturnE = PCL_VOTER_PASS;
             }
else
            {
       valReturnE = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnE = PCL_VOTER_FAIL;
}

  return valReturnE;
}




//LV56 - E#1GH2 FCV - DEENERGIZED
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
int MPS_PCL_FCV2_CO_CMD::F()
{
  int  valReturnF = PCL_VOTER_FAIL;

try
{
            if (NMPSLV56STATE.getValue() == MPS_ORB_SOL_STATE::DEENERGIZED)
     {
              valReturnF = PCL_VOTER_PASS;
             }
else
            {
       valReturnF = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnF = PCL_VOTER_FAIL;
}

  return valReturnF;
}



//LV58 - E#3 GH2 FCV - DEENERGIZED

int MPS_PCL_FCV2_CO_CMD::G()
{
  int  valReturnG = PCL_VOTER_FAIL;

try
{
            if (NMPSLV58STATE.getValue() == MPS_ORB_SOL_STATE::DEENERGIZED)
     {
              valReturnG = PCL_VOTER_PASS;
             }
else
            {
       valReturnG = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnG = PCL_VOTER_FAIL;
}

  return valReturnG;
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

MPS_PCL_FCV2_CO_CMD mps_pcl_fcv2_co_cmd(seqName);

3.4.33. MPS_PCL_FCV3_CO_CMD

3.4.33.1. Description

This PCL provides protection for the MPS LH2 FCV #3 Checkout HI Commands.
This class inherits from the class CAS_Control_Logic. The Algorithm will be implemented directly in C++ Code.

Reference - LPS Program GCH39

3.4.33.1.1. Instance

Instance Name
MPS_PCL_FCV_CO_CMD

3.4.33.1.2. PCL Initiating Commands

Command
V41K0054NL

3.4.33.1.3. Operations Index

Operation	**Description**
determineState()	This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE INHIBITED ERROR
activate()	This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.
inhibit()	This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.
getState()	This method is inherited from the base class - CAS_Control_Logic

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

resetCLNotification()	This method is inherited from the base class - CAS_Control_Logic

3.4.33.2. Operations

3.4.33.2.1. getState()

3.4.33.2.1.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.33.2.2. resetCLNotification()

3.4.33.2.2.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.33.2.3. determineState()

3.4.33.2.3.1. Descriptive Text

This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE, INHIBITED or ERROR.

3.4.33.2.3.2. Sequence

```
if  (<V41K0054NL>.PCLSTAT = active)
  {
  CLState = ACTIVE;
  }
else
  {
   if  (<V41K0054NL>.PCLSTAT = inhibited)
     {
     CLState = INHIBITED;
      }
    else
     {
     CLState = ERROR;
      }
   }
```

3.4.33.2.4. activate()

3.4.33.2.4.1. Descriptive Text

This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.33.2.4.2. Sequence

```
A FD <V41K0054NL> PCL;

return;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.4.33.2.5. inhibit()

3.4.33.2.5.1. Descriptive Text

This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.33.2.5.2. Sequence

I FD <V41K0054NL> PCL;

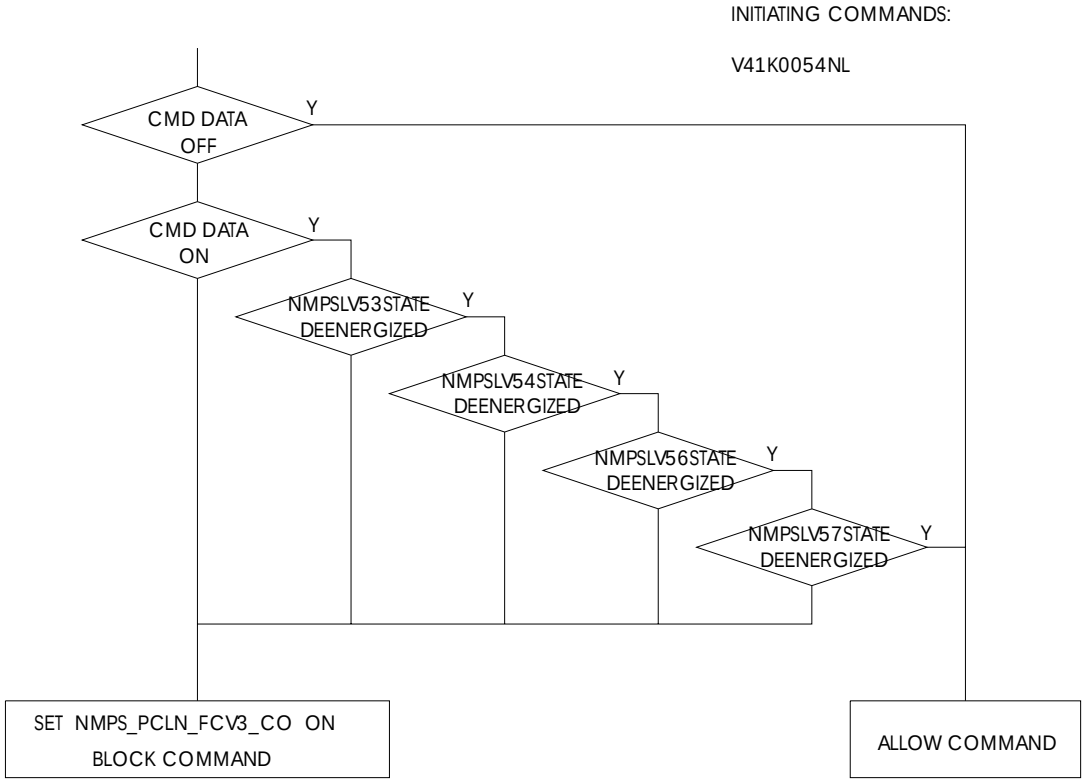
return;

3.4.33.3. Algorithm

3.4.33.3.1. PCL Algorithm

3.4.33.3.1.1. Flowchart

MPS_PCL_FCV3_CO_CMD



3.4.33.3.1.2. Sequence

#include "mps_pcl_fcv3_co_cmd.h"

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
#include "SME.h"
#include "LO2.h"
#include "MPS.h"
#include "LH2.h"

static const char* seqName = "MPS_PCL_FCV3_CO_CMD";
static const char* algorithmDescription = "Revision Date - 01/20/2000\n\
if (A || B || (C && (D && E && F && G)))\n\
where:   A  = NMPS_PCLO_FCV3_CO - PCL Bypass ON\n\
         B  = COMMAND DATA - OFF\n\
         C  = COMMAND DATA - ON\n\
         D  = LV53 - E1 GO2 FCV  - DEENERGIZED\n\
         E  = LV54 - E2 GO2 FCV  - DEENERGIZED\n\
         F  = LV56 - E1 GH2 FCV  - DEENERGIZED\n\
         G  = LV57 - E2 GH2 FCV  - DEENERGIZED";

MPS_PCL_FCV3_CO_CMD::MPS_PCL_FCV3_CO_CMD(const char*
seqName):asv_pcsVPCL_DS_ON_OFF_(seqName)
{
  userInit();
}

MPS_PCL_FCV3_CO_CMD::~MPS_PCL_FCV3_CO_CMD()
{
}


void MPS_PCL_FCV3_CO_CMD::userInit(void)
{
  initiatingFD(V41K0054NL);
}

int MPS_PCL_FCV3_CO_CMD::userExecute(Discrete::ON_OFF dsVar)
{

if (A() || B() || (C() && (D() && E() && F() && G())))
    {
   return PCL_ALLOW_CMD;
  }
  else
  {
   NMPS_PCLN_FCV3_CO.Set(Discrete::On);
   return PCL_BLOCK_CMD;
  }
}




// PCL Override ON

int MPS_PCL_FCV3_CO_CMD::A()
{
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
  int  valReturnA = PCL_VOTER_FAIL;

    try
    {
     valReturnA =  (Discrete::On == (NMPS_PCLO_FCV3_CO.getValue())) ;
    }

    catch(...)
    {
     valReturnA = PCL_VOTER_FAIL;
    }

  return valReturnA;
}



// COMMAND DATA - OFF

int MPS_PCL_FCV3_CO_CMD::B()
{
  int  valReturnB = PCL_VOTER_FAIL;

    try
    {
     valReturnB =  (Discrete::Off == dsVar);
    }

    catch(...)
    {
     valReturnB = PCL_VOTER_FAIL;
    }

  return valReturnB;
}



// COMMAND DATA - ON

int MPS_PCL_FCV3_CO_CMD::C()
{
  int  valReturnC = PCL_VOTER_FAIL;

    try
    {
     valReturnC =  (Discrete::On == dsVar);
    }

    catch(...)
    {
     valReturnC = PCL_VOTER_FAIL;
    }
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
  return valReturnC;
}




//LV53 - E#1 GO2 FCV - DEENERGIZED

int MPS_PCL_FCV3_CO_CMD::D()
{
  int  valReturnD = PCL_VOTER_FAIL;

try
{
            if (NMPSLV53STATE.getValue() == MPS_ORB_SOL_STATE::DEENERGIZED)
     {
              valReturnD = PCL_VOTER_PASS;
             }
else
            {
       valReturnD = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnD = PCL_VOTER_FAIL;
}

  return valReturnD;
}




//LV54 - E#2 GO2 FCV - DEENERGIZED

int MPS_PCL_FCV3_CO_CMD::E()
{
  int  valReturnE = PCL_VOTER_FAIL;

try
{
            if (NMPSLV54STATE.getValue() == MPS_ORB_SOL_STATE::DEENERGIZED)
     {
              valReturnE = PCL_VOTER_PASS;
             }
else
            {
       valReturnE = PCL_VOTER_FAIL;
            }
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
catch(...)
{
    valReturnE = PCL_VOTER_FAIL;
}

 return valReturnE;
}



//LV56 - E#1GH2 FCV - DEENERGIZED

int MPS_PCL_FCV3_CO_CMD::F()
{
 int  valReturnF = PCL_VOTER_FAIL;

try
{
          if (NMPSLV56STATE.getValue() == MPS_ORB_SOL_STATE::DEENERGIZED)
    {
            valReturnF = PCL_VOTER_PASS;
           }
else
          {
      valReturnF = PCL_VOTER_FAIL;
          }
}

catch(...)
{
    valReturnF = PCL_VOTER_FAIL;
}

 return valReturnF;
}


//LV57 - E#2 GH2 FCV - DEENERGIZED

int MPS_PCL_FCV3_CO_CMD::G()
{
 int  valReturnG = PCL_VOTER_FAIL;

try
{
          if (NMPSLV57STATE.getValue() == MPS_ORB_SOL_STATE::DEENERGIZED)
    {
            valReturnG = PCL_VOTER_PASS;
           }
else
          {
      valReturnG = PCL_VOTER_FAIL;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
            }
}

catch(...)
{
    valReturnG = PCL_VOTER_FAIL;
}

 return valReturnG;
}


MPS_PCL_FCV3_CO_CMD mps_pcl_fcv3_co_cmd(seqName);
```

3.4.34. MPS_PCL_PNEU_OP_CMD

3.4.34.1. Description

This PCL provides protection for the MPS Pneumatic GHe Isolation Valves Open Commands.
This class inherits from the class CAS_Control_Logic. The Algorithm will be implemented directly in C++ Code.

Reference - LPS Program VCE11/VCE12

3.4.34.1.1. Instance

Instance Name
MPS_PCL_PNEU_OP_CMD

3.4.34.1.2. PCL Initiating Commands

Command
V41K1607XL
V41K1608XL

3.4.34.1.3. Operations Index

Operation	**Description**
determineState()	This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either:

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

	ACTIVE INHIBITED ERROR
activate()	This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.
inhibit()	This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.
getState()	This method is inherited from the base class - CAS_Control_Logic
resetCLNotification()	This method is inherited from the base class - CAS_Control_Logic

3.4.34.2. Operations

3.4.34.2.1. getState()

3.4.34.2.1.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.34.2.2. resetCLNotification()

3.4.34.2.2.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.34.2.3. determineState()

3.4.34.2.3.1. Descriptive Text

This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE, INHIBITED or ERROR.

3.4.34.2.3.2. Sequence

```
if  (<V41K1607XL>.PCLSTAT = active  AND
     <V41K1608XL>.PCLSTAT = active)
  {
  CLState = ACTIVE;
  }
else
  {
   if  (<V41K1607XL>.PCLSTAT = inhibited  AND
        <V41K1607XL>.PCLSTAT = inhibited)
     {
     CLState = INHIBITED;
      }
    else
      {
     CLState = ERROR;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
    }
  }
```

3.4.34.2.4. activate()

3.4.34.2.4.1. Descriptive Text

This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.34.2.4.2. Sequence

```
A FD <V41K1607XL> PCL;
A FD <V41K1608XL> PCL;

return;
```

3.4.34.2.5. inhibit()

3.4.34.2.5.1. Descriptive Text

This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.34.2.5.2. Sequence

```
I FD <V41K1607XL> PCL;
I FD <V41K1608XL> PCL;

return;
```

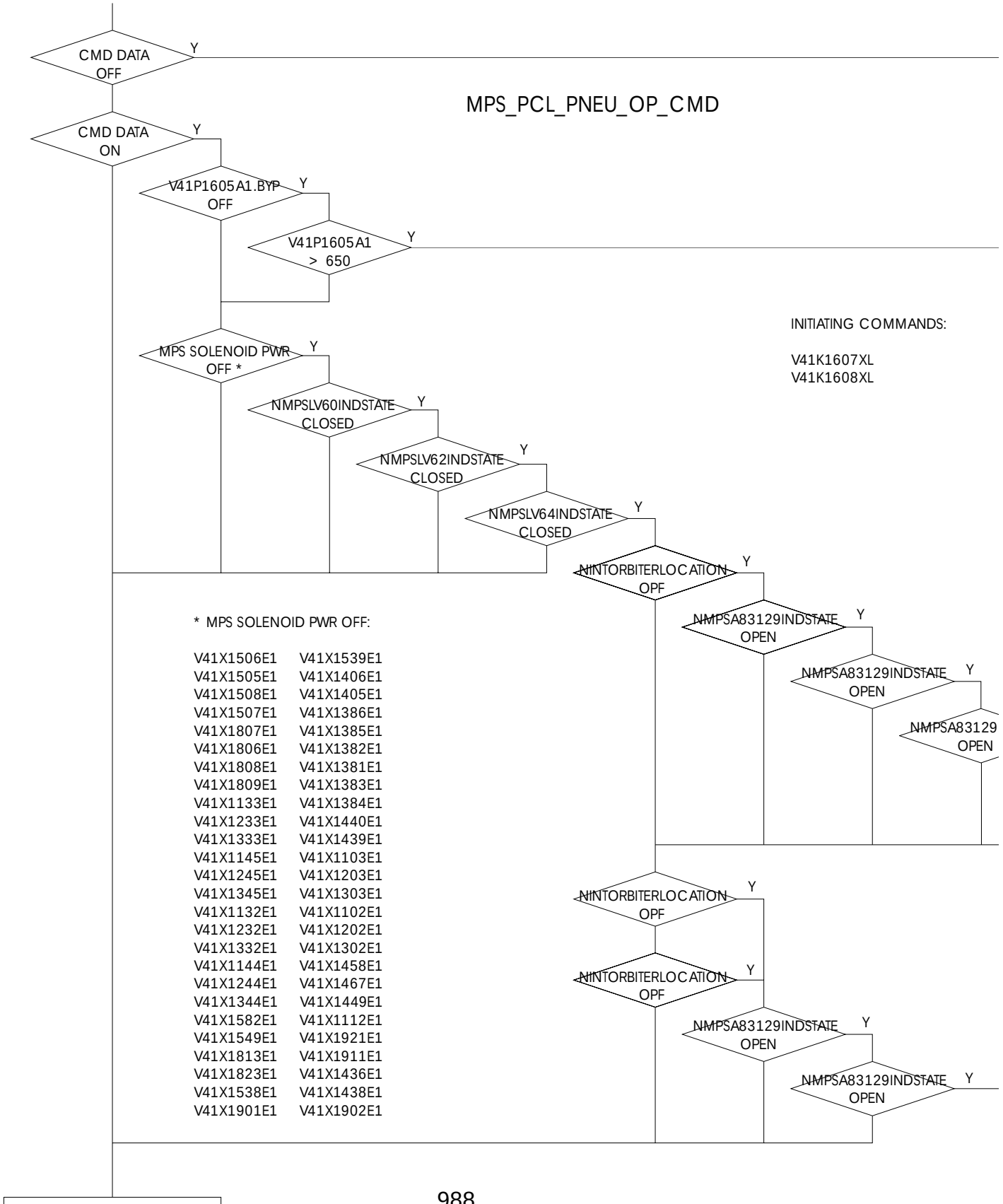

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.4.34.3. Algorithm

3.4.34.3.1. PCL Algorithm

3.4.34.3.1.1. Flowchart

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.4.34.3.1.2. Sequence

```
#include "mps_pcl_pneu_op_cmd.h"
#include "SME.h"
#include "LO2.h"
#include "MPS.h"
#include "LH2.h"

static const char* seqName = "MPS_PCL_PNEU_OP_CMD";
static const char* algorithmDescription = "Revision Date - 01/20/2000\n\
if (A || B || (C && (D || ((E &&F && G && H) && ((I && J && K && L && M) || (N && O)))))\n\
where:   A  = NMPS_PCLO_PNEU_OP - PCL Bypass ON\n\
         B  = COMMAND DATA - OFF\n\
         C  = COMMAND DATA - ON\n\
         D  = MPS Pneu Reg Press > 650 psia\n\
         E  = ALL MPS SOLENOID POWER - OFF\n\
         F  = LV60 - E1 INTERCONNECT OUT VALVE - CLOSED\n\
         G  = LV62 - E2 INTERCONNECT OUT VALVE - CLOSED\n\
         H  = LV64 - E3 INTERCONNECT OUT VALVE - CLOSED\n\
         I  = NO3IS078E - ORB LOCATED IN OPF - ON\n\
         J  = A83129 - HE FILL LO FLOW SOV - CLOSED\n\
         K  = A83130 - HE FILL HI FLOW SOV - CLOSED\n\
         L  = A83132 - 2K HE FILL HI FLOW SOV - CLOSED\n\
         M  = A83133 - 2K HE FILL LO FLOW SOV - CLOSED\n\
         N  = NO3IS078E - ORB LOCATED IN OPF - OFF\n\
         O  = A74983 - HE FILL LO FLOW SOV - CLOSED";

MPS_PCL_PNEU_OP_CMD::MPS_PCL_PNEU_OP_CMD(const char*
seqName):asv_pcsVPCL_DS_ON_OFF_(seqName)
{
  userInit();
}

MPS_PCL_PNEU_OP_CMD::~MPS_PCL_PNEU_OP_CMD()
{
}


void MPS_PCL_PNEU_OP_CMD::userInit(void)
{
  initiatingFD(V41K1607XL);
  initiatingFD(V41K1608XL);
}

int MPS_PCL_PNEU_OP_CMD::userExecute(Discrete::ON_OFF dsVar)
{

if  (A() || B() || (C() && (D() || ((E() && F() && G() && H()) && ((I() && J() && K() && L() && M()) || (N() &&
O())))))
    {
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
   return PCL_ALLOW_CMD;
  }
  else
  {
  NMPS_PCLN_PNEU_OP.Set(Discrete::On);
   return PCL_BLOCK_CMD;
  }
}



// Function that determines the state of the indicator bypasses.

int MPS_PCL_PNEU_OP_CMD::indicatorBypass(PD_ &pseudo_ind)
{
  if ( Discrete::On == pseudo_ind.getValue())
   return IND_BYPASS_ON;
 else
   return IND_BYPASS_OFF;
}




// PCL Override ON

int MPS_PCL_PNEU_OP_CMD::A()
{
  int  valReturnA = PCL_VOTER_FAIL;

    try
    {
      valReturnA =  (Discrete::On == (NMPS_PCLO_PNEU_OP.getValue()));
    }

    catch(...)
    {
      valReturnA = PCL_VOTER_FAIL;
    }

  return valReturnA;
}



// COMMAND DATA - OFF

int MPS_PCL_PNEU_OP_CMD::B()
{
  int  valReturnB = PCL_VOTER_FAIL;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
    try
    {
      valReturnB =  (Discrete::Off == dsVar);
    }

    catch(...)
    {
      valReturnB = PCL_VOTER_FAIL;
    }

  return valReturnB;
}



// COMMAND DATA - ON

int MPS_PCL_PNEU_OP_CMD::C()
{
  int  valReturnC = PCL_VOTER_FAIL;

    try
    {
      valReturnC =  (Discrete::On == dsVar);
    }

    catch(...)
    {
      valReturnC = PCL_VOTER_FAIL;
    }

  return valReturnC;
}




// MPS Pneu Reg Press > 650

int MPS_PCL_PNEU_OP_CMD::D()
{
  int  valReturnD = PCL_VOTER_FAIL;

  if (indicatorBypass(N41P1605A1))
  {
    try
    {
      valReturnD =  (V41P1605A1.getValue() > 650);
    }

    catch(...)
    {
      valReturnD = PCL_VOTER_FAIL;
    }
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
  }

  else
  {
    valReturnD = PCL_VOTER_FAIL;
  }

  return valReturnD;
}




// All MPS Solenoid Power - OFF

int MPS_PCL_PNEU_OP_CMD::E()
{
  int  valReturnE = PCL_VOTER_FAIL;

 try
 {
                    if ((V41X1506E1.getValue() == Discrete::Off) &&
                        (V41X1505E1.getValue() == Discrete::Off) &&
                        (V41X1508E1.getValue() == Discrete::Off) &&
                        (V41X1507E1.getValue() == Discrete::Off) &&
                        (V41X1807E1.getValue() == Discrete::Off) &&
                        (V41X1806E1.getValue() == Discrete::Off) &&
                        (V41X1808E1.getValue() == Discrete::Off) &&
                        (V41X1809E1.getValue() == Discrete::Off) &&
                        (V41X1133E1.getValue() == Discrete::Off) &&
                        (V41X1233E1.getValue() == Discrete::Off) &&
                        (V41X1333E1.getValue() == Discrete::Off) &&
                        (V41X1145E1.getValue() == Discrete::Off) &&
                        (V41X1245E1.getValue() == Discrete::Off) &&
                        (V41X1345E1.getValue() == Discrete::Off) &&
                        (V41X1132E1.getValue() == Discrete::Off) &&
                        (V41X1232E1.getValue() == Discrete::Off) &&
                        (V41X1332E1.getValue() == Discrete::Off) &&
                        (V41X1144E1.getValue() == Discrete::Off) &&
                        (V41X1244E1.getValue() == Discrete::Off) &&
                        (V41X1344E1.getValue() == Discrete::Off) &&
                        (V41X1582E1.getValue() == Discrete::Off) &&
                        (V41X1549E1.getValue() == Discrete::Off) &&
                        (V41X1813E1.getValue() == Discrete::Off) &&
                        (V41X1823E1.getValue() == Discrete::Off) &&
                        (V41X1538E1.getValue() == Discrete::Off) &&
                        (V41X1539E1.getValue() == Discrete::Off) &&
                        (V41X1406E1.getValue() == Discrete::Off) &&
                        (V41X1405E1.getValue() == Discrete::Off) &&
                        (V41X1386E1.getValue() == Discrete::Off) &&
                        (V41X1385E1.getValue() == Discrete::Off) &&
                        (V41X1382E1.getValue() == Discrete::Off) &&
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
                    (V41X1381E1.getValue() == Discrete::Off) &&
                    (V41X1383E1.getValue() == Discrete::Off) &&
                    (V41X1384E1.getValue() == Discrete::Off) &&
                    (V41X1440E1.getValue() == Discrete::Off) &&
                    (V41X1439E1.getValue() == Discrete::Off) &&
                    (V41X1103E1.getValue() == Discrete::Off) &&
                    (V41X1203E1.getValue() == Discrete::Off) &&
                    (V41X1303E1.getValue() == Discrete::Off) &&
                    (V41X1102E1.getValue() == Discrete::Off) &&
                    (V41X1202E1.getValue() == Discrete::Off) &&
                    (V41X1302E1.getValue() == Discrete::Off) &&
                    (V41X1458E1.getValue() == Discrete::Off) &&
                    (V41X1467E1.getValue() == Discrete::Off) &&
                    (V41X1449E1.getValue() == Discrete::Off) &&
                    (V41X1112E1.getValue() == Discrete::Off) &&
                    (V41X1921E1.getValue() == Discrete::Off) &&
                    (V41X1911E1.getValue() == Discrete::Off) &&
                    (V41X1436E1.getValue() == Discrete::Off) &&
                    (V41X1438E1.getValue() == Discrete::Off) &&
                    (V41X1901E1.getValue() == Discrete::Off) &&
                    (V41X1902E1.getValue() == Discrete::Off))
          {
            valReturnE = PCL_VOTER_PASS;
           }
else
           {
      valReturnE = PCL_VOTER_FAIL;
           }
}

catch(...)
{
    valReturnE = PCL_VOTER_FAIL;
}

 return valReturnE;
}
```

```
//LV60 - E#1 Interconnect Out SOV - Closed

int MPS_PCL_PNEU_OP_CMD::F()
{
  int  valReturnF = PCL_VOTER_FAIL;

try
{
          if (NMPSLV60INDSTATE.getValue() == MPS_VALVE_IND_STATE::CLOSED)
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
    {
            valReturnF = PCL_VOTER_PASS;
           }
else
           {
      valReturnF = PCL_VOTER_FAIL;
           }
}

catch(...)
{
    valReturnF = PCL_VOTER_FAIL;
}

 return valReturnF;
}



//LV62 - E#2 Interconnect Out SOV - Closed

int MPS_PCL_PNEU_OP_CMD::G()
{
 int  valReturnG = PCL_VOTER_FAIL;

try
{
           if (NMPSLV62INDSTATE.getValue() == MPS_VALVE_IND_STATE::CLOSED)
    {
            valReturnG = PCL_VOTER_PASS;
           }
else
           {
      valReturnG = PCL_VOTER_FAIL;
           }
}

catch(...)
{
    valReturnG = PCL_VOTER_FAIL;
}

 return valReturnG;
}



//LV64 - E#3 Interconnect Out SOV - Closed

int MPS_PCL_PNEU_OP_CMD::H()
{
 int  valReturnH = PCL_VOTER_FAIL;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
try
{
            if (NMPSLV64INDSTATE.getValue() == MPS_VALVE_IND_STATE::CLOSED)
     {
              valReturnH = PCL_VOTER_PASS;
             }
else
            {
       valReturnH = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnH = PCL_VOTER_FAIL;
}

  return valReturnH;
}




// ORB Location = OPF

int MPS_PCL_PNEU_OP_CMD::I()
{
  int  valReturnI = PCL_VOTER_FAIL;

    try
    {
      valReturnI =  (dISCRETE::ON == N03IS078E.getValue());
    }

    catch(...)
    {
      valReturnI = PCL_VOTER_FAIL;
    }

  return valReturnI;
}




//  A83129 - OPF He Fill Lo Flow SOV - Closed

int MPS_PCL_PNEU_OP_CMD::J()
{
  int  valReturnJ = PCL_VOTER_FAIL;

try
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
{
            if (NMPSA83129INDSTATE.getValue() == MPS_VALVE_IND_STATE::CLOSED)
     {
              valReturnJ = PCL_VOTER_PASS;
             }
else
            {
       valReturnJ = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnJ = PCL_VOTER_FAIL;
}

  return valReturnJ;
}




//  A83130 - OPF He Fill Hi Flow SOV - Closed

int MPS_PCL_PNEU_OP_CMD::K()
{
  int  valReturnK = PCL_VOTER_FAIL;

try
{
            if (NMPSA83130INDSTATE.getValue() == MPS_VALVE_IND_STATE::CLOSED)
     {
              valReturnK = PCL_VOTER_PASS;
             }
else
            {
       valReturnK = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnK = PCL_VOTER_FAIL;
}

  return valReturnK;
}




//  A83132 - OPF 2K He Fill Hi Flow SOV - Closed
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
int MPS_PCL_PNEU_OP_CMD::L()
{
  int  valReturnL = PCL_VOTER_FAIL;

try
{
            if (NMPSA83132INDSTATE.getValue() == MPS_VALVE_IND_STATE::CLOSED)
     {
              valReturnL = PCL_VOTER_PASS;
             }
else
            {
       valReturnL = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnL = PCL_VOTER_FAIL;
}

  return valReturnL;
}



//  A83133 - OPF 2K He Fill Lo Flow SOV - Closed

int MPS_PCL_PNEU_OP_CMD::M()
{
  int  valReturnM = PCL_VOTER_FAIL;

try
{
            if (NMPSA83133INDSTATE.getValue() == MPS_VALVE_IND_STATE::CLOSED)
     {
              valReturnM = PCL_VOTER_PASS;
             }
else
            {
       valReturnM = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnM = PCL_VOTER_FAIL;
}

  return valReturnM;
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
// ORB Location = MLP

int MPS_PCL_PNEU_OP_CMD::N()
{
  int  valReturnN = PCL_VOTER_FAIL;

    try
    {
      valReturnN =  (Discrete::Off == N03IS078E.getValue());
    }

    catch(...)
    {
      valReturnN = PCL_VOTER_FAIL;
    }

  return valReturnN;
}




//  A74983 - MLP He Fill SOV - Closed

int MPS_PCL_PNEU_OP_CMD::O()
{
  int  valReturnO = PCL_VOTER_FAIL;

try
{
            if (NMPSA74983INDSTATE.getValue() == MPS_VALVE_IND_STATE::CLOSED)
     {
              valReturnO = PCL_VOTER_PASS;
             }
else
            {
       valReturnO = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnO = PCL_VOTER_FAIL;
}

  return valReturnO;
}
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

MPS_PCL_PNEU_OP_CMD mps_pcl_pneu_op_cmd(seqName);

3.4.35. MPS_PCL_LV10_OP_CMD

3.4.35.1. Description

This PCL provides protection for the MPS Pneumatic GHe Crossover Valve Open Command.
This class inherits from the class CAS_Control_Logic. The Algorithm will be implemented directly in C++ Code.

Reference - LPS Program VCE13

3.4.35.1.1. Instance

Instance Name
MPS_PCL_LV10_OP_CMD

3.4.35.1.2. PCL Initiating Commands

Command
V41K1613XL

3.4.35.1.3. Operations Index

Operation	**Description**
determineState()	This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE INHIBITED ERROR
activate()	This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

inhibit()	This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.
getState()	This method is inherited from the base class - CAS_Control_Logic
resetCLNotification()	This method is inherited from the base class - CAS_Control_Logic

3.4.35.2. Operations

3.4.35.2.1. getState()

3.4.35.2.1.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.35.2.2. resetCLNotification()

3.4.35.2.2.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.35.2.3. determineState()

3.4.35.2.3.1. Descriptive Text

This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE, INHIBITED or ERROR.

3.4.35.2.3.2. Sequence

```
if  (<V41K1613XL>.PCLSTAT = active)
  {
  CLState = ACTIVE;
  }
else
  {
   if  (<V41K1613XL>.PCLSTAT = inhibited)
     {
     CLState = INHIBITED;
      }
    else
      {
     CLState = ERROR;
      }
   }
```

3.4.35.2.4. activate()

3.4.35.2.4.1. Descriptive Text

This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.4.35.2.4.2. Sequence

A FD <V41K1613XL> PCL;

return;

3.4.35.2.5. inhibit()

3.4.35.2.5.1. Descriptive Text

This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.35.2.5.2. Sequence

I FD <V41K1613XL> PCL;

return;

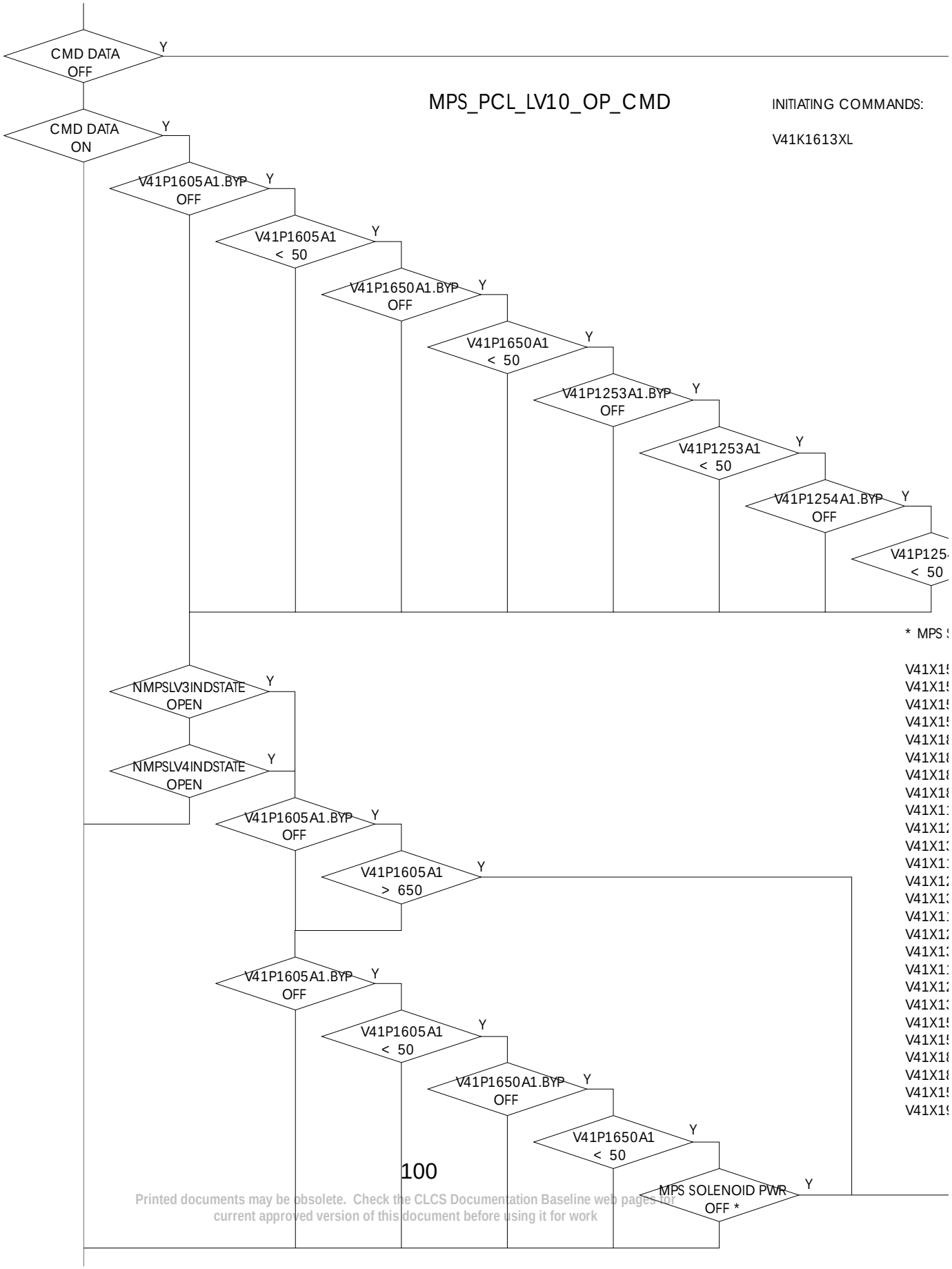
Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.4.35.3. Algorithm

3.4.35.3.1. PCL Algorithm

3.4.35.3.1.1. Flowchart

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work



Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.4.35.3.1.2. Sequence

```
#include "mps_pcl_lv10_op_cmd.h"
#include "SME.h"
#include "LO2.h"
#include "MPS.h"
#include "LH2.h"

static const char* seqName = "MPS_PCL_LV10_OP_CMD";
static const char* algorithmDescription = "Revision Date - 01/20/2000\n\
if (A || B || (C && (D && E && F && G) || (H && (I || J)) || (D && E && (I || J) && K)))\n\
where:   A  = NMPS_PCLO_LV10_OP - PCL Bypass ON\n\
         B  = COMMAND DATA - OFF\n\
         C  = COMMAND DATA - ON\n\
         D  = MPS Pneu Reg Press < 50 psia\n\
         E  = MPS Pneu Accum Press < 50 psia\n\
         F  = MPS E2 Pneu Reg A Press < 50 psia\n\
         G  = MPS E2 Pneu Reg B Press < 50 psia\n\
         H  = MPS Pneu Reg Press > 650 psia\n\
         I  = LV3 - E2 PNEU HE REG ISO A SOV - OPEN\n\
         J  = LV4 - E2 PNEU HE REG ISO B SOV - OPEN\n\
         K  = ALL MPS SOLENOID POWER - OFF";

MPS_PCL_LV10_OP_CMD::MPS_PCL_LV10_OP_CMD(const char*
seqName):asv_pcsVPCL_DS_ON_OFF_(seqName)
{
  userInit();
}

MPS_PCL_LV10_OP_CMD::~MPS_PCL_LV10_OP_CMD()
{
}


void MPS_PCL_LV10_OP_CMD::userInit(void)
{
  initiatingFD(V41K1613XL);
}

int MPS_PCL_LV10_OP_CMD::userExecute(Discrete::ON_OFF dsVar)
{

if  (A() || B() || (C() && (D() && E() && F() && G()) || (H() && (I() || J())) || (D() && E() && (I() || J()) && K())))
    {
   return PCL_ALLOW_CMD;
  }
  else
  {
   NMPS_PCLN_LV10_OP.Set(Discrete::On);
   return PCL_BLOCK_CMD;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
  }
}



// Function that determines the state of the indicator bypasses.

int MPS_PCL_LV10_OP_CMD::indicatorBypass(PD_ &pseudo_ind)
{
  if ( Discrete::On == pseudo_ind.getValue())
   return IND_BYPASS_ON;
 else
   return IND_BYPASS_OFF;
}




// PCL Override ON

int MPS_PCL_LV10_OP_CMD::A()
{
  int  valReturnA = PCL_VOTER_FAIL;

    try
    {
      valReturnA =  (Discrete::On == (NMPS_PCLO_LV10_OP.getValue()));
    }

    catch(...)
    {
      valReturnA = PCL_VOTER_FAIL;
    }

  return valReturnA;
}



// COMMAND DATA - OFF

int MPS_PCL_LV10_OP_CMD::B()
{
  int  valReturnB = PCL_VOTER_FAIL;

    try
    {
      valReturnB =  (Discrete::Off == dsVar);
    }

    catch(...)
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
    {
      valReturnB = PCL_VOTER_FAIL;
    }

  return valReturnB;
}



// COMMAND DATA - ON

int MPS_PCL_LV10_OP_CMD::C()
{
  int  valReturnC = PCL_VOTER_FAIL;

    try
    {
      valReturnC =  (Discrete::On == dsVar);
    }

    catch(...)
    {
      valReturnC = PCL_VOTER_FAIL;
    }

  return valReturnC;
}




// MPS Pneu Reg Press < 50

int MPS_PCL_LV10_OP_CMD::D()
{
  int  valReturnD = PCL_VOTER_FAIL;

  if (indicatorBypass(N41P1605A1))
  {
    try
    {
      valReturnD =  (V41P1605A1.getValue() < 50);
    }

    catch(...)
    {
      valReturnD = PCL_VOTER_FAIL;
    }
  }

  else
  {
    valReturnD = PCL_VOTER_FAIL;
  }
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
  return valReturnD;
}



// MPS Pneu Accum Press < 50

int MPS_PCL_LV10_OP_CMD::E()
{
  int  valReturnE = PCL_VOTER_FAIL;

  if (indicatorBypass(N41P1650A1))
  {
    try
    {
      valReturnE =  (V41P1650A1.getValue() < 50);
    }

    catch(...)
    {
      valReturnE = PCL_VOTER_FAIL;
    }
  }

  else
  {
    valReturnE = PCL_VOTER_FAIL;
  }

  return valReturnE;
}



// MPS E2 Reg A Pneu Out Press < 50

int MPS_PCL_LV10_OP_CMD::F()
{
  int  valReturnF = PCL_VOTER_FAIL;

  if (indicatorBypass(N41P1254A1))
  {
    try
    {
      valReturnF =  (V41P1254A1.getValue() < 50);
    }

    catch(...)
    {
      valReturnF = PCL_VOTER_FAIL;
    }
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
  }

  else
  {
    valReturnF = PCL_VOTER_FAIL;
  }

  return valReturnF;
}



// MPS E2 Reg B Pneu Out Press < 50

int MPS_PCL_LV10_OP_CMD::G()
{
  int  valReturnG = PCL_VOTER_FAIL;

  if (indicatorBypass(N41P1253A1))
  {
    try
    {
      valReturnG =  (V41P1253A1.getValue() < 50);
    }

    catch(...)
    {
      valReturnG = PCL_VOTER_FAIL;
    }
  }

  else
  {
    valReturnG = PCL_VOTER_FAIL;
  }

  return valReturnG;
}



// MPS Pneu Reg Press > 650

int MPS_PCL_LV10_OP_CMD::H()
{
  int  valReturnH = PCL_VOTER_FAIL;

  if (indicatorBypass(N41P1605A1))
  {
    try
    {
      valReturnH =  (V41P1605A1.getValue() > 650);
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
    }

    catch(...)
    {
      valReturnH = PCL_VOTER_FAIL;
    }
  }

  else
  {
    valReturnH = PCL_VOTER_FAIL;
  }

  return valReturnH;
}




//  LV3 - MPS E2 PNEU REG ISO A SOV - Open

int MPS_PCL_LV10_OP_CMD::I()
{
  int  valReturnI = PCL_VOTER_FAIL;

try
{
            if (NMPSLV3STATE.getValue() == OPEN)
     {
              valReturnI = PCL_VOTER_PASS;
             }
else
            {
       valReturnI = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnI = PCL_VOTER_FAIL;
}

  return valReturnI;
}



//  LV4 - MPS E2 PNEU REG ISO B SOV - Open

int MPS_PCL_LV10_OP_CMD::J()
{
  int  valReturnJ = PCL_VOTER_FAIL;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
try
{
          if (NMPSLV4STATE.getValue() == OPEN)
    {
            valReturnJ = PCL_VOTER_PASS;
           }
else
          {
      valReturnJ = PCL_VOTER_FAIL;
          }
}

catch(...)
{
    valReturnJ = PCL_VOTER_FAIL;
}

 return valReturnJ;
}




// All MPS Solenoid Power - OFF

int MPS_PCL_LV10_OP_CMD::K()
{
 int  valReturnK = PCL_VOTER_FAIL;

 try
{
                if (V41X1506E1.getValue() == OFF &&
                    V41X1505E1.getValue() == OFF &&
                    V41X1508E1.getValue() == OFF &&
                    V41X1507E1.getValue() == OFF &&
                    V41X1807E1.getValue() == OFF &&
                    V41X1806E1.getValue() == OFF &&
                    V41X1808E1.getValue() == OFF &&
                    V41X1809E1.getValue() == OFF &&
                    V41X1133E1.getValue() == OFF &&
                    V41X1233E1.getValue() == OFF &&
                    V41X1333E1.getValue() == OFF &&
                    V41X1145E1.getValue() == OFF &&
                    V41X1245E1.getValue() == OFF &&
                    V41X1345E1.getValue() == OFF &&
                    V41X1132E1.getValue() == OFF &&
                    V41X1232E1.getValue() == OFF &&
                    V41X1332E1.getValue() == OFF &&
                    V41X1144E1.getValue() == OFF &&
                    V41X1244E1.getValue() == OFF &&
                    V41X1344E1.getValue() == OFF &&
                    V41X1582E1.getValue() == OFF &&
                    V41X1549E1.getValue() == OFF &&
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
                    V41X1813E1.getValue() == OFF &&
                    V41X1823E1.getValue() == OFF &&
                    V41X1538E1.getValue() == OFF &&
                    V41X1539E1.getValue() == OFF &&
                    V41X1406E1.getValue() == OFF &&
                    V41X1405E1.getValue() == OFF &&
                    V41X1386E1.getValue() == OFF &&
                    V41X1385E1.getValue() == OFF &&
                    V41X1382E1.getValue() == OFF &&
                    V41X1381E1.getValue() == OFF &&
                    V41X1383E1.getValue() == OFF &&
                    V41X1384E1.getValue() == OFF &&
                    V41X1440E1.getValue() == OFF &&
                    V41X1439E1.getValue() == OFF &&
                    V41X1103E1.getValue() == OFF &&
                    V41X1203E1.getValue() == OFF &&
                    V41X1303E1.getValue() == OFF &&
                    V41X1102E1.getValue() == OFF &&
                    V41X1202E1.getValue() == OFF &&
                    V41X1302E1.getValue() == OFF &&
                    V41X1458E1.getValue() == OFF &&
                    V41X1467E1.getValue() == OFF &&
                    V41X1449E1.getValue() == OFF &&
                    V41X1112E1.getValue() == OFF &&
                    V41X1921E1.getValue() == OFF &&
                    V41X1911E1.getValue() == OFF &&
                    V41X1436E1.getValue() == OFF &&
                    V41X1438E1.getValue() == OFF &&
                    V41X1901E1.getValue() == OFF &&
                    V41X1902E1.getValue() == OFF &&)
          {
            valReturnK = PCL_VOTER_PASS;
           }
else
           {
       valReturnK = PCL_VOTER_FAIL;
           }
}

catch(...)
{
    valReturnK = PCL_VOTER_FAIL;
}

 return valReturnK;
}
```

MPS_PCL_LV10_OP_CMD mps_pcl_lv10_op_cmd(seqName);

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.4.36. MPS_PCL_BLOWDOWN_OP_CMD

3.4.36.1. Description

This PCL provides protection for the MPS Pneumatic GHe Blowdown Valves Open Commands.
This class inherits from the class CAS_Control_Logic. The Algorithm will be implemented directly in C++ Code.

Reference - LPS Program VCE17

3.4.36.1.1. Instance

Instance Name
MPS_PCL_BLOWDOWN_OP_CMD

3.4.36.1.2. PCL Initiating Commands

Command
V41K1631XL
V41K1633XL

3.4.36.1.3. Operations Index

Operation	**Description**
determineState()	This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE INHIBITED ERROR
activate()	This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.
inhibit()	This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.
getState()	This method is inherited from the base class - CAS_Control_Logic
resetCLNotification()	This method is inherited from the base class - CAS_Control_Logic

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.4.36.2. Operations

3.4.36.2.1. getState()

3.4.36.2.1.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.36.2.2. resetCLNotification()

3.4.36.2.2.1. Descriptive Text

This method is inherited from the base class - CAS_Control_Logic

3.4.36.2.3. determineState()

3.4.36.2.3.1. Descriptive Text

This method will determine the enumerated state of the PCL sequence based on the associated commands and will return either: ACTIVE, INHIBITED or ERROR.

3.4.36.2.3.2. Sequence

```
if  (<V41K1631XL>.PCLSTAT = active  AND
    <V41K1633XL>.PCLSTAT = active)
  {
  CLState = ACTIVE;
  }
else
  {
   if  (<V41K1631XL>.PCLSTAT = inhibited  AND
       <V41K1633XL>.PCLSTAT = inhibited)
     {
     CLState = INHIBITED;
      }
    else
      {
     CLState = ERROR;
      }
   }
```

3.4.36.2.4. activate()

3.4.36.2.4.1. Descriptive Text

This method will activate system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.36.2.4.2. Sequence

```
A FD <V41K1631XL> PCL;
A FD <V41K1633XL> PCL;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

return;

3.4.36.2.5. inhibit()

3.4.36.2.5.1. Descriptive Text

This method will inhibit system Prerequisite Control Logic in the CCP for all protected Commands.

3.4.36.2.5.2. Sequence

I FD <V41K1631XL> PCL;
I FD <V41K1633XL> PCL;

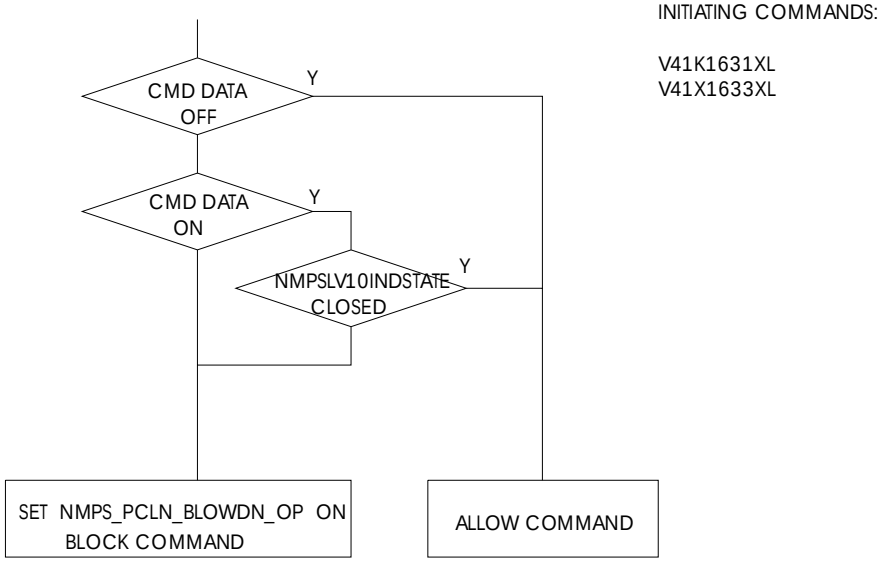
return;

3.4.36.3. Algorithm

3.4.36.3.1. PCL Algorithm

3.4.36.3.1.1. Flowchart

MPS_PCL_BLOWDOWN_OP_CMD



3.4.36.3.1.2. Sequence

#include "mps_pcl_blowdown_op_cmd.h"
#include "SME.h"
#include "LO2.h"
#include "MPS.h"
#include "LH2.h"

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
static const char* seqName = "MPS_PCL_BLOWDOWN_OP_CMD";
static const char* algorithmDescription = "Revision Date - 01/20/2000\n\
if (A  || B || (C && D))\n\
where:   A  = NMPS_PCLO_BLOWDN_OP - PCL Bypass ON\n\
         B =  Command Data - OFF\n\
         C =  Command Data - ON\n\
         D  = LV10 - MPS/E2 PNEU CROSSOVER VALVE - CLOSED";

MPS_PCL_BLOWDOWN_OP_CMD::MPS_PCL_BLOWDOWN_OP_CMD(const char*
seqName):asv_pcsVPCL_DS_ON_OFF_(seqName)
{
  userInit();
}

MPS_PCL_BLOWDOWN_OP_CMD::~MPS_PCL_BLOWDOWN_OP_CMD()
{
}


void MPS_PCL_BLOWDOWN_OP_CMD::userInit(void)
{
  initiatingFD(V41K1631XL);
  initiatingFD(V41K1633XL);
}

int MPS_PCL_BLOWDOWN_OP_CMD::userExecute(Discrete::ON_OFF dsVar)
{
  if (A() || B() || (C() && D()))
  {
   return PCL_ALLOW_CMD;
  }
  else
  {
   NMPS_PCLN_BLOWDN_OP.Set(Discrete::On);
   return PCL_BLOCK_CMD;
  }
}



// PCL Override ON

int MPS_PCL_BLOWDOWN_OP_CMD::A()
{
  int  valReturnA = PCL_VOTER_FAIL;

    try
    {
      valReturnA =  (Discrete::On == (NMPS_PCLO_BLOWDN_OP.getValue()));
    }

    catch(...)
    {
      valReturnA = PCL_VOTER_FAIL;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
    }

  return valReturnA;
}



// COMMAND DATA - OFF

int MPS_PCL_BLOWDOWN_OP_CMD::B()
{
  int  valReturnB = PCL_VOTER_FAIL;

    try
    {
      valReturnB =  (Discrete::Off == dsVar);
    }

    catch(...)
    {
      valReturnB = PCL_VOTER_FAIL;
    }

  return valReturnB;
}



// COMMAND DATA - ON

int MPS_PCL_PD3_BLOWDOWN_CMD::C()
{
  int  valReturnC = PCL_VOTER_FAIL;

    try
    {
      valReturnC =  (Discrete::On == dsVar);
    }

    catch(...)
    {
      valReturnC = PCL_VOTER_FAIL;
    }

  return valReturnC;
}



//LV10 - MPS/E2 PNEU CROSSOVER VALVE - CLOSED
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
int MPS_PCL_BLOWDOWN_OP_CMD::D()
{
  int  valReturnD = PCL_VOTER_FAIL;

try
{
            if (NMPSLV10STATE.getValue() == CLOSED)
     {
              valReturnD = PCL_VOTER_PASS;
             }
else
            {
       valReturnD = PCL_VOTER_FAIL;
            }
}

catch(...)
{
     valReturnD = PCL_VOTER_FAIL;
}

  return valReturnD;
}
```

```
MPS_PCL_BLOWDOWN_OP_CMD mps_pcl_blowdown_op_cmd(seqName);
```

3.5. DATA FUSION SEQUENCES

3.6. TEST CLASSES

3.6.1. MPS Bus Redundancy

3.6.1.1. Description

MPS_Bus_Redundancy class is a base class that will contain the logic for the overall control of the MPS bus redundancy test. The class will be associated with 36 instances of the Bus_Test class, the MPS_GHe_Subsystem, the MPS_LH2_Subsystem, and the MPS_LO2_Subsystem. All commands to the vehicle will reside in the MPS_Bus_Redundancy Class methods. The .

3.6.1.1.1. Instance Index

Instance	Other Users

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.6.1.1.2. Class Diagram

<insert class diagram here>

3.6.1.1.3. Attributes and Aggregations

Attribute	Data Type	Initial Value	Publish / Persistence	Client Access	Description
testID	String				
testDescription	String				
testState	VALVE_STATE_ENUM				
expectedBusTable	BUS_STATE_ENUM				
expectedSwitchTable	SWITCH_STATE_ENUM				
expectedValveTable	array of discrete		PUB		
actualBusTable	PA_SECS		PUB		
actualSwitchTable					
actualValveTable					
commandTable					

3.6.1.1.4. Methods Index

Operation	Description
initialSetUp()	This method
test()	This method will call the appropriate instance of the test class to configure the switches, check the indicators, and notify the operator of required input.
compareBusTables()	This method will be called by the test method, and will compare the expectedBusTable to the actualBusTable. The method will notify the operator of bus configuration and prompt the operator to reconfigure or continue.
compareSwitchTables(seq, step)	This method will call be called by the test method, and will compare the appropriate row of the expectedSwitchTable to the actualSwitchTable. The compareSwitchTable method will prompt the operator with the necessary switch configuration, and then will verify that the proper

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

	switch configuration has been achieved.
compareValveTables(seq)	This method will call be called by the test method, and will compare the appropriate row of the expectedValveTable to the actualValveTable. The compareValveTable method will verify that the proper valve configuration has been achieved.
setCommands(seq)	This method will call be called by the test method, and will set the commands listed in the commandTable.
setUpPneumatics()	This method will set the reg out and accumulator high limits by calling the appropriate instance::method of the MPS_GHe_subsystem. (need an association here.

3.6.1.1.5. Associations

Association	Type	Class	Description
MPS_LH2_Subsystem			
MPS_LO2_Subsystem			
INS(switches???????)			
MPS_GHe_Subsystem			

3.6.1.2. State Description

3.6.1.2.1. State Diagram

<insert state diagram here>

3.6.1.2.2. State Table

3.6.1.3. Operations

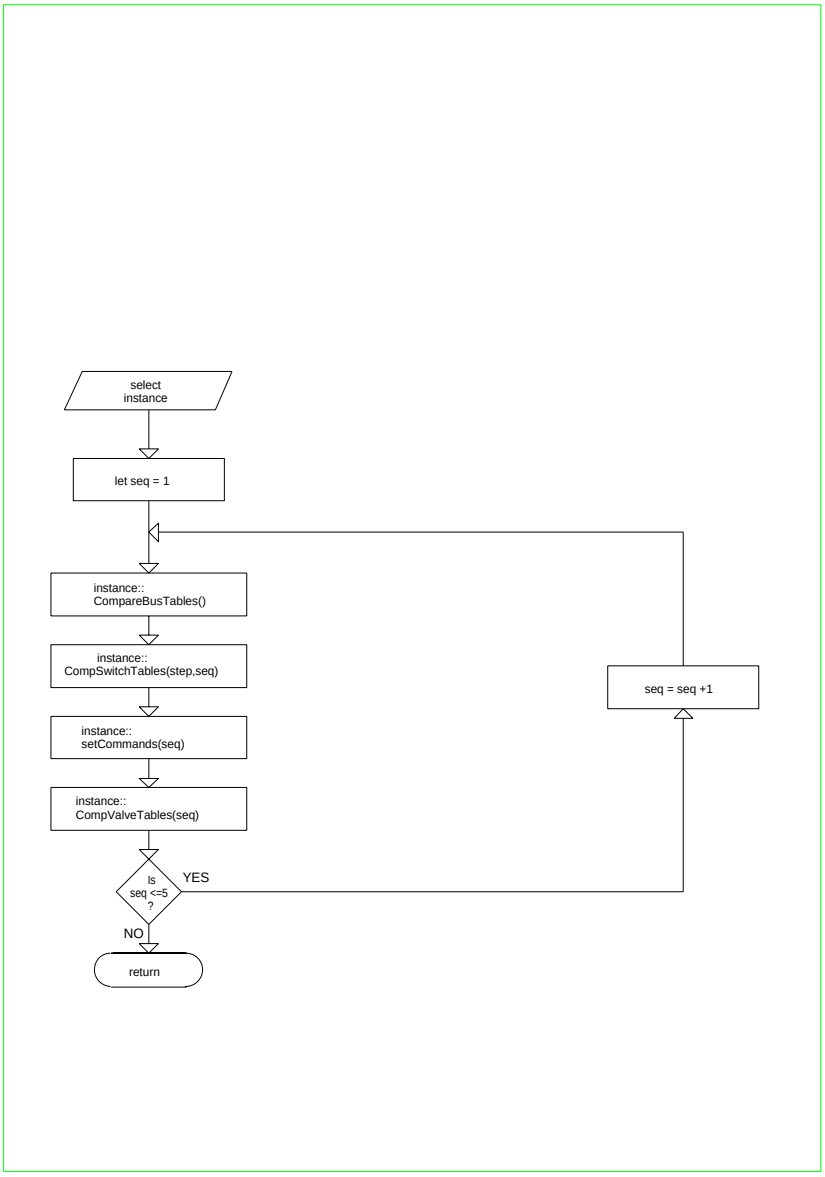
3.6.1.3.1. test()

3.6.1.3.1.1. Descriptive Text

This method will call the appropriate Bus_Test::AB1 methods to configure the switches, check the indicators, and notify the operator of required input.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.6.1.3.1.2. Sequence



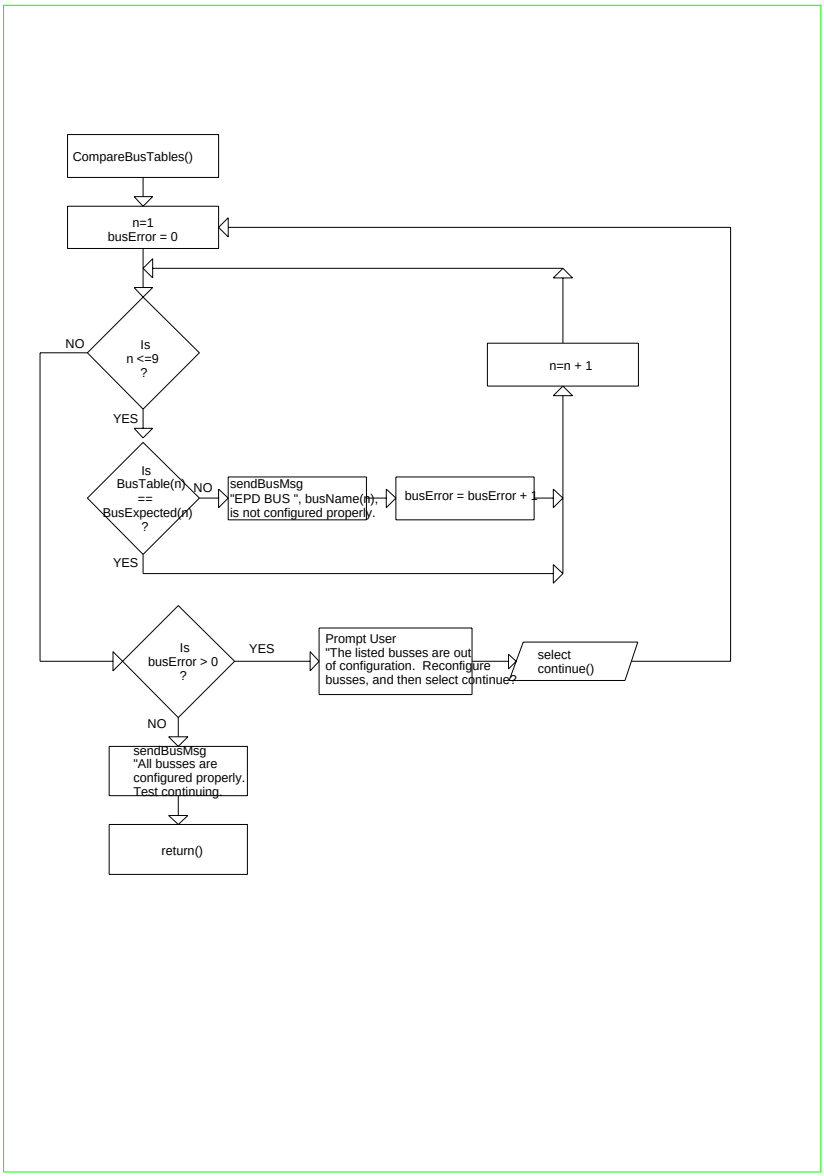
3.6.1.3.2. compareBusTables()

3.6.1.3.2.1. Descriptive Text

This method will be called from the test() method. The method will verify that the busses are configured as defined in the expectedBus attribute. The method inform the operator of any switch configuration problems.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.6.1.3.2.2. Sequence



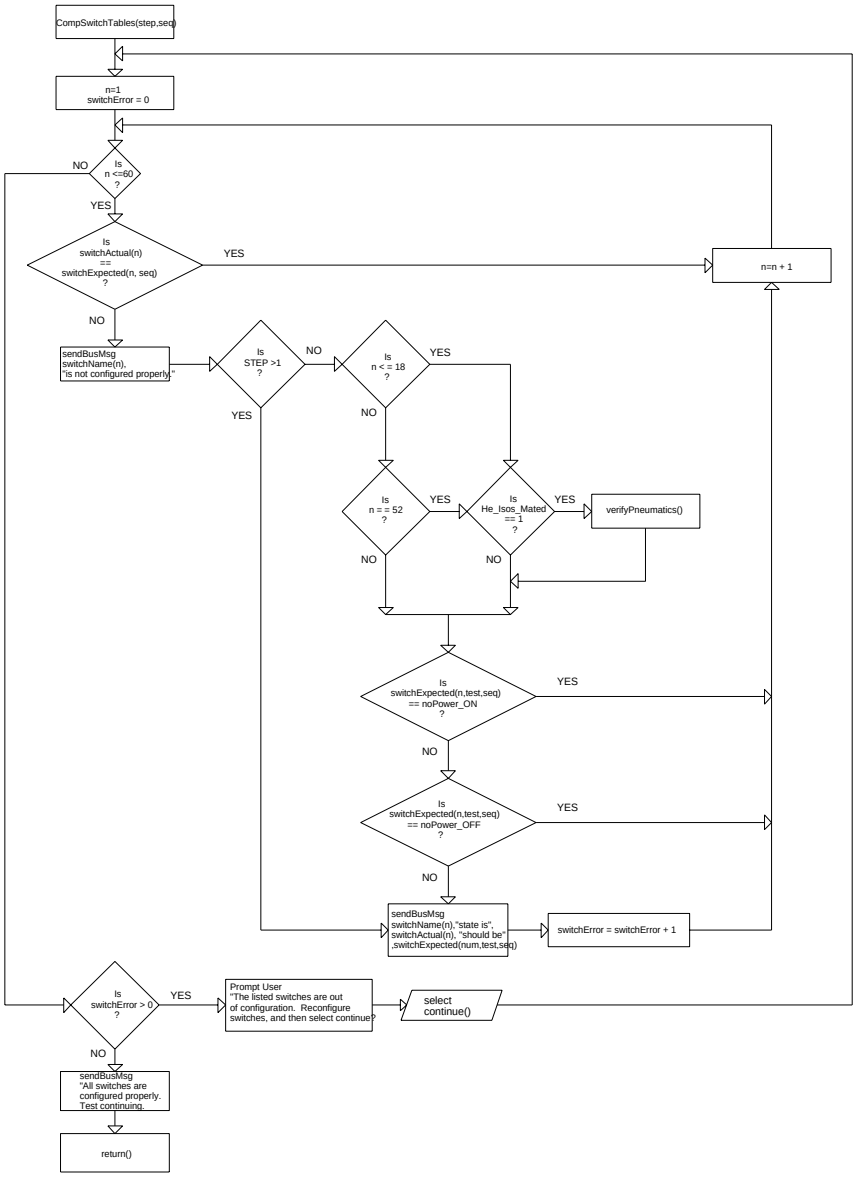
3.6.1.3.3. compareSwitchTables()

3.6.1.3.3.1. Descriptive Text

This method will be called by the test method, and will compare the appropriate row of the expectedSwitchTable to the actualSwitchTable. The compareSwitchTable makes two passes. In the first pass the method compares the actual and expected switch positions and prompts the operator to position the switches that are out of configuration. In the second pass the method verifies that the swithes have been properly configured. If the actual and expected agree then the operator is notified and prompted to continue. If the actual and expected do not agree then the operator is notified of the discrepant switches, and prompted to reconfigure and verify again, to continue with errors, or to abort the current test.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.6.1.3.3.2. Sequence



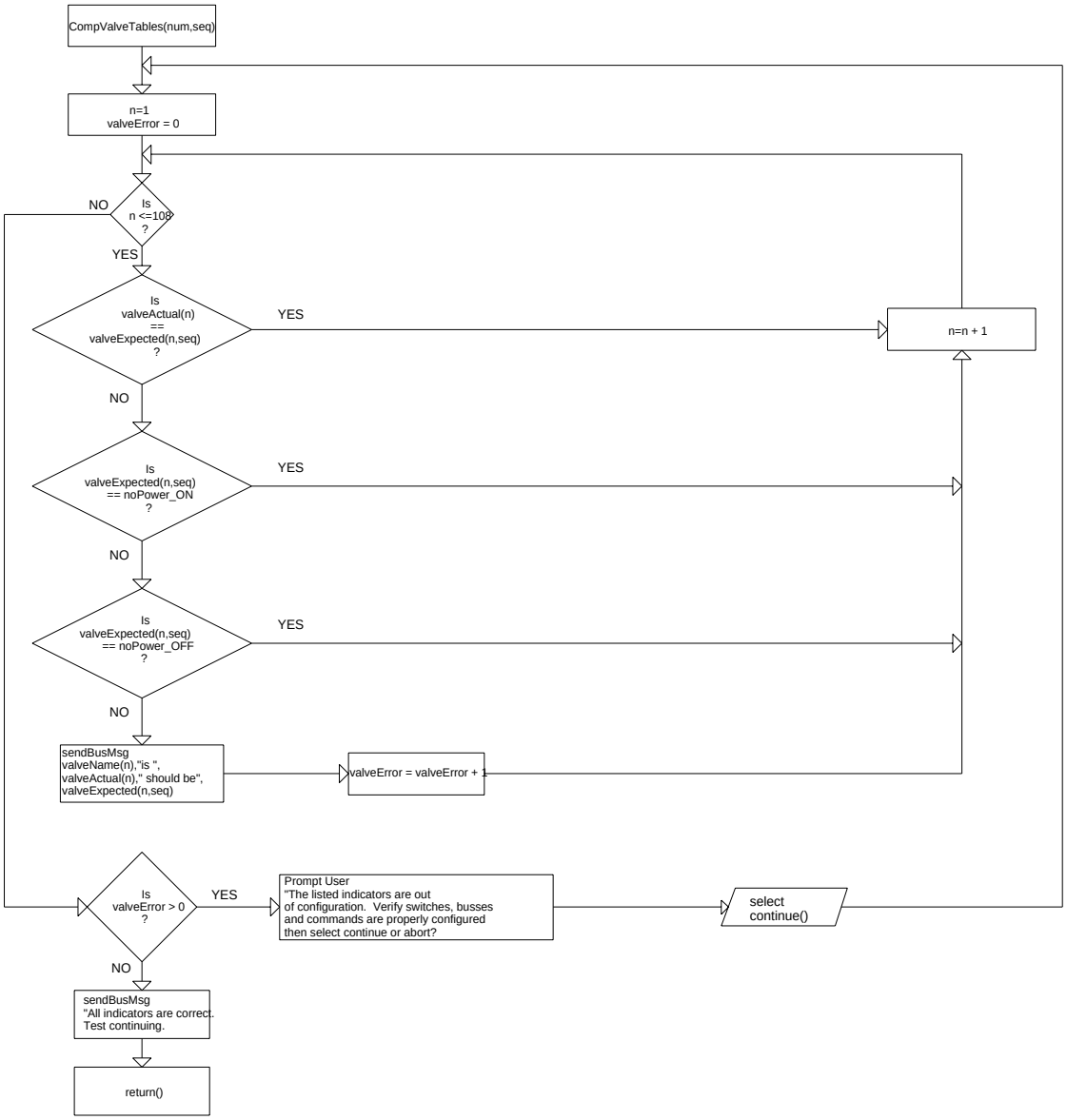
3.6.1.3.4. compareValveTables()

3.6.1.3.4.1. Descriptive Text

This method will be called by the test method, and will compare the appropriate row of the expectedValveTable to the actualValveTable. The compareValveTable prompts the operator if any valve indicators are out of configuration. If the actual and expected agree then the operator is notified and prompted to continue. If the actual and expected do not agree then the operator is notified of the discrepant indicators, and prompted to reconfigure and verify again, to continue with errors, or to abort the current test.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.6.1.3.4.2. Sequence



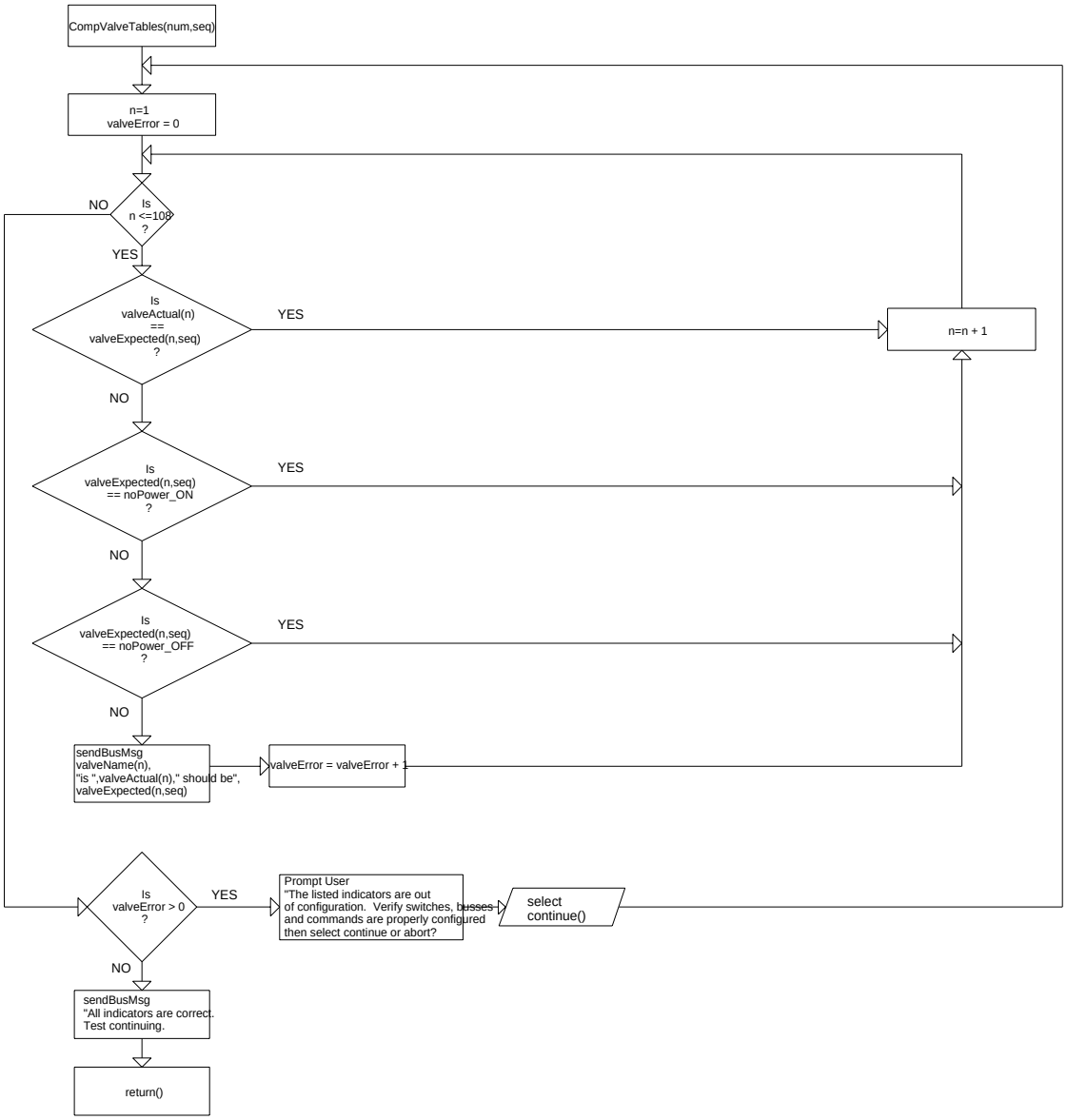
3.6.1.3.5. compareValveTables()

3.6.1.3.5.1. Descriptive Text

This method will be called by the test method, and will compare the appropriate row of the expectedValveTable to the actualValveTable. The compareValveTable prompts the operator if any valve indicators are out of configuration. If the actual and expected agree then the operator is notified and prompted to continue. If the actual and expected do not agree then the operator is notified of the discrepant indicators, and prompted to reconfigure and verify again, to continue with errors, or to abort the current test.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.6.1.3.5.2. Sequence



3.6.1.4. User Interfaces

3.6.1.5. Safety Requirements

3.6.2. LOX MDM Command Redundancy

3.6.2.1. Description

3.6.2.1.1. Instance Index

Instance	Other Users

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.6.2.1.2. Class Diagram

<insert class diagram here>

3.6.2.1.3. Attributes and Aggregations

Attribute	Data Type	Initial Value	Publish / Persistence	Client Access	Description

3.6.2.1.4. Operations Index

Operation	Description

3.6.2.1.5. Associations

Association	Type	Class	Description

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.6.2.2. State Description

3.6.2.2.1. State Table

3.6.2.3. Operations

3.6.2.3.1. Name of Operation

3.6.2.3.1.1. SET COMMANDS / VERIFY INDICATORS

3.6.2.3.1.1.1. AutoCheckout

3.6.2.3.1.1.1.1. Table: SET/VERIFY

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

	V76X4143E1
	V76X4047E1
	V76X4046E1
TEST	
SET COMMANDS ON	VERIFY INDICATOR = ON
V41K1136XL	V41X1145E1
V41K1138XL	V76X4130E1
V41K1236XL	V76X4040E1
V41K1238XL	V41X1245E1
V41K1336XL	V76X4136E1
V41K1338XL	V76X4043E1
	V41X1345E1
	V76X4142E1
	V76X4047E1
TEST	
SET COMMANDS ON	VERIFY INDICATOR = ON
V41K1139XL	V76X4133E1
V41K1239XL	V76X4011E1
V41K1339XL	V76X4139E1
	V76X4014E1
	V76X4145E1
	V76X4016E1
TEST	
SET COMMANDS ON	VERIFY INDICATOR = ON
V41K1139XL	V41X1132E1
V41K1140XL	V41X1144E1
V41K1239XL	V76X4133E1
V41K1240XL	V76X4011E1
V41K1339XL	V41X1232E1
V41K1340XL	V41X1244E1
	V76X4139E1
	V76X4014E1
	V41X1332E1
	V41X1344E1
	V76X4145E1
	V76X4016E1
TEST	
SET COMMANDS ON	VERIFY INDICATOR ALL IND = OFF
V41K1140XL	
V41K1240XL	
V41K1340XL	

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.6.2.3.1.2. Parameters

3.6.2.3.1.3. Effectivity

3.6.2.3.1.4. Prerequisites

3.6.2.3.1.5. Sequence

// autoCommandRedundancy

// Initialize table to be used during Cmd Redundancy checkout

Autocheckout Table:	Should Be	Actual:
<V41X1133E1>	OFF	ON/OFF
<V41X1145E1>	OFF	ON/OFF
<V41S1136E1>	OFF	ON/OFF
<V76X4130E1>	OFF	ON/OFF
<V76X4131E1>	OFF	ON/OFF
<V76X4040E1>	OFF	ON/OFF
<V76X4041E1>	OFF	ON/OFF
<V41X1132E1>	OFF	ON/OFF
<V41X1144E1>	OFF	ON/OFF
<V41S1139E1>	OFF	ON/OFF
<V76X4133E1>	OFF	ON/OFF
<V76X4134E1>	OFF	ON/OFF
<V76X4010E1>	OFF	ON/OFF
<V76X4011E1>	OFF	ON/OFF

// Inhibit EMON monitoring during the checkout

MPS_LO2_System.setPowerDownLimits();

// Perform Command redundancy and stray command/voltage checks

// OMRS V41AB0.240 - PV1 MDM CMD REDUNDANCY

// OMRS V41AB0.240 LINE A

Set <V41K1136XL> to OFF;
Set <V41K1137XL> to OFF;
Set <V41K1138XL> to OFF;
Set <V41K1143XL> to OFF;

ASSIGN Autocheckout Table:

<V41X1133E1>	Column:	Should Be	OFF,
<V41X1145E1>	Column:	Should Be	OFF,
<V41S1136E>	Column:	Should Be	OFF,
<V76X4130E1>	Column:	Should Be	OFF,
<V76X4131E1>	Column:	Should Be	OFF,
<V76X4040E1>	Column:	Should Be	OFF,
<V76X4041E1>	Column:	Should Be	OFF,
<V41X1132E1>	Column:	Should Be	OFF,
<V41X1144E1>	Column:	Should Be	OFF,

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

<V41S1139E1> Column: Should Be OFF,
<V76X4133E1> Column: Should Be OFF,
<V76X4134E1> Column: Should Be OFF,
<V76X4010E1> Column: Should Be OFF,
<V76X4011E1> Column: Should Be OFF,

Delay 2 sec;

Read Autocheckout Table functions and save as Column Actual;

Verify Autocheckout Table Actual equal to Should Be ELSE
  Begin Sequence;
    Record exceptions ‘ <’Measurement’> ‘is’ Actual ‘should be’ Should Be...
      ‘ with <V41K1136XL> set to OFF;
        <V41K1137XL> set to OFF;
        <V41K1138XL> set to OFF;
        <V41K1143XL> set to OFF’ to <PAGE-A>;
    Set AutoCheckoutError = ON;
  End Sequence;

// OMRS V41AB0.240 LINE B

Set <V41K1136XL> to ON;
Set <V41K1137XL> to ON;
Set <V41K1138XL> to OFF;
Set <V41K1143XL> to OFF;

ASSIGN Autocheckout Table:
<V41X1133E1> Column: Should Be ON,
<V41X1145E1> Column: Should Be OFF,
<V41S1136E> Column: Should Be OFF,
<V76X4130E1> Column: Should Be ON,
<V76X4131E1> Column: Should Be OFF,
<V76X4040E1> Column: Should Be ON,
<V76X4041E1> Column: Should Be OFF,
<V41X1132E1> Column: Should Be OFF,
<V41X1144E1> Column: Should Be OFF,
<V41S1139E1> Column: Should Be OFF,
<V76X4133E1> Column: Should Be OFF,
<V76X4134E1> Column: Should Be OFF,
<V76X4010E1> Column: Should Be OFF,
<V76X4011E1> Column: Should Be OFF,

Delay 2 sec;

Read Autocheckout Table functions and save as Column Actual;

Verify Autocheckout Table Actual equal to Should Be ELSE
  Begin Sequence;
    Record exceptions ‘ <’Measurement’> ‘is’ Actual ‘should be’ Should Be...
      ‘ with <V41K1136XL> set to ON;
        <V41K1137XL> set to ON;
        <V41K1138XL> set to OFF;

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
                        <V41K1143XL>  set to OFF'  to  <PAGE-A>;
        Set AutoCheckoutError = ON;
  End Sequence;

//  OMRS V41AB0.240  LINE C

Set <V41K1136XL>  to ON;
Set <V41K1137XL>  to ON;
Set <V41K1138XL>  to ON;
Set <V41K1143XL>  to ON;

ASSIGN Autocheckout Table:
<V41X1133E1>          Column:  Should Be      ON,
<V41X1145E1>          Column:  Should Be      ON,
<V41S1136E>           Column:  Should Be      OFF,
<V76X4130E1>          Column:  Should Be      ON,
<V76X4131E1>          Column:  Should Be      ON,
<V76X4040E1>          Column:  Should Be      ON,
<V76X4041E1>          Column:  Should Be      ON,
<V41X1132E1>          Column:  Should Be      OFF,
<V41X1144E1>          Column:  Should Be      OFF,
<V41S1139E1>          Column:  Should Be      OFF,
<V76X4133E1>          Column:  Should Be      OFF,
<V76X4134E1>          Column:  Should Be      OFF,
<V76X4010E1>          Column:  Should Be      OFF,
<V76X4011E1>          Column:  Should Be      OFF,

Delay 2 sec;

Read Autocheckout Table functions and save as Column Actual;

Verify Autocheckout Table Actual equal to Should Be ELSE
  Begin Sequence;
        Record exceptions '  <'Measurement'>  'is' Actual  'should be'  Should Be...
                ' with <V41K1136XL>  set to ON;
                        <V41K1137XL>  set to ON;
                        <V41K1138XL>  set to ON;
                        <V41K1143XL>  set to ON'  to  <PAGE-A>;
        Set AutoCheckoutError = ON;
  End Sequence;


//  OMRS V41AB0.240  CLEANUP

Set <V41K1136XL>  to OFF;
Set <V41K1137XL>  to OFF;
Set <V41K1138XL>  to OFF;
Set <V41K1143XL>  to OFF;

ASSIGN Autocheckout Table:
<V41X1133E1>          Column:  Should Be      OFF,
<V41X1145E1>          Column:  Should Be      OFF,
<V41S1136E>           Column:  Should Be      OFF,
<V76X4130E1>          Column:  Should Be      OFF,
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
<V76X4131E1>        Column:  Should Be      OFF,
<V76X4040E1>        Column:  Should Be      OFF,
<V76X4041E1>        Column:  Should Be      OFF,
<V41X1132E1>        Column:  Should Be      OFF,
<V41X1144E1>        Column:  Should Be      OFF,
<V41S1139E1>        Column:  Should Be      OFF,
<V76X4133E1>        Column:  Should Be      OFF,
<V76X4134E1>        Column:  Should Be      OFF,
<V76X4010E1>        Column:  Should Be      OFF,
<V76X4011E1>        Column:  Should Be      OFF,

Delay 2 sec;

Read Autocheckout Table functions and save as Column Actual;

Verify Autocheckout Table Actual equal to Should Be ELSE
  Begin Sequence;
        Record exceptions ‘  <’Measurement’>  ‘is’ Actual  ‘should be’  Should Be...
                ‘ with <V41K1136XL>  set to OFF;
                        <V41K1137XL>  set to OFF;
                        <V41K1138XL>  set to OFF;
                        <V41K1143XL>  set to OFF’  to  <PAGE-A>;
        Set AutoCheckoutError = ON;
  End Sequence;



// OMRS V41AB0.240  LINE H

Set <V41K1139XL>  to OFF;
Set <V41K1140XL>  to ON;
Set <V41K1141XL>  to OFF;
Set <V41K1142XL>  to ON;

ASSIGN Autocheckout Table:
<V41X1133E1>        Column:  Should Be      OFF,
<V41X1145E1>        Column:  Should Be      OFF,
<V41S1136E>         Column:  Should Be      OFF,
<V76X4130E1>        Column:  Should Be      OFF,
<V76X4131E1>        Column:  Should Be      OFF,
<V76X4040E1>        Column:  Should Be      OFF,
<V76X4041E1>        Column:  Should Be      OFF,
<V41X1132E1>        Column:  Should Be      ON,
<V41X1144E1>        Column:  Should Be      OFF,
<V41S1139E1>        Column:  Should Be      OFF,
<V76X4133E1>        Column:  Should Be      OFF,
<V76X4134E1>        Column:  Should Be      ON,
<V76X4010E1>        Column:  Should Be      OFF,
<V76X4011E1>        Column:  Should Be      OFF,

Delay 2 sec;

Read Autocheckout Table functions and save as Column Actual;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
Verify Autocheckout Table Actual equal to Should Be ELSE
  Begin Sequence;
        Record exceptions ‘ <’Measurement’> ‘is’ Actual ‘should be’ Should Be...
                ‘ with <V41K1139XL> set to OFF;
                        <V41K1140XL> set to ON;
                        <V41K1141XL> set to OFF;
                        <V41K1142XL> set to ON’ to <PAGE-A>;
        Set AutoCheckoutError = ON;
  End Sequence;



// OMRS V41AB0.240 LINE I

Set <V41K1139XL> to ON;
Set <V41K1140XL> to ON;
Set <V41K1141XL> to ON;
Set <V41K1142XL> to ON;

ASSIGN Autocheckout Table:
<V41X1133E1>        Column: Should Be     OFF,
<V41X1145E1>        Column: Should Be     OFF,
<V41S1136E>         Column: Should Be     OFF,
<V76X4130E1>        Column: Should Be     OFF,
<V76X4131E1>        Column: Should Be     OFF,
<V76X4040E1>        Column: Should Be     OFF,
<V76X4041E1>        Column: Should Be     OFF,
<V41X1132E1>        Column: Should Be     ON,
<V41X1144E1>        Column: Should Be     ON,
<V41S1139E1>        Column: Should Be     OFF,
<V76X4133E1>        Column: Should Be     ON,
<V76X4134E1>        Column: Should Be     ON,
<V76X4010E1>        Column: Should Be     ON,
<V76X4011E1>        Column: Should Be     ON,

Delay 2 sec;

Read Autocheckout Table functions and save as Column Actual;

Verify Autocheckout Table Actual equal to Should Be ELSE
  Begin Sequence;
        Record exceptions ‘ <’Measurement’> ‘is’ Actual ‘should be’ Should Be...
                ‘ with <V41K1139XL> set to ON;
                        <V41K1140XL> set to ON;
                        <V41K1141XL> set to ON;
                        <V41K1142XL> set to ON’ to <PAGE-A>;
        Set AutoCheckoutError = ON;
  End Sequence;


// OMRS V41AB0.240 LINE K

Set <V41K1139XL> to OFF;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

Set <V41K1140XL> to ON;
Set <V41K1141XL> to ON;
Set <V41K1142XL> to OFF;

ASSIGN Autocheckout Table:
<V41X1133E1> Column: Should Be OFF,
<V41X1145E1> Column: Should Be OFF,
<V41S1136E> Column: Should Be OFF,
<V76X4130E1> Column: Should Be OFF,
<V76X4131E1> Column: Should Be OFF,
<V76X4040E1> Column: Should Be OFF,
<V76X4041E1> Column: Should Be OFF,
<V41X1132E1> Column: Should Be OFF,
<V41X1144E1> Column: Should Be ON,
<V41S1139E1> Column: Should Be OFF,
<V76X4133E1> Column: Should Be OFF,
<V76X4134E1> Column: Should Be OFF,
<V76X4010E1> Column: Should Be ON,
<V76X4011E1> Column: Should Be OFF,

Delay 2 sec;

Read Autocheckout Table functions and save as Column Actual;

Verify Autocheckout Table Actual equal to Should Be ELSE
Begin Sequence;
Record exceptions ‘ <’Measurement’> ‘is’ Actual ‘should be’ Should Be...
‘ with <V41K1139XL> set to OFF;
<V41K1140XL> set to ON;
<V41K1141XL> set to ON;
<V41K1142XL> set to OFF’ to <PAGE-A>;
Set AutoCheckoutError = ON;
End Sequence;

// OMRS V41AB0.240 CLEANUP

Set <V41K1139XL> to OFF;
Set <V41K1140XL> to OFF;
Set <V41K1141XL> to OFF;
Set <V41K1142XL> to OFF;

ASSIGN Autocheckout Table:
<V41X1133E1> Column: Should Be OFF,
<V41X1145E1> Column: Should Be OFF,
<V41S1136E> Column: Should Be OFF,
<V76X4130E1> Column: Should Be OFF,
<V76X4131E1> Column: Should Be OFF,
<V76X4040E1> Column: Should Be OFF,
<V76X4041E1> Column: Should Be OFF,
<V41X1132E1> Column: Should Be OFF,
<V41X1144E1> Column: Should Be OFF,
<V41S1139E1> Column: Should Be OFF,
<V76X4133E1> Column: Should Be OFF,

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

<V76X4134E1> Column: Should Be OFF,
<V76X4010E1> Column: Should Be OFF,
<V76X4011E1> Column: Should Be OFF,

Delay 2 sec;

Read Autocheckout Table functions and save as Column Actual;

Verify Autocheckout Table Actual equal to Should Be ELSE
Begin Sequence;
Record exceptions ‘ <’Measurement’> ‘is’ Actual ‘should be’ Should Be...
‘ with <V41K1139XL> set to OFF;
<V41K1140XL> set to OFF;
<V41K1141XL> set to OFF;
<V41K1142XL> set to OFF’ to <PAGE-A>;
Set AutoCheckoutError = ON;
End Sequence;

// OMRS V41AE0.380 - PV1 MDM CMD REDUNDANCY

// OMRS V41AE0.380 LINE A

Set <V41K1136XL> to ON;
Set <V41K1137XL> to OFF;
Set <V41K1138XL> to OFF;
Set <V41K1143XL> to OFF;

ASSIGN Autocheckout Table:
<V41X1133E1> Column: Should Be OFF,
<V41X1145E1> Column: Should Be OFF,
<V41S1136E> Column: Should Be OFF,
<V76X4130E1> Column: Should Be ON,
<V76X4131E1> Column: Should Be OFF,
<V76X4040E1> Column: Should Be ON,
<V76X4041E1> Column: Should Be OFF,
<V41X1132E1> Column: Should Be OFF,
<V41X1144E1> Column: Should Be OFF,
<V41S1139E1> Column: Should Be OFF,
<V76X4133E1> Column: Should Be OFF,
<V76X4134E1> Column: Should Be OFF,
<V76X4010E1> Column: Should Be OFF,
<V76X4011E1> Column: Should Be OFF,

Delay 2 sec;

Read Autocheckout Table functions and save as Column Actual;

Verify Autocheckout Table Actual equal to Should Be ELSE
Begin Sequence;
Record exceptions ‘ <’Measurement’> ‘is’ Actual ‘should be’ Should Be...
‘ with <V41K1136XL> set to ON;
<V41K1137XL> set to OFF;
<V41K1138XL> set to OFF;

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
                        <V41K1143XL>  set to OFF'  to  <PAGE-A>;
        Set AutoCheckoutError = ON;
  End Sequence;


//  OMRS V41AE0.380  LINE B

Set <V41K1136XL>  to ON;
Set <V41K1137XL>  to ON;
Set <V41K1138XL>  to OFF;
Set <V41K1143XL>  to OFF;

ASSIGN Autocheckout Table:
<V41X1133E1>          Column:  Should Be      ON,
<V41X1145E1>          Column:  Should Be      OFF,
<V41S1136E>           Column:  Should Be      OFF,
<V76X4130E1>          Column:  Should Be      ON,
<V76X4131E1>          Column:  Should Be      OFF,
<V76X4040E1>          Column:  Should Be      ON,
<V76X4041E1>          Column:  Should Be      OFF,
<V41X1132E1>          Column:  Should Be      OFF,
<V41X1144E1>          Column:  Should Be      OFF,
<V41S1139E1>          Column:  Should Be      OFF,
<V76X4133E1>          Column:  Should Be      OFF,
<V76X4134E1>          Column:  Should Be      OFF,
<V76X4010E1>          Column:  Should Be      OFF,
<V76X4011E1>          Column:  Should Be      OFF,

Delay 2 sec;

Read Autocheckout Table functions and save as Column Actual;

Verify Autocheckout Table Actual equal to Should Be ELSE
  Begin Sequence;
        Record exceptions '  <'Measurement'>  'is' Actual  'should be'  Should Be...
                ' with <V41K1136XL>  set to ON;
                        <V41K1137XL>  set to ON;
                        <V41K1138XL>  set to OFF;
                        <V41K1143XL>  set to OFF'  to  <PAGE-A>;
        Set AutoCheckoutError = ON;
  End Sequence;

//  OMRS V41AE0.380  LINE C

Set <V41K1136XL>  to OFF;
Set <V41K1137XL>  to ON;
Set <V41K1138XL>  to OFF;
Set <V41K1143XL>  to OFF;

ASSIGN Autocheckout Table:
<V41X1133E1>          Column:  Should Be      OFF,
<V41X1145E1>          Column:  Should Be      OFF,
<V41S1136E>           Column:  Should Be      OFF,
<V76X4130E1>          Column:  Should Be      OFF,
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

<V76X4131E1> Column: Should Be OFF,
<V76X4040E1> Column: Should Be OFF,
<V76X4041E1> Column: Should Be OFF,
<V41X1132E1> Column: Should Be OFF,
<V41X1144E1> Column: Should Be OFF,
<V41S1139E1> Column: Should Be OFF,
<V76X4133E1> Column: Should Be OFF,
<V76X4134E1> Column: Should Be OFF,
<V76X4010E1> Column: Should Be OFF,
<V76X4011E1> Column: Should Be OFF,

Delay 2 sec;

Read Autocheckout Table functions and save as Column Actual;

Verify Autocheckout Table Actual equal to Should Be ELSE
Begin Sequence;
Record exceptions ‘ <’Measurement’> ‘is’ Actual ‘should be’ Should Be...
‘ with <V41K1136XL> set to OFF
<V41K1137XL> set to ON
<V41K1138XL> set to OFF
<V41K1143XL> set to OFF’ to <PAGE-A>;
Set AutoCheckoutError = ON;
End Sequence;

// OMRS V41AE0.380 LINE D

Set <V41K1136XL> to OFF;
Set <V41K1137XL> to ON;
Set <V41K1138XL> to ON;
Set <V41K1143XL> to OFF;

ASSIGN Autocheckout Table:
<V41X1133E1> Column: Should Be OFF,
<V41X1145E1> Column: Should Be OFF,
<V41S1136E> Column: Should Be OFF,
<V76X4130E1> Column: Should Be OFF,
<V76X4131E1> Column: Should Be OFF,
<V76X4040E1> Column: Should Be OFF,
<V76X4041E1> Column: Should Be OFF,
<V41X1132E1> Column: Should Be OFF,
<V41X1144E1> Column: Should Be OFF,
<V41S1139E1> Column: Should Be OFF,
<V76X4133E1> Column: Should Be OFF,
<V76X4134E1> Column: Should Be OFF,
<V76X4010E1> Column: Should Be OFF,
<V76X4011E1> Column: Should Be OFF,

Delay 2 sec;

Read Autocheckout Table functions and save as Column Actual;

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
Verify Autocheckout Table Actual equal to Should Be ELSE
  Begin Sequence;
        Record exceptions ‘ <’Measurement’> ‘is’ Actual ‘should be’ Should Be...
                ‘ with <V41K1136XL> set to OFF
                        <V41K1137XL> set to ON
                        <V41K1138XL> set to ON
                        <V41K1143XL> set to OFF’ to <PAGE-A>;
        Set AutoCheckoutError = ON;
  End Sequence;



//  OMRS V41AE0.380  LINE E

Set <V41K1136XL> to OFF;
Set <V41K1137XL> to OFF;
Set <V41K1138XL> to ON;
Set <V41K1143XL> to OFF;

ASSIGN Autocheckout Table:
<V41X1133E1>        Column: Should Be       OFF,
<V41X1145E1>        Column: Should Be       OFF,
<V41S1136E>         Column: Should Be       OFF,
<V76X4130E1>        Column: Should Be       OFF,
<V76X4131E1>        Column: Should Be       OFF,
<V76X4040E1>        Column: Should Be       OFF,
<V76X4041E1>        Column: Should Be       OFF,
<V41X1132E1>        Column: Should Be       OFF,
<V41X1144E1>        Column: Should Be       OFF,
<V41S1139E1>        Column: Should Be       OFF,
<V76X4133E1>        Column: Should Be       OFF,
<V76X4134E1>        Column: Should Be       OFF,
<V76X4010E1>        Column: Should Be       OFF,
<V76X4011E1>        Column: Should Be       OFF,

Delay 2 sec;

Read Autocheckout Table functions and save as Column Actual;

Verify Autocheckout Table Actual equal to Should Be ELSE
  Begin Sequence;
        Record exceptions ‘ <’Measurement’> ‘is’ Actual ‘should be’ Should Be...
                ‘ with <V41K1136XL> set to OFF
                        <V41K1137XL> set to OFF
                        <V41K1138XL> set to ON
                        <V41K1143XL> set to OFF’ to <PAGE-A>;
        Set AutoCheckoutError = ON;
  End Sequence;



//  OMRS V41AE0.380  LINE F
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

Set <V41K1136XL> to OFF;
Set <V41K1137XL> to OFF;
Set <V41K1138XL> to ON;
Set <V41K1143XL> to ON;

ASSIGN Autocheckout Table:

<V41X1133E1>	Column:	Should Be	ON,
<V41X1145E1>	Column:	Should Be	OFF,
<V41S1136E>	Column:	Should Be	OFF,
<V76X4130E1>	Column:	Should Be	OFF,
<V76X4131E1>	Column:	Should Be	ON,
<V76X4040E1>	Column:	Should Be	OFF,
<V76X4041E1>	Column:	Should Be	ON,
<V41X1132E1>	Column:	Should Be	OFF,
<V41X1144E1>	Column:	Should Be	OFF,
<V41S1139E1>	Column:	Should Be	OFF,
<V76X4133E1>	Column:	Should Be	OFF,
<V76X4134E1>	Column:	Should Be	OFF,
<V76X4010E1>	Column:	Should Be	OFF,
<V76X4011E1>	Column:	Should Be	OFF,

Delay 2 sec;

Read Autocheckout Table functions and save as Column Actual;

Verify Autocheckout Table Actual equal to Should Be ELSE
  Begin Sequence;
    Record exceptions ‘ <’Measurement’> ‘is’ Actual ‘should be’ Should Be...
      ‘ with <V41K1136XL> set to OFF
        <V41K1137XL> set to OFF
        <V41K1138XL> set to ON
        <V41K1143XL> set to ON’ to <PAGE-A>;
    Set AutoCheckoutError = ON;
  End Sequence;

// OMRS V41AE0.380 LINE G

Set <V41K1136XL> to OFF;
Set <V41K1137XL> to OFF;
Set <V41K1138XL> to OFF;
Set <V41K1143XL> to ON;

ASSIGN Autocheckout Table:

<V41X1133E1>	Column:	Should Be	OFF,
<V41X1145E1>	Column:	Should Be	OFF,
<V41S1136E>	Column:	Should Be	OFF,
<V76X4130E1>	Column:	Should Be	OFF,
<V76X4131E1>	Column:	Should Be	ON,
<V76X4040E1>	Column:	Should Be	OFF,
<V76X4041E1>	Column:	Should Be	ON,
<V41X1132E1>	Column:	Should Be	OFF,
<V41X1144E1>	Column:	Should Be	OFF,

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

<V41S1139E1> Column: Should Be OFF,
<V76X4133E1> Column: Should Be OFF,
<V76X4134E1> Column: Should Be OFF,
<V76X4010E1> Column: Should Be OFF,
<V76X4011E1> Column: Should Be OFF,

Delay 2 sec;

Read Autocheckout Table functions and save as Column Actual;

Verify Autocheckout Table Actual equal to Should Be ELSE
Begin Sequence;
Record exceptions ' <'Measurement'> 'is' Actual 'should be' Should Be...
' with <V41K1136XL> set to OFF
<V41K1137XL> set to OFF
<V41K1138XL> set to OFF
<V41K1143XL> set to ON' to <PAGE-A>;
Set AutoCheckoutError = ON;
End Sequence;

// OMRS V41AE0.380 LINE H

Set <V41K1136XL> to OFF;
Set <V41K1137XL> to ON;
Set <V41K1138XL> to OFF;
Set <V41K1143XL> to ON;

ASSIGN Autocheckout Table:
<V41X1133E1> Column: Should Be OFF,
<V41X1145E1> Column: Should Be ON,
<V41S1136E> Column: Should Be OFF,
<V76X4130E1> Column: Should Be OFF,
<V76X4131E1> Column: Should Be ON,
<V76X4040E1> Column: Should Be OFF,
<V76X4041E1> Column: Should Be ON,
<V41X1132E1> Column: Should Be OFF,
<V41X1144E1> Column: Should Be OFF,
<V41S1139E1> Column: Should Be OFF,
<V76X4133E1> Column: Should Be OFF,
<V76X4134E1> Column: Should Be OFF,
<V76X4010E1> Column: Should Be OFF,
<V76X4011E1> Column: Should Be OFF,

Delay 2 sec;

Read Autocheckout Table functions and save as Column Actual;

Verify Autocheckout Table Actual equal to Should Be ELSE
Begin Sequence;
Record exceptions ' <'Measurement'> 'is' Actual 'should be' Should Be...
' with <V41K1136XL> set to OFF
<V41K1137XL> set to ON

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
                        <V41K1138XL>  set to OFF
                        <V41K1143XL>  set to ON'  to  <PAGE-A>;
        Set AutoCheckoutError = ON;
  End Sequence;



//  OMRS V41AE0.380  LINE I

Set <V41K1136XL>  to ON;
Set <V41K1137XL>  to OFF;
Set <V41K1138XL>  to OFF;
Set <V41K1143XL>  to ON;

ASSIGN Autocheckout Table:
<V41X1133E1>          Column:  Should Be      OFF,
<V41X1145E1>          Column:  Should Be      OFF,
<V41S1136E>           Column:  Should Be      OFF,
<V76X4130E1>          Column:  Should Be      ON,
<V76X4131E1>          Column:  Should Be      ON,
<V76X4040E1>          Column:  Should Be      ON,
<V76X4041E1>          Column:  Should Be      ON,
<V41X1132E1>          Column:  Should Be      OFF,
<V41X1144E1>          Column:  Should Be      OFF,
<V41S1139E1>          Column:  Should Be      OFF,
<V76X4133E1>          Column:  Should Be      OFF,
<V76X4134E1>          Column:  Should Be      OFF,
<V76X4010E1>          Column:  Should Be      OFF,
<V76X4011E1>          Column:  Should Be      OFF,

Delay 2 sec;

Read Autocheckout Table functions and save as Column Actual;

Verify Autocheckout Table Actual equal to Should Be ELSE
  Begin Sequence;
        Record exceptions '  <'Measurement'>  'is' Actual  'should be'  Should Be...
                ' with <V41K1136XL>  set to ON
                        <V41K1137XL>  set to OFF
                        <V41K1138XL>  set to OFF
                        <V41K1143XL>  set to ON'  to  <PAGE-A>;
        Set AutoCheckoutError = ON;
  End Sequence;



//  OMRS V41AE0.380  LINE J

Set <V41K1136XL>  to ON;
Set <V41K1137XL>  to OFF;
Set <V41K1138XL>  to ON;
Set <V41K1143XL>  to OFF;

ASSIGN Autocheckout Table:
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

<V41X1133E1> Column: Should Be OFF,
<V41X1145E1> Column: Should Be ON,
<V41S1136E> Column: Should Be OFF,
<V76X4130E1> Column: Should Be ON,
<V76X4131E1> Column: Should Be OFF,
<V76X4040E1> Column: Should Be ON,
<V76X4041E1> Column: Should Be OFF,
<V41X1132E1> Column: Should Be OFF,
<V41X1144E1> Column: Should Be OFF,
<V41S1139E1> Column: Should Be OFF,
<V76X4133E1> Column: Should Be OFF,
<V76X4134E1> Column: Should Be OFF,
<V76X4010E1> Column: Should Be OFF,
<V76X4011E1> Column: Should Be OFF,

Delay 2 sec;

Read Autocheckout Table functions and save as Column Actual;

Verify Autocheckout Table Actual equal to Should Be ELSE
Begin Sequence;
Record exceptions ‘ <’Measurement’> ‘is’ Actual ‘should be’ Should Be...
‘ with <V41K1136XL> set to ON
<V41K1137XL> set to OFF
<V41K1138XL> set to ON
<V41K1143XL> set to OFF’ to <PAGE-A>;
Set AutoCheckoutError = ON;
End Sequence;

// OMRS V41AB0.240 CLEANUP

Set <V41K1136XL> to OFF;
Set <V41K1137XL> to OFF;
Set <V41K1138XL> to OFF;
Set <V41K1143XL> to OFF;

ASSIGN Autocheckout Table:
<V41X1133E1> Column: Should Be OFF,
<V41X1145E1> Column: Should Be OFF,
<V41S1136E> Column: Should Be OFF,
<V76X4130E1> Column: Should Be OFF,
<V76X4131E1> Column: Should Be OFF,
<V76X4040E1> Column: Should Be OFF,
<V76X4041E1> Column: Should Be OFF,
<V41X1132E1> Column: Should Be OFF,
<V41X1144E1> Column: Should Be OFF,
<V41S1139E1> Column: Should Be OFF,
<V76X4133E1> Column: Should Be OFF,
<V76X4134E1> Column: Should Be OFF,
<V76X4010E1> Column: Should Be OFF,
<V76X4011E1> Column: Should Be OFF,

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

Delay 2 sec;

Read Autocheckout Table functions and save as Column Actual;

Verify Autocheckout Table Actual equal to Should Be ELSE
  Begin Sequence;
        Record exceptions ‘ <’Measurement’> ‘is’ Actual ‘should be’ Should Be...
                ‘ with <V41K1136XL> set to OFF;
                        <V41K1137XL> set to OFF;
                        <V41K1138XL> set to OFF;
                        <V41K1143XL> set to OFF’ to <PAGE-A>;
        Set AutoCheckoutError = ON;
  End Sequence;

// OMRS V41AE0.380 LINE K

Set <V41K1139XL> to ON;
Set <V41K1140XL> to OFF;
Set <V41K1141XL> to OFF;
Set <V41K1142XL> to OFF;

ASSIGN Autocheckout Table:

<V41X1133E1>	Column:	Should Be	OFF,
<V41X1145E1>	Column:	Should Be	OFF,
<V41S1136E>	Column:	Should Be	OFF,
<V76X4130E1>	Column:	Should Be	OFF,
<V76X4131E1>	Column:	Should Be	OFF,
<V76X4040E1>	Column:	Should Be	OFF,
<V76X4041E1>	Column:	Should Be	OFF,
<V41X1132E1>	Column:	Should Be	OFF,
<V41X1144E1>	Column:	Should Be	OFF,
<V41S1139E1>	Column:	Should Be	OFF,
<V76X4133E1>	Column:	Should Be	ON,
<V76X4134E1>	Column:	Should Be	OFF,
<V76X4010E1>	Column:	Should Be	OFF,
<V76X4011E1>	Column:	Should Be	ON,

Delay 2 sec;

Read Autocheckout Table functions and save as Column Actual;

Verify Autocheckout Table Actual equal to Should Be ELSE
  Begin Sequence;
        Record exceptions ‘ <’Measurement’> ‘is’ Actual ‘should be’ Should Be...
                ‘ with <V41K1139XL> set to ON;
                        <V41K1140XL> set to OFF;
                        <V41K1141XL> set to OFF;
                        <V41K1142XL> set to OFF’ to <PAGE-A>;
        Set AutoCheckoutError = ON;
  End Sequence;

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
// OMRS V41AE0.380 LINE L

Set <V41K1139XL> to ON;
Set <V41K1140XL> to ON;
Set <V41K1141XL> to OFF;
Set <V41K1142XL> to OFF;

ASSIGN Autocheckout Table:
<V41X1133E1>        Column: Should Be    OFF,
<V41X1145E1>        Column: Should Be    OFF,
<V41S1136E>         Column: Should Be    OFF,
<V76X4130E1>        Column: Should Be    OFF,
<V76X4131E1>        Column: Should Be    OFF,
<V76X4040E1>        Column: Should Be    OFF,
<V76X4041E1>        Column: Should Be    OFF,
<V41X1132E1>        Column: Should Be    ON,
<V41X1144E1>        Column: Should Be    ON,
<V41S1139E1>        Column: Should Be    OFF,
<V76X4133E1>        Column: Should Be    ON,
<V76X4134E1>        Column: Should Be    OFF,
<V76X4010E1>        Column: Should Be    OFF,
<V76X4011E1>        Column: Should Be    ON,

Delay 2 sec;

Read Autocheckout Table functions and save as Column Actual;

Verify Autocheckout Table Actual equal to Should Be ELSE
  Begin Sequence;
        Record exceptions ' <'Measurement'> 'is' Actual 'should be' Should Be...
                ' with <V41K1139XL> set to ON;
                       <V41K1140XL> set to ON;
                       <V41K1141XL> set to OFF;
                       <V41K1142XL> set to OFF' to <PAGE-A>;
        Set AutoCheckoutError = ON;
  End Sequence;


// OMRS V41AE0.380 LINE M

Set <V41K1139XL> to OFF;
Set <V41K1140XL> to ON;
Set <V41K1141XL> to OFF;
Set <V41K1142XL> to OFF;

ASSIGN Autocheckout Table:
<V41X1133E1>        Column: Should Be    OFF,
<V41X1145E1>        Column: Should Be    OFF,
<V41S1136E>         Column: Should Be    OFF,
<V76X4130E1>        Column: Should Be    OFF,
<V76X4131E1>        Column: Should Be    OFF,
<V76X4040E1>        Column: Should Be    OFF,
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

<V76X4041E1> Column: Should Be OFF,
<V41X1132E1> Column: Should Be OFF,
<V41X1144E1> Column: Should Be OFF,
<V41S1139E1> Column: Should Be OFF,
<V76X4133E1> Column: Should Be OFF,
<V76X4134E1> Column: Should Be OFF,
<V76X4010E1> Column: Should Be OFF,
<V76X4011E1> Column: Should Be OFF,

Delay 2 sec;

Read Autocheckout Table functions and save as Column Actual;

Verify Autocheckout Table Actual equal to Should Be ELSE
Begin Sequence;
Record exceptions ‘ <’Measurement’> ‘is’ Actual ‘should be’ Should Be...
‘ with <V41K1139XL> set to OFF;
<V41K1140XL> set to ON;
<V41K1141XL> set to OFF;
<V41K1142XL> set to OFF’ to <PAGE-A>;
Set AutoCheckoutError = ON;
End Sequence;

// OMRS V41AE0.380 LINE N

Set <V41K1139XL> to OFF;
Set <V41K1140XL> to ON;
Set <V41K1141XL> to ON;
Set <V41K1142XL> to OFF;

ASSIGN Autocheckout Table:
<V41X1133E1> Column: Should Be OFF,
<V41X1145E1> Column: Should Be OFF,
<V41S1136E> Column: Should Be OFF,
<V76X4130E1> Column: Should Be OFF,
<V76X4131E1> Column: Should Be OFF,
<V76X4040E1> Column: Should Be OFF,
<V76X4041E1> Column: Should Be OFF,
<V41X1132E1> Column: Should Be OFF,
<V41X1144E1> Column: Should Be ON,
<V41S1139E1> Column: Should Be OFF,
<V76X4133E1> Column: Should Be OFF,
<V76X4134E1> Column: Should Be OFF,
<V76X4010E1> Column: Should Be ON,
<V76X4011E1> Column: Should Be OFF,

Delay 2 sec;

Read Autocheckout Table functions and save as Column Actual;

Verify Autocheckout Table Actual equal to Should Be ELSE
Begin Sequence;

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
        Record exceptions ‘ <’Measurement’> ‘is’ Actual ‘should be’ Should Be...
                ‘ with <V41K1139XL> set to OFF;
                        <V41K1140XL> set to ON;
                        <V41K1141XL> set to ON;
                        <V41K1142XL> set to OFF’ to <PAGE-A>;
        Set AutoCheckoutError = ON;
  End Sequence;


// OMRS V41AE0.380 LINE O

Set <V41K1139XL> to OFF;
Set <V41K1140XL> to OFF;
Set <V41K1141XL> to ON;
Set <V41K1142XL> to OFF;

ASSIGN Autocheckout Table:
<V41X1133E1>        Column: Should Be       OFF,
<V41X1145E1>        Column: Should Be       OFF,
<V41S1136E>         Column: Should Be       OFF,
<V76X4130E1>        Column: Should Be       OFF,
<V76X4131E1>        Column: Should Be       OFF,
<V76X4040E1>        Column: Should Be       OFF,
<V76X4041E1>        Column: Should Be       OFF,
<V41X1132E1>        Column: Should Be       OFF,
<V41X1144E1>        Column: Should Be       OFF,
<V41S1139E1>        Column: Should Be       OFF,
<V76X4133E1>        Column: Should Be       OFF,
<V76X4134E1>        Column: Should Be       OFF,
<V76X4010E1>        Column: Should Be       ON,
<V76X4011E1>        Column: Should Be       OFF,

Delay 2 sec;

Read Autocheckout Table functions and save as Column Actual;

Verify Autocheckout Table Actual equal to Should Be ELSE
  Begin Sequence;
        Record exceptions ‘ <’Measurement’> ‘is’ Actual ‘should be’ Should Be...
                ‘ with <V41K1139XL> set to OFF;
                        <V41K1140XL> set to OFF;
                        <V41K1141XL> set to ON;
                        <V41K1142XL> set to OFF’ to <PAGE-A>;
        Set AutoCheckoutError = ON;
  End Sequence;


// OMRS V41AE0.380 LINE P

Set <V41K1139XL> to OFF;
Set <V41K1140XL> to OFF;
Set <V41K1141XL> to ON;
Set <V41K1142XL> to ON;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
ASSIGN Autocheckout Table:
<V41X1133E1>        Column:  Should Be      OFF,
<V41X1145E1>        Column:  Should Be      OFF,
<V41S1136E>         Column:  Should Be      OFF,
<V76X4130E1>        Column:  Should Be      OFF,
<V76X4131E1>        Column:  Should Be      OFF,
<V76X4040E1>        Column:  Should Be      OFF,
<V76X4041E1>        Column:  Should Be      OFF,
<V41X1132E1>        Column:  Should Be      ON,
<V41X1144E1>        Column:  Should Be      ON,
<V41S1139E1>        Column:  Should Be      OFF,
<V76X4133E1>        Column:  Should Be      OFF,
<V76X4134E1>        Column:  Should Be      ON,
<V76X4010E1>        Column:  Should Be      ON,
<V76X4011E1>        Column:  Should Be      OFF,

Delay 2 sec;

Read Autocheckout Table functions and save as Column Actual;

Verify Autocheckout Table Actual equal to Should Be ELSE
  Begin Sequence;
        Record exceptions ‘  <’Measurement’>  ‘is’ Actual  ‘should be’  Should Be...
                ‘ with <V41K1139XL>  set to OFF;
                        <V41K1140XL>  set to OFF;
                        <V41K1141XL>  set to ON;
                        <V41K1142XL>  set to ON’  to  <PAGE-A>;
        Set AutoCheckoutError = ON;
  End Sequence;


//  OMRS V41AE0.380  LINE Q

Set <V41K1139XL>  to OFF;
Set <V41K1140XL>  to OFF;
Set <V41K1141XL>  to OFF;
Set <V41K1142XL>  to ON;

ASSIGN Autocheckout Table:
<V41X1133E1>        Column:  Should Be      OFF,
<V41X1145E1>        Column:  Should Be      OFF,
<V41S1136E>         Column:  Should Be      OFF,
<V76X4130E1>        Column:  Should Be      OFF,
<V76X4131E1>        Column:  Should Be      OFF,
<V76X4040E1>        Column:  Should Be      OFF,
<V76X4041E1>        Column:  Should Be      OFF,
<V41X1132E1>        Column:  Should Be      OFF,
<V41X1144E1>        Column:  Should Be      OFF,
<V41S1139E1>        Column:  Should Be      OFF,
<V76X4133E1>        Column:  Should Be      OFF,
<V76X4134E1>        Column:  Should Be      ON,
<V76X4010E1>        Column:  Should Be      OFF,
<V76X4011E1>        Column:  Should Be      OFF,
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

Delay 2 sec;

Read Autocheckout Table functions and save as Column Actual;

Verify Autocheckout Table Actual equal to Should Be ELSE
  Begin Sequence;
        Record exceptions ‘ <’Measurement’> ‘is’ Actual ‘should be’ Should Be...
                ‘ with <V41K1139XL> set to OFF;
                        <V41K1140XL> set to OFF;
                        <V41K1141XL> set to OFF;
                        <V41K1142XL> set to ON’ to <PAGE-A>;
        Set AutoCheckoutError = ON;
  End Sequence;

// OMRS V41AE0.380 LINE R

Set <V41K1139XL> to OFF;
Set <V41K1140XL> to ON;
Set <V41K1141XL> to OFF;
Set <V41K1142XL> to ON;

ASSIGN Autocheckout Table:
<V41X1133E1> Column: Should Be OFF,
<V41X1145E1> Column: Should Be OFF,
<V41S1136E> Column: Should Be OFF,
<V76X4130E1> Column: Should Be OFF,
<V76X4131E1> Column: Should Be OFF,
<V76X4040E1> Column: Should Be OFF,
<V76X4041E1> Column: Should Be OFF,
<V41X1132E1> Column: Should Be ON,
<V41X1144E1> Column: Should Be OFF,
<V41S1139E1> Column: Should Be OFF,
<V76X4133E1> Column: Should Be OFF,
<V76X4134E1> Column: Should Be ON,
<V76X4010E1> Column: Should Be OFF,
<V76X4011E1> Column: Should Be OFF,

Delay 2 sec;

Read Autocheckout Table functions and save as Column Actual;

Verify Autocheckout Table Actual equal to Should Be ELSE
  Begin Sequence;
        Record exceptions ‘ <’Measurement’> ‘is’ Actual ‘should be’ Should Be...
                ‘ with <V41K1139XL> set to OFF;
                        <V41K1140XL> set to ON;
                        <V41K1141XL> set to OFF;
                        <V41K1142XL> set to ON’ to <PAGE-A>;
        Set AutoCheckoutError = ON;
  End Sequence;

// OMRS V41AE0.380 LINE S

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
Set <V41K1139XL>  to ON;
Set <V41K1140XL>  to OFF;
Set <V41K1141XL>  to OFF;
Set <V41K1142XL>  to ON;

ASSIGN Autocheckout Table:
<V41X1133E1>          Column:  Should Be      OFF,
<V41X1145E1>          Column:  Should Be      OFF,
<V41S1136E>           Column:  Should Be      OFF,
<V76X4130E1>          Column:  Should Be      OFF,
<V76X4131E1>          Column:  Should Be      OFF,
<V76X4040E1>          Column:  Should Be      OFF,
<V76X4041E1>          Column:  Should Be      OFF,
<V41X1132E1>          Column:  Should Be      OFF,
<V41X1144E1>          Column:  Should Be      ON,
<V41S1139E1>          Column:  Should Be      OFF,
<V76X4133E1>          Column:  Should Be      ON,
<V76X4134E1>          Column:  Should Be      ON,
<V76X4010E1>          Column:  Should Be      OFF,
<V76X4011E1>          Column:  Should Be      ON,

Delay 2 sec;

Read Autocheckout Table functions and save as Column Actual;

Verify Autocheckout Table Actual equal to Should Be ELSE
  Begin Sequence;
        Record exceptions ‘  <’Measurement’>  ‘is’ Actual  ‘should be’  Should Be...
                ‘ with <V41K1139XL>  set to ON;
                        <V41K1140XL>  set to OFF
                        <V41K1141XL>  set to OFF
                        <V41K1142XL>  set to ON’  to  <PAGE-A>;
        Set AutoCheckoutError = ON;
  End Sequence;



//  OMRS V41AE0.380  LINE T

Set <V41K1139XL>  to ON;
Set <V41K1140XL>  to OFF;
Set <V41K1141XL>  to ON;
Set <V41K1142XL>  to OFF;

ASSIGN Autocheckout Table:
<V41X1133E1>          Column:  Should Be      OFF,
<V41X1145E1>          Column:  Should Be      OFF,
<V41S1136E>           Column:  Should Be      OFF,
<V76X4130E1>          Column:  Should Be      OFF,
<V76X4131E1>          Column:  Should Be      OFF,
<V76X4040E1>          Column:  Should Be      OFF,
<V76X4041E1>          Column:  Should Be      OFF,
<V41X1132E1>          Column:  Should Be      ON,
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

<V41X1144E1> Column: Should Be OFF,
<V41S1139E1> Column: Should Be OFF,
<V76X4133E1> Column: Should Be ON,
<V76X4134E1> Column: Should Be OFF,
<V76X4010E1> Column: Should Be ON,
<V76X4011E1> Column: Should Be ON,

Delay 2 sec;

Read Autocheckout Table functions and save as Column Actual;

Verify Autocheckout Table Actual equal to Should Be ELSE
Begin Sequence;
Record exceptions ‘ <’Measurement’> ‘is’ Actual ‘should be’ Should Be...
‘ with <V41K1139XL> set to ON;
<V41K1140XL> set to OFF;
<V41K1141XL> set to ON;
<V41K1142XL> set to OFF’ to <PAGE-A>;
Set AutoCheckoutError = ON;
End Sequence;

// OMRS V41AB0.240 CLEANUP

Set <V41K1139XL> to OFF;
Set <V41K1140XL> to OFF;
Set <V41K1141XL> to OFF;
Set <V41K1142XL> to OFF;

ASSIGN Autocheckout Table:
<V41X1133E1> Column: Should Be OFF,
<V41X1145E1> Column: Should Be OFF,
<V41S1136E> Column: Should Be OFF,
<V76X4130E1> Column: Should Be OFF,
<V76X4131E1> Column: Should Be OFF,
<V76X4040E1> Column: Should Be OFF,
<V76X4041E1> Column: Should Be OFF,
<V41X1132E1> Column: Should Be OFF,
<V41X1144E1> Column: Should Be OFF,
<V41S1139E1> Column: Should Be OFF,
<V76X4133E1> Column: Should Be OFF,
<V76X4134E1> Column: Should Be OFF,
<V76X4010E1> Column: Should Be OFF,
<V76X4011E1> Column: Should Be OFF,

Delay 2 sec;

Read Autocheckout Table functions and save as Column Actual;

Verify Autocheckout Table Actual equal to Should Be ELSE
Begin Sequence;
Record exceptions ‘ <’Measurement’> ‘is’ Actual ‘should be’ Should Be...
‘ with <V41K1139XL> set to OFF;
<V41K1140XL> set to OFF;

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
                        <V41K1141XL>  set to OFF;
                        <V41K1142XL>  set to OFF'  to  <PAGE-A>;
        Set AutoCheckoutError = ON;
  End Sequence;



// OMRS V41AG0.070  -  PV1  CL INHIBIT MDM CMD REDUNDANCY

// OMRS V41AG0.070  LINE A

Set <V41K1139XL>  to ON;
Set <V41K1140XL>  to ON;
Set <V41K1141XL>  to ON;
Set <V41K1142XL>  to ON;
Set <V41K1125XL>  to ON;
Set <V41K1126XL>  to OFF;
Set <V41K1127XL>  to OFF;

ASSIGN Autocheckout Table:
<V41X1133E1>          Column:  Should Be      OFF,
<V41X1145E1>          Column:  Should Be      OFF,
<V41S1136E>           Column:  Should Be      OFF,
<V41X1132E1>          Column:  Should Be      ON,
<V41X1144E1>          Column:  Should Be      ON,
<V41S1139E1>          Column:  Should Be      OFF,

Delay 2 sec;

Read Autocheckout Table functions and save as Column Actual;

Verify Autocheckout Table Actual equal to Should Be ELSE
  Begin Sequence;
        Record exceptions '  <'Measurement'>  'is' Actual  'should be'  Should Be...
                ' with <V41K1139XL>  set to ON;
                        <V41K1140XL>  set to ON;
                        <V41K1141XL>  set to ON;
                        <V41K1142XL>  set to ON;
                        <V41K1125XL>  set to ON;
                        <V41K1126XL>  set to OFF;
                        <V41K1127XL>  set to OFF;'  to  <PAGE-A>;
        Set AutoCheckoutError = ON;
  End Sequence;

// OMRS V41AG0.070  LINE B

Set <V41K1139XL>  to ON;
Set <V41K1140XL>  to ON;
Set <V41K1141XL>  to ON;
Set <V41K1142XL>  to ON;
Set <V41K1125XL>  to ON;
Set <V41K1126XL>  to ON;
Set <V41K1127XL>  to OFF;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
ASSIGN Autocheckout Table:
<V41X1133E1>        Column:  Should Be      OFF,
<V41X1145E1>        Column:  Should Be      OFF,
<V41S1136E>         Column:  Should Be      OFF,
<V41X1132E1>        Column:  Should Be      OFF,
<V41X1144E1>        Column:  Should Be      ON,
<V41S1139E1>        Column:  Should Be      OFF,

Delay 2 sec;

Read Autocheckout Table functions and save as Column Actual;

Verify Autocheckout Table Actual equal to Should Be ELSE
  Begin Sequence;
        Record exceptions ‘ <’Measurement’>  ‘is’ Actual  ‘should be’  Should Be...
                ‘ with <V41K1139XL>  set to ON;
                        <V41K1140XL>  set to ON;
                        <V41K1141XL>  set to ON;
                        <V41K1142XL>  set to ON;
                        <V41K1125XL>  set to ON;
                        <V41K1126XL>  set to ON;
                        <V41K1127XL>  set to OFF;’  to  <PAGE-A>;
        Set AutoCheckoutError = ON;
  End Sequence;


//  OMRS V41AG0.070  LINE C

Set <V41K1139XL>  to ON;
Set <V41K1140XL>  to ON;
Set <V41K1141XL>  to ON;
Set <V41K1142XL>  to ON;
Set <V41K1125XL>  to ON;
Set <V41K1126XL>  to ON;
Set <V41K1127XL>  to ON;

ASSIGN Autocheckout Table:
<V41X1133E1>        Column:  Should Be      OFF,
<V41X1145E1>        Column:  Should Be      OFF,
<V41S1136E>         Column:  Should Be      OFF,
<V41X1132E1>        Column:  Should Be      OFF,
<V41X1144E1>        Column:  Should Be      OFF,
<V41S1139E1>        Column:  Should Be      OFF,

Delay 2 sec;

Read Autocheckout Table functions and save as Column Actual;

Verify Autocheckout Table Actual equal to Should Be ELSE
  Begin Sequence;
        Record exceptions ‘ <’Measurement’>  ‘is’ Actual  ‘should be’  Should Be...
                ‘ with <V41K1139XL>  set to ON;
                        <V41K1140XL>  set to ON;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
                        <V41K1141XL>  set to ON;
                        <V41K1142XL>  set to ON;
                        <V41K1125XL>  set to ON;
                        <V41K1126XL>  set to ON;
                        <V41K1127XL>  set to ON;'  to  <PAGE-A>;
        Set AutoCheckoutError = ON;
  End Sequence;


// OMRS V41AG0.070  LINE D

Set <V41K1139XL>  to ON;
Set <V41K1140XL>  to ON;
Set <V41K1141XL>  to ON;
Set <V41K1142XL>  to ON;
Set <V41K1125XL>  to OFF;
Set <V41K1126XL>  to ON;
Set <V41K1127XL>  to ON;

ASSIGN Autocheckout Table:
<V41X1133E1>          Column:  Should Be      OFF,
<V41X1145E1>          Column:  Should Be      OFF,
<V41S1136E>           Column:  Should Be      OFF,
<V41X1132E1>          Column:  Should Be      ON,
<V41X1144E1>          Column:  Should Be      ON,
<V41S1139E1>          Column:  Should Be      OFF,

Delay 2 sec;

Read Autocheckout Table functions and save as Column Actual;

Verify Autocheckout Table Actual equal to Should Be ELSE
  Begin Sequence;
        Record exceptions '  <'Measurement'>  'is' Actual  'should be'  Should Be...
                ' with <V41K1139XL>  set to ON;
                        <V41K1140XL>  set to ON;
                        <V41K1141XL>  set to ON;
                        <V41K1142XL>  set to ON;
                        <V41K1125XL>  set to OFF;
                        <V41K1126XL>  set to ON;
                        <V41K1127XL>  set to ON;'  to  <PAGE-A>;
        Set AutoCheckoutError = ON;
  End Sequence;


// OMRS V41AG0.070  CLEANUP

Set <V41K1139XL>  to OFF;
Set <V41K1140XL>  to OFF;
Set <V41K1141XL>  to OFF;
Set <V41K1142XL>  to OFF;
Set <V41K1125XL>  to OFF;
Set <V41K1126XL>  to OFF;
Set <V41K1127XL>  to OFF;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

ASSIGN Autocheckout Table:
<V41X1133E1> Column: Should Be OFF,
<V41X1145E1> Column: Should Be OFF,
<V41S1136E> Column: Should Be OFF,
<V41X1132E1> Column: Should Be OFF,
<V41X1144E1> Column: Should Be OFF,
<V41S1139E1> Column: Should Be OFF,

Delay 2 sec;

Read Autocheckout Table functions and save as Column Actual;

Verify Autocheckout Table Actual equal to Should Be ELSE
Begin Sequence;
Record exceptions ' <'Measurement'> 'is' Actual 'should be' Should Be...
' with <V41K1139XL> set to OFF;
<V41K1140XL> set to OFF;
<V41K1141XL> set to OFF;
<V41K1142XL> set to OFF;
<V41K1125XL> set to OFF;
<V41K1126XL> set to OFF;
<V41K1127XL> set to OFF;' to <PAGE-A>;
Set AutoCheckoutError = ON;
End Sequence;

3.6.2.3.1.6. Post-Requisites

3.6.2.3.1.7. State Diagram

<insert state diagram here>

3.6.2.3.1.8. State Table

3.6.2.3.1.9. Descriptive Text

3.6.2.4. User Interfaces

3.6.2.5. Safety Requirements

3.6.3. LH2 MDM Command Redundancy

3.6.3.1. Description

3.6.3.1.1. Instance Index

Instance	Other Users

3.6.3.1.2. Class Diagram

<insert class diagram here>

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.6.3.1.3. Attributes and Aggregations

Attribute	Data Type	Initial Value	Publish / Persistence	Client Access	Description

3.6.3.1.4. Operations Index

Operation	Description

3.6.3.1.5. Associations

Association	Type	Class	Description

3.6.3.2. State Description

3.6.3.2.1. State Diagram

<insert state diagram here>

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.6.3.2.2. State Table

3.6.3.3. Operations

3.6.3.3.1. Name of Operation

3.6.3.3.1.1. SET COMMANDS / VERIFY INDICATORS

3.6.3.3.1.1.1. AutoCheckout

3.6.3.3.1.1.1.1. Table: SET/VERIFY

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

V41K1223XL	V76X4120E1
V41K1224XL	V41X1302E1
V41K1225XL	V76X4125E1
V41K1322XL	V76X4126E1
V41K1323XL	V41X1406E1
V41K1324XL	V41X1458E1
V41K1325XL	
V41K1401XL	
V41K1402XL	
V41K1411XL	
V41K1465NL	
TEST	
SET COMMANDS ON	VERIFY INDICATOR = ON
V41K1122XL	V76X4113E1
V41K1123XL	V76X4114E1
V41K1124XL	V76X4119E1
V41K1125XL	V76X4120E1
V41K1126XL	V76X4125E1
V41K1222XL	V76X4126E1
V41K1223XL	V41X1406E1
V41K1224XL	V41X1458E1
V41K1225XL	
V41K1226XL	
V41K1322XL	
V41K1323XL	
V41K1324XL	
V41K1325XL	
V41K1326XL	
V41K1401XL	
V41K1402XL	
V41K1411XL	
V41K1466NL	
TEST	
SET COMMANDS ON	VERIFY INDICATOR = ON
V41K1122XL	V41X1102E1
V41K1123XL	V76X4113E1
V41K1124XL	V76X4114E1
V41K1126XL	V41X1202E1
V41K1222XL	V76X4119E1
V41K1223XL	V76X4120E1
V41K1224XL	V41X1302E1
V41K1226XL	V76X4125E1
V41K1322XL	V76X4126E1
V41K1323XL	V41X1406E1
V41K1324XL	
V41K1326XL	

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.6.3.3.1.2. Parameters

3.6.3.3.1.3. Effectivity

3.6.3.3.1.4. Prerequisites

3.6.3.3.1.5. Sequence

3.6.3.3.1.6. Post-Requisites

3.6.3.3.1.7. State Diagram

<insert state diagram here>

3.6.3.3.1.8. State Table

3.6.3.3.1.9. Descriptive Text

3.6.3.4. User Interfaces

3.6.3.5. Safety Requirements

3.6.4. MPS 750 Pneumatic Decay

3.6.4.1. Description

MPS 750 PNEUMATIC DECAY CLASS IS COMPOSED OF SIX INSTANCES. THIS CLASS MUST BE ABLE TO ACCESS ALL MPS PRESSURE MEASURMENTS AS WELL AS INDIVIDUAL FDS FOR THE LO2,LH2, AND GHE SYSTEMS. FURTHER DISPO TO FOLLOW.

3.6.4.1.1. Instance Index

Instance	Other Users
TEST_A	
TEST_B	
TEST_C	
TEST_D	
TEST_E	
TEST_ALL	

3.6.4.1.2. Class Diagram

<insert class diagram here>

3.6.4.1.3. Attributes and Aggregations

Attribute	Data Type	Initial Value	Publish / Persistence	Client Access	Description
REGOUT_LEG_VOLUME					GROUND LEG

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

					SOLENOID/VALVE VOLUME
ACCU_LEG_VOLUME					FLIGHT LEG SOLENOID/VALVE VOLUME
DELAY_TIME		15 MIN			THERMAL STABILIZATION DELAY AFTER VALVE SETUP. DEFAULT 15 MINUTES
DELAY_REMAINING					COUNTDOWN OF THERMAL STABILIZATION DELAY
DECAY_TIME		30 MIN			DURATION OF PRESSURE DECAY ANALYSIS. DEFAULT 30 MINUTES
DECAY_REMAINING					COUNTDOWN OF PRESSURE DECAY ANALYSIS
DWELL_TIME		20 SEC			PERIOD OF TIME IN WHICH PRESSURE HAS BEEN STABLE BETWEEN TOGGLE POINTS. DEFAULT 20 SECONDS
TEST_ID					IDENTIFIES CURRENT TEST IN PROGRESS
REGOUT_TOGGLE_POINTS					AMOUNT OF TOGGLE POINTS DETECTED DURING REGOUT DATA ANALYSIS
REGOUT_PR_START					PRESSURE RECORDED AT REGOUT DATA ANALYSIS START
REGOUT_GMT_START	GMT				GMT RECORDED AT REGOUT DATA ANALYSIS START
REGOUT_PR_INIT					PRESSURE RECORDED AT FIRST REGOUT TOGGLE POINT DETECTION
REGOUT_GMT_INIT	GMT				GMT RECORDED AT FIRST REGOUT TOGGLE POINT DETECTION
REGOUT_PR_PREV					
REGOUT_GMT_PREV	GMT				

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

REGOUT_PR_LAST					PRESSURE RECORDED AT LAST REGOUT TOGGLE POINT DETECTION
REGOUT_GMT_LAST	GMT				GMT RECORDED AT LAST REGOUT TOGGLE POINT DETECTION
REGOUT_PR_END					PRESSURE RECORDED AT REGOUT DATA ANALYSIS END
REGOUT_GMT_END	GMT				GMT RECORDED AT REGOUT DATA ANALYSIS END
REGOUT_DELTA_P					PRESSURE DIFFERENCE USED IN REGOUT DECAY RATE CALCULATIONS
REGOUT_DELTA_T					GMT DIFFERENCE USED IN REGOUT DECAY RATE CALCULATIONS
REGOUT_DECAY_RATE					CALCULATION OF REGOUT DP/DT
REGOUT_ALLOWABLE					REGOUT DECAY RATE ALLOWABLE PER OMRS
REGOUT_MARGIN					DIFFERNCE BETWEEN REGOUT DECAY RATE AND REGOUT ALLOWABLE
REGOUT_SCIM					REGOUT DECAY RATE IN STANDARD CUBIC INCHES PER MINUTE (VOLUMETRIC FLOW RATE)
REGOUT_RESULT_STATUS					PASS / FAIL
ACCUM_TOGGLE_POINTS					AMOUNT OF TOGGLE POINTS DETECTED DURING ACCUM DATA ANALYSIS
ACCUM_PR_START					PRESSURE RECORDED AT ACCUM DATA ANALYSIS START
ACCUM_GMT_START	GMT				GMT RECORDED AT ACCUM DATA ANALYSIS START

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

ACCUM_PR_INIT					PRESSURE RECORDED AT FIRST ACCUM TOGGLE POINT DETECTION
ACCUM_GMT_INIT	GMT				GMT RECORDED AT FIRST ACCUM TOGGLE POINT DETECTION
ACCUM_PR_PREV					
ACCUM_GMT_PREV	GMT				
ACCUM_PR_LAST					PRESSURE RECORDED AT LAST ACCUM TOGGLE POINT DETECTION
ACCUM_GMT_LAST	GMT				GMT RECORDED AT LAST ACCUM TOGGLE POINT DETECTION
ACCUM_PR_END					PRESSURE RECORDED AT ACCUM DATA ANALYSIS END
ACCUM_GMT_END	GMT				GMT RECORDED AT ACCUM DATA ANALYSIS END
ACCUM_DELTA_P					PRESSURE DIFFERENCE USED IN ACCUM DECAY RATE CALCULATIONS
ACCUM_DELTA_T					GMT DIFFERENCE USED IN ACCUM DECAY RATE CALCULATIONS
ACCUM_DECAY_RATE					CALCULATION OF ACCUM DP/DT
ACCUM_ALLOWABLE					ACCUM DECAY RATE ALLOWABLE PER OMRS
ACCUM_MARGIN					DIFFERNCE BETWEEN ACCUM DECAY RATE AND ACCUM ALLOWABLE
ACCUM_SCIM					ACCUM DECAY RATE IN STANDARD CUBIC INCHES PER MINUTE (VOLUMETRIC FLOW RATE)
ACCUM_RESULT_STATUS					PASS / FAIL

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.6.4.1.4. Operations Index

Operation	**Description**
ORBITER_STANDBY_CONFIG()	FUNCTION CONFIGURES ORBITER VALVES AND SOLENOIDS TO OPF STANDBY-CONFIGURATION.
DATA_ANALYSIS()	ANALYSES MPS HELIUM REGOUT AND ACCUMULATOR PRESSURE DECAY. TOGGLE POINT LOGIC IS DETERMINED HERE.
START_DECAY()	INITIALIZES PERTINENT VARIABLES AND CALLS DATA_ANALYSIS().
CHECK_STANDBY()	VERIFIES ORBITER LH2,LO2,GHE VALVES AND SOLENOIDS ARE IN STAND-BY CONFIGURATION. ALSO, ME-2 PNEUMATIC PRESSURE,GSE PANEL POWER AND VALVE CONFIGURATION.
MONITOR_LOOP()	ENVOKES A FINATE STATE MACHINE (FSM) THAT LOOPS IN TIME FOR A SPECIFIED PERIOD OF TIME.
SET_PRESSURE_LIMITS()	SETS CONTRAINT LIMITS AT BEGINNING OF DECAY AND AT ANYTIME REGOUT OR ACCUMULATOR PRESSURE CHANGES.
INITIAL_SETUP()	INITIALIZES VARIABLES FOR TESTS A-E AND T/S.
VALVE_CONFIGURATION_A()	CONFIGURES VALVES/SOLENOIDS PER OMRS V41AZ0.100-A.
VALVE_CONFIGURATION_B()	CONFIGURES VALVES/SOLENOIDS PER OMRS V41AZ0.100-B.
VALVE_CONFIGURATION_C()	CONFIGURES VALVES/SOLENOIDS PER OMRS V41AZ0.100-C.
VALVE_CONFIGURATION_D()	CONFIGURES VALVES/SOLENOIDS PER OMRS V41AZ0.100-D.
VALVE_CONFIGURATION_E()	CONFIGURES VALVES/SOLENOIDS PER OMRS V41AZ0.100-E.
VALVE_CONFIGURATION_ALL()	CALLS VALVE_CONFIGURATION_A() THROUGH E()
TROUBLE_SHOOTING()	CONFIGURES VALVES/SOLENOIDS FOR TROUBLESHOOTING.
DECAY_RATE_RESULTS()	RECORDS REGOUT AND ACCUMULATOR DATA ANALYSIS TO PAGE-B AND PRINTERS.
LEG_VOLUME()	CALCULATES SOLENOID/VALVE PNEUMATIC LINE VOLUMES ASSOCIATED WITH REGOUT(GROUND LEG) AND

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

	ACCUM(FLIGH LEG).
CV9_DELTA_PRESSURE()	COMPUTES PRESSURE DIFFERENCE BETWEEN ACCUMULATOR AND REGOUT.
HIGH_MANIFOLD_PRESS()	DETERMINES LH2/ LO2 MANIFOLD PRESSURE INCREASE AFTER SELECTED SOLENOID CYCLES.

3.6.4.1.5. Associations

Association	Type	Class	Description

3.6.4.2. State Description

3.6.4.2.1. State Diagram

<insert state diagram here>

3.6.4.2.2. State Table

3.6.4.3. Operations

3.6.4.3.1. ORBITER_STANDBY_MODE()

3.6.4.3.1.1. Parameters

3.6.4.3.1.2. Effectivity

3.6.4.3.1.3. Prerequisites

3.6.4.3.1.4. Sequence

```
   RECORD TEXT (ORBITER STAND-BY RECONFIGURATION IN PROGRESS)TO <PAGE-B//

            (RECONFIG ERROR) = FALSE;


// CLOSE LV60

             RECORD TEXT (CHECKING LV60...)
                    TO <PAGE-B>

             VERIFY <V41X1170E1> IS ON THEN

             MPS_LV60.CLOSE();

//CLOSE LV62

             RECORD TEXT (CHECKING LV62...)
                    TO <PAGE-B>
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
             VERIFY <V41X1270E1> IS ON THEN

              MPS_LV62.CLOSE();

//CLOSE LV64
             RECORD TEXT (CHECKING LV64...)
                    TO <PAGE-B>

             VERIFY <V41X1370E1> IS ON THEN

            MPS_LV64.CLOSE();

//CLOSE LV40

             RECORD TEXT (CHECKING LV40...)
                    TO <PAGE-B>

             VERIFY <V41X1538E1> IS ON THEN

            MPS_LV40.CLOSE();

//CLOSE LV41

             RECORD TEXT (CHECKING LV41...)
                    TO <PAGE-B>

             VERIFY <V41X1539E1> IS ON THEN

            MPS_LV41.CLOSE();


//CLOSE LV42

             RECORD TEXT (CHECKING LV42...)
                    TO <PAGE-B>

             VERIFY <V41X1436E1> IS ON THEN

            MPS_LV42.CLOSE();

//CLOSE LV43

             RECORD TEXT (CHECKING LV43...)
                    TO <PAGE-B>

             VERIFY <V41X1438E1> IS ON THEN

            MPS_LV43.CLOSE();


//CLOSE LV74

             RECORD TEXT (CHECKING LV74...)
                    TO <PAGE-B>
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
             VERIFY <V41X1901E1> IS ON THEN

MPS_LV74.CLOSE();

CLOSE LV75

             RECORD TEXT (CHECKING LV75...)
                    TO <PAGE-B>

             VERIFY <V41X1902E1> IS ON THEN

MPS_LV75.CLOSE();


//CLOSE LV26


             RECORD TEXT (CHECKING LV26...)
                    TO <PAGE-B>

             VERIFY <V41X1632E1> IS ON THEN

            MPS_LV26.CLOSE();

//CLOSE LV27

             RECORD TEXT (CHECKING LV27...)
                    TO <PAGE-B>

             VERIFY <V41X1634E1> IS ON THEN

MPS_LV27.CLOSE();


//************** VERIFY  OPF OR MLP GSE IS SECURED **************

                  VERIFY <N03IS078E> IS ON THEN

                  BEGIN SEQUENCE;

   RECORD TEXT(CHECKING IF OPF GSE IS SECURED                    ) TO
                  <PAGE-B>

                  DELAY 3 SEC;

                  // OPF CHECK A83129,A83130,A83132,A83133 //

                  VERIFY <GHEX2015E>
                         <GHEX2025E>
                         <GHEX2055E>
                         <GHEX2065E> ARE OFF

                   ELSE BEGIN SEQUENCE;

                   (VLVERROR) = ON;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
 RECORD TEXT(OPF GSE NOT SECURED.CHECK A83133,A83129,A83132,A83130) TO
   <PAGE-B> RED;

                        END SEQUENCE;


                    END SEQUENCE;

                ELSE BEGIN SEQUENCE;


                  // MLP CHECK A74983 //

                  RECORD TEXT(CHECKING IF MLP GSE IS SECURED) TO
                  <PAGE-B> LINE 8
                  DELAY 3 SEC;

                  VERIFY  <GHEX1093E>
                          <GHEX1193E> ARE ON

                  ELSE BEGIN SEQUENCE;
                  ASSIGN (VLVERROR) = ON;

  RECORD TEXT(MLP GSE NOT SECURED, CHECK A74983) TO
                <PAGE-B> RED;

                      END SEQUENCE;

                END SEQUENCE;



//VALVE ERROR MESSAGE

         IF (VLVERROR) = ON THEN
    BEGIN SEQUENCE
          RECORD TEXT (EVALUATE BEFORE CONTINUING)
          TO  <PAGE-B> RED;
          ASSIGN (VLVERROR) = OFF;
          END SEQUENCE;


//MPS PNEUMATICS ACTIVATION

//OPEN LV7

             RECORD TEXT (CHECKING LV7... )
                    TO <PAGE-B>

             VERIFY <V41P1605A1> IS LESS THAN 650 PSIA THEN
                SET <NLHK0072X> TO ON;

             VERIFY <V41X1645E1> IS OFF THEN
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
            BEGIN SEQUENCE;
 MPS_LV7.OPEN();

            SET <NLHK0072X> TO OFF;

            END SEQUENCE;

//OPEN LV8

            RECORD TEXT (CHECKING LV8... )
                   TO <PAGE-B>

            VERIFY <V41P1605A1> IS LESS THAN 650 PSIA THEN
               SET <NLHK0073X> TO ON;

            VERIFY <V41X1646E1> IS OFF THEN

               MPS_LV8.OPEN();

            SET <NLHK0073X> TO OFF;

            DELAY 4 SEC;


//LV10



//LH2

//PD2

//DE-ENERGIZE PD2 OPENING SOLENOID, LV48

//UNLOCK PD2 LATCH

            RECORD TEXT (CHECKING PD2 LATCH...)
                   TO <PAGE-B>

            IF (PD2 LOCK) IS ON THEN


MPS_PD2_LATCH.UNLOCK();

//DE-ENERGIZE PD2 LATCHING SOLENOID, LV67

            RECORD TEXT (CHECKING LV67...)
                   TO <PAGE-B>
            VERIFY <V41X1383E1> IS ON THEN

           MPS_LV67.CLOSE();


//DE-ENERGIZE PD2 UNLATCHING SOLENOID, LV68
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
            RECORD TEXT (CHECKING LV68...)
                   TO <PAGE-B>

            VERIFY <V41X1384E1> IS ON THEN

      MPS_LV68.CLOSE();



RECORD TEXT (CHECKING PD2 ...) TO <PAGE-B>

      IF (PD2 OP)  IS OFF THEN
                LET (RECONFIGURE) = (RECONFIGURE) + 2;

            END SEQUENCE;

// ENERGIZE PD2 OPENING SOLENOID, LV48 //

 STATEMENT GROUP FOR (RECONFIGURE) = (20)
  BEGIN SEQUENCE;

            RECORD TEXT (CHECKING LV48...)
                   TO <PAGE-B> LINE 8 COLUMN 3 CYAN;

            VERIFY <V41X1382E1> IS OFF THEN
               BEGIN SEQUENCE;
               WHEN INTERRUPT <TIMER> EQUAL TO 1 SEC OCCURS GOTO STEP 1010;
               DELAY 2 SEC;
               END SEQUENCE;
            END SEQUENCE;

// ENERGIZE PD2 CLOSING SOLENOID, LV49 //

            RECORD TEXT (CHECKING LV49...)
                   TO <PAGE-B>

            VERIFY <V41X1381E1> IS OFF THEN

MPS_LV49.OPEN();

// DE-ENERGIZE PD2 OPENING SOLENOID, LV48 //


            RECORD TEXT (CHECKING LV48...)
                   TO <PAGE-B>
            VERIFY <V41X1382E1> IS ON THEN
              MPS_LV48.CLOSE();

// DE-ENERGIZE PD2 CLOSING SOLENOID, LV49 //

            RECORD TEXT (CHECKING LV49...)
                   TO <PAGE-B>

            VERIFY <V41X1381E1> IS ON THEN
              MPS_LV49.CLOSE();
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
            RECORD TEXT (CHECKING PD3 ...)
                   TO <PAGE-B>

            IF (PD3 OP) = OFF THEN
               LET (RECONFIGURE) = (RECONFIGURE) + 2;

            END SEQUENCE;

// ENERGIZE PD3 OPENING SOLENOID, LV50 //

          STATEMENT GROUP FOR (RECONFIGURE) = (25)
            BEGIN SEQUENCE;

            RECORD TEXT (CHECKING LV50...)
                   TO <PAGE-B>

            VERIFY <V41X1440E1> = OFF THEN
              MPS_LV50.OPEN();

// ENERGIZE PD3 CLOSING SOLENOID, LV51 //


            RECORD TEXT (CHECKING LV51...)
                   TO <PAGE-B>

            VERIFY <V41X1439E1> = OFF THEN
              MPS_LV51.OPEN();

// DE-ENERGIZE PD3 OPENING SOLENOID, LV50 //


            RECORD TEXT (CHECKING LV50...)
                   TO <PAGE-B>

            VERIFY <V41X1440E1> IS ON THEN
              MPS_LV50.CLOSE();

// DE-ENERGIZE PD3 CLOSING SOLENOID, LV51 //

            RECORD TEXT (CHECKING LV51...)
                   TO <PAGE-B>

            VERIFY <V41X1439E1> IS ON THEN
              MPS_LV51.CLOSE()

// CLOSE PV4 //

            RECORD TEXT (CHECKING PV4 ...)
                   TO <PAGE-B>
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
             IF (PV4 OP) = OFF THEN
              MPS_PV4.OPEN();

// DE-ENERGIZE PV4 CLOSING SOLENOID, LV19 //


             RECORD TEXT (CHECKING LV19...)
                    TO <PAGE-B>

          VERIFY <V41X1102E1> = ON THEN

MPS_LV19.CLOSE();

// DE-ENERGIZE PV4 OPENING SOLENOID, LV18 //

               RECORD TEXT (CHECKING LV18...)
                      TO <PAGE-B>

             VERIFY <V41X1103E1> = ON THEN
               MPS_LV18.CLOSE();

// CLOSE PV5 //

             RECORD TEXT (CHECKING PV5 ...)
                    TO <PAGE-B>

             IF (PV5 OP) = OFF THEN
                MPS_PV5.CLOSE();

// DE-ENERGIZE PV5 CLOSING SOLENOID, LV21 //

             RECORD TEXT (CHECKING LV21...)
                    TO <PAGE-B>

             VERIFY <V41X1202E1> = ON THEN
               MPS_LV21.CLOSE();

// DE-ENERGIZE PV5 OPENING SOLENOID, LV20 //

             RECORD TEXT (CHECKING LV20...)
                    TO <PAGE-B>

             VERIFY <V41X1203E1> = ON THEN

               MPS_LV20.CLOSE();

// CLOSE PV6 //


             RECORD TEXT (CHECKING PV6 ...)
                    TO <PAGE-B>

             IF (PV6 OP) = OFF THEN

       MPS_PV6.CLOSE();
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
// DE-ENERGIZE PV6 CLOSING SOLENOID, LV23 //

             RECORD TEXT (CHECKING LV23...)
                    TO <PAGE-B>

             VERIFY <V41X1302E1> = ON THEN

      MPS_LV23_CLOSE();

// DE-ENERGIZE PV6 OPENING SOLENOID, LV22 //

             RECORD TEXT (CHECKING LV22...)
                    TO <PAGE-B>

             VERIFY <V41X1303E1> = ON THEN

      MPS_LV22_CLOSE();

// OPEN PV8 BY DE-ENERGIZING LV25 //

             RECORD TEXT (CHECKING PV8/LV25...)
                    TO <PAGE-B>

             IF (PV8 OP) = OFF THEN
               MPS_LV25.CLOSE();


// OPEN PV11 //


             RECORD TEXT (CHECKING PV11...)
                    TO <PAGE-B> L

             IF (PV11 OP) = OFF THEN

      MPS_PV11_OPEN();


// DE-ENERGIZE PV11 CLOSING SOLENOID, LV33 //
             RECORD TEXT (CHECKING LV33...)
                    TO <PAGE-B>

             VERIFY <V41X1385E1> = ON THEN
             MPS_LV33.CLOSE();

// DE-ENERGIZE PV11 OPENING SOLENOID, LV32 //

             RECORD TEXT (CHECKING LV32...)
                     TO <PAGE-B>

             VERIFY <V41X1386E1> = ON THEN
               MPS_LV32.CLOSE();

// OPEN PV12 //
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
             RECORD TEXT (CHECKING PV12...)
                    TO <PAGE-B>
             IF (PV12 OP) = OFF THEN

               MPS_PV12.OPEN();

// DE-ENERGIZE PV12 CLOSING SOLENOID, LV35 //

             RECORD TEXT (CHECKING LV35...)
                    TO <PAGE-B>

             VERIFY <V41X1405E1> = ON THEN

      MPS_LV35.CLOSE();

// DE-ENERGIZE PV12 OPENING SOLENOID, LV34 //

             RECORD TEXT (CHECKING LV34...)
                    TO <PAGE-B>

             VERIFY <V41X1406E1> = ON THEN
              MPS_LV34.CLOSE();

// CLOSE PV13 BY DE-ENERGIZING OPENING SOLENOID, LV39 //


             RECORD TEXT (CHECKING LV39...)
                    TO <PAGE-B>

             IF (PV13 OP) = ON THEN

            MPS_LV39.CLOSE();

// CLOSE PV14, PV15, & PV16 BY DE-ENERGIZING OPENING SOLENOID, LV36 //

             RECORD TEXT (CHECKING PV14,PV15,PV16/LV36...)
                    TO <PAGE-B>

             IF (PV14 OP) = ON THEN

      MPS_LV36.CLOSE();

// CLOSE PV17 BY DE-ENERGIZING OPENING SOLENOID, LV72 //


            RECORD TEXT (CHECKING PV17/LV72...)
                   TO <PAGE-B>

            IF (PV17 OP) = ON THEN

              MPS_LV72.CLOSE();

// CLOSE PV18 BY DE-ENERGIZING OPENING SOLENOID, LV73 //
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
            RECORD TEXT (CHECKING PV18/LV73...)
                   TO <PAGE-B>

            IF (PV18 OP) = ON THEN
              MPS_LV73.CLOSE();

// CLOSE PV22 BY DE-ENERGIZING OPENING SOLENOID, LV79 //



          RECORD TEXT (CHECKING PV22/LV79...)
                 TO <PAGE-B>

          IF (PV22 OP) = ON THEN

MPS_LV79.CLOSE();

// LH2 //

CONTINUE

// UNLOCK PD1 LATCH//

          STATEMENT GROUP FOR (RECONFIGURE) = (50)
             BEGIN SEQUENCE;

             RECORD TEXT (CHECKING PD1 LATCH...)
                    TO <PAGE-B> LINE 8 COLUMN 3 CYAN;

             IF (PD1 LOCK) IS ON THEN
                BEGIN SEQUENCE;
                WHEN INTERRUPT <TIMER> EQUAL TO 1 SEC OCCURS GOTO STEP 2055;
                DELAY 2 SEC;
                END SEQUENCE;
             END SEQUENCE;

// DE-ENERGIZE PD1 LATCHING SOLENOID, LV65 //

          STATEMENT GROUP FOR (RECONFIGURE) = (51)
             BEGIN SEQUENCE;

             RECORD TEXT (CHECKING LV65...)
                    TO <PAGE-B> LINE 8 COLUMN 3 CYAN;

             VERIFY <V41X1808E1> IS ON THEN
                BEGIN SEQUENCE;
                WHEN INTERRUPT <TIMER> EQUAL TO 1 SEC OCCURS GOTO STEP 2060;
                DELAY 2 SEC;
                END SEQUENCE;
             END SEQUENCE;

// DE-ENERGIZE PD1 UNLATCHING SOLENOID, LV66 //

          STATEMENT GROUP FOR (RECONFIGURE) = (52)
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
            BEGIN SEQUENCE;

            RECORD TEXT (CHECKING LV66...)
                   TO <PAGE-B> LINE 8 COLUMN 3 CYAN;

            VERIFY <V41X1809E1> IS ON THEN
               BEGIN SEQUENCE;
               WHEN INTERRUPT <TIMER> EQUAL TO 1 SEC OCCURS GOTO STEP 2070;
               DELAY 2 SEC;
               END SEQUENCE;
            END SEQUENCE;

// PD1. IF PD1 IS CLOSED THEN OPEN BOTH SOLENOIDS AND THEN CLOSE THEM.
HOWEVER
  IF PD1 IS CLOSED THEN SKIP THE TWO STEPS THAT ENERGIZE BOTH SOLENOIDS AND
  GO STRAIGHT TO CLOSING THE SOLENOIDS.//


         STATEMENT GROUP FOR (RECONFIGURE) = (53)
            BEGIN SEQUENCE;

            RECORD TEXT (CHECKING PD1 ...)
                   TO <PAGE-B> LINE 8 COLUMN 3 CYAN;

            IF (PD1 OP)  IS OFF THEN
               LET (RECONFIGURE) = (RECONFIGURE) + 2;

            END SEQUENCE;

// ENERGIZE PD1 OPENING SOLENOID, LV46 //

         STATEMENT GROUP FOR (RECONFIGURE) = (54)
            BEGIN SEQUENCE;

            RECORD TEXT (CHECKING LV46...)
                   TO <PAGE-B> LINE 8 COLUMN 3 CYAN;

            VERIFY <V41X1807E1> IS OFF THEN
               BEGIN SEQUENCE;
               WHEN INTERRUPT <TIMER> EQUAL TO 1 SEC OCCURS GOTO STEP 2010;
               DELAY 2 SEC;
               END SEQUENCE;
            END SEQUENCE;

// ENERGIZE PD1 CLOSING SOLENOID, LV47 //

         STATEMENT GROUP FOR (RECONFIGURE) = (55)
            BEGIN SEQUENCE;

            RECORD TEXT (CHECKING LV47...)
                   TO <PAGE-B> LINE 8 COLUMN 3 CYAN;

            VERIFY <V41X1806E1> IS OFF THEN
               BEGIN SEQUENCE;
               WHEN INTERRUPT <TIMER> EQUAL TO 1 SEC OCCURS GOTO STEP 2020;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
               DELAY 2 SEC;
               END SEQUENCE;
            END SEQUENCE;

// DE-ENERGIZE PD1 OPENING SOLENOID, LV46 //

         STATEMENT GROUP FOR (RECONFIGURE) = (56)
            BEGIN SEQUENCE;

            RECORD TEXT (CHECKING LV46...)
                   TO <PAGE-B> LINE 8 COLUMN 3 CYAN;

            VERIFY <V41X1807E1> IS ON THEN
               BEGIN SEQUENCE;
               WHEN INTERRUPT <TIMER> EQUAL TO 1 SEC OCCURS GOTO STEP 2010;
               DELAY 2 SEC;
               END SEQUENCE;
            END SEQUENCE;

// DE-ENERGIZE PD1 CLOSING SOLENOID, LV47 //

         STATEMENT GROUP FOR (RECONFIGURE) = (57)
            BEGIN SEQUENCE;

            RECORD TEXT (CHECKING LV47...)
                   TO <PAGE-B> LINE 8 COLUMN 3 CYAN;

            VERIFY <V41X1806E1> IS ON THEN
               BEGIN SEQUENCE;
               WHEN INTERRUPT <TIMER> EQUAL TO 1 SEC OCCURS GOTO STEP 2020;
               DELAY 2 SEC;
            END SEQUENCE;
         END SEQUENCE;

// CLOSE PV1 //

         STATEMENT GROUP FOR (RECONFIGURE) = (58)
            BEGIN SEQUENCE;

            RECORD TEXT (CHECKING PV1 ...)
                   TO <PAGE-B> LINE 8 COLUMN 3 CYAN;

            IF (PV1 OP) IS ON THEN
               BEGIN SEQUENCE;
               WHEN INTERRUPT <TIMER> EQUAL TO 1 SEC OCCURS GOTO STEP 2090;
               DELAY 2 SEC;
               END SEQUENCE;
            END SEQUENCE;

// DE-ENERGIZE PV1 OPENING SOLENOID, LV12 //

         STATEMENT GROUP FOR (RECONFIGURE) = (59)
            BEGIN SEQUENCE;

            RECORD TEXT (CHECKING LV12...)
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
                    TO <PAGE-B> LINE 8 COLUMN 3 CYAN;

              VERIFY <V41X1133E1> IS ON THEN
                 BEGIN SEQUENCE;
                 WHEN INTERRUPT <TIMER> EQUAL TO 1 SEC OCCURS GOTO STEP 2110;
                 DELAY 2 SEC;
                 END SEQUENCE;
              END SEQUENCE;

// DE-ENERGIZE PV1 OPENING SOLENOID, LV83 //

           STATEMENT GROUP FOR (RECONFIGURE) = (60)
              BEGIN SEQUENCE;

              RECORD TEXT (CHECKING LV83 ...)
                     TO <PAGE-B> LINE 8 COLUMN 3 CYAN;

              VERIFY <V41X1145E1> IS ON THEN
                 BEGIN SEQUENCE;
                 WHEN INTERRUPT <TIMER> EQUAL TO 1 SEC OCCURS GOTO STEP 2130;
                 DELAY 2 SEC;
                 END SEQUENCE;
              END SEQUENCE;

// DE-ENERGIZE PV1 CLOSING SOLENOID, LV13 //

           STATEMENT GROUP FOR (RECONFIGURE) = (61)
              BEGIN SEQUENCE;

              RECORD TEXT (CHECKING LV13...)
                     TO <PAGE-B> LINE 8 COLUMN 3 CYAN;

              VERIFY <V41X1132E1> IS ON THEN
                 BEGIN SEQUENCE;
                 WHEN INTERRUPT <TIMER> EQUAL TO 1 SEC OCCURS GOTO STEP 2120;
                 DELAY 2 SEC;
                 END SEQUENCE;
              END SEQUENCE;

// DE-ENERGIZE PV1 CLOSING SOLENOID, LV80 //

           STATEMENT GROUP FOR (RECONFIGURE) = (62)
              BEGIN SEQUENCE;

              RECORD TEXT (CHECKING LV80...)
                     TO <PAGE-B> LINE 8 COLUMN 3 CYAN;

              VERIFY <V41X1144E1> IS ON THEN
                 BEGIN SEQUENCE;
                 WHEN INTERRUPT <TIMER> EQUAL TO 1 SEC OCCURS GOTO STEP 2140;
                 DELAY 2 SEC;
                 END SEQUENCE;
              END SEQUENCE;

// CLOSE PV2 //
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
          STATEMENT GROUP FOR (RECONFIGURE) = (63)
             BEGIN SEQUENCE;

             RECORD TEXT (CHECKING PV2 ...)
                    TO <PAGE-B> LINE 8 COLUMN 3 CYAN;

             IF (PV2 OP) IS ON THEN
                BEGIN SEQUENCE;
                WHEN INTERRUPT <TIMER> EQUAL TO 1 SEC OCCURS GOTO STEP 2148;
                DELAY 2 SEC;
                END SEQUENCE;
             END SEQUENCE;

// DE-ENERGIZE PV2 OPENING SOLENOID, LV14 //

          STATEMENT GROUP FOR (RECONFIGURE) = (64)
             BEGIN SEQUENCE;

             RECORD TEXT (CHECKING LV14...)
                    TO <PAGE-B> LINE 8 COLUMN 3 CYAN;

             VERIFY <V41X1233E1> IS ON THEN
                BEGIN SEQUENCE;
                WHEN INTERRUPT <TIMER> EQUAL TO 1 SEC OCCURS
                GOTO STEP 2160;
                DELAY 2 SEC;
                END SEQUENCE;
             END SEQUENCE;

// DE-ENERGIZE PV2 OPENING SOLENOID, LV84 //

          STATEMENT GROUP FOR (RECONFIGURE) = (65)
             BEGIN SEQUENCE;

             RECORD TEXT (CHECKING LV84...)
                    TO <PAGE-B> LINE 8 COLUMN 3 CYAN;

             VERIFY <V41X1245E1> IS ON THEN
                BEGIN SEQUENCE;
                WHEN INTERRUPT <TIMER> EQUAL TO 1 SEC OCCURS
                GOTO STEP 2180;
                DELAY 2 SEC;
                END SEQUENCE;
             END SEQUENCE;

// DE-ENERGIZE PV2 CLOSING SOLENOID, LV15 //

          STATEMENT GROUP FOR (RECONFIGURE) = (66)
             BEGIN SEQUENCE;

             RECORD TEXT (CHECKING LV15...)
                    TO <PAGE-B> LINE 8 COLUMN 3 CYAN;

             VERIFY <V41X1232E1> IS ON THEN
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
                BEGIN SEQUENCE;
                WHEN INTERRUPT <TIMER> EQUAL TO 1 SEC OCCURS
                GOTO STEP 2170;
                DELAY 2 SEC;
                END SEQUENCE;
             END SEQUENCE;

// DE-ENERGIZE PV2 CLOSING SOLENOID, LV81 //

          STATEMENT GROUP FOR (RECONFIGURE) = (67)
             BEGIN SEQUENCE;

             RECORD TEXT (CHECKING LV81...)
                    TO <PAGE-B> LINE 8 COLUMN 3 CYAN;

             VERIFY <V41X1244E1> IS ON THEN
                BEGIN SEQUENCE;
                WHEN INTERRUPT <TIMER> EQUAL TO 1 SEC OCCURS
                GOTO STEP 2190;
                DELAY 2 SEC;
                END SEQUENCE;
             END SEQUENCE;

// CLOSE PV3 //

          STATEMENT GROUP FOR (RECONFIGURE) = (68)
             BEGIN SEQUENCE;

             RECORD TEXT (CHECKING PV3 ...)
                    TO <PAGE-B> LINE 8 COLUMN 3 CYAN;

             IF (PV3 OP) IS ON THEN
                BEGIN SEQUENCE;
                WHEN INTERRUPT <TIMER> EQUAL TO 1 SEC OCCURS
                GOTO STEP 2198;
                DELAY 2 SEC;
                END SEQUENCE;
             END SEQUENCE;

// DE-ENERGIZE PV3 OPENING SOLENOID, LV16 //

          STATEMENT GROUP FOR (RECONFIGURE) = (69)
             BEGIN SEQUENCE;

             RECORD TEXT (CHECKING LV16...)
                    TO <PAGE-B> LINE 8 COLUMN 3 CYAN;

             VERIFY <V41X1333E1> IS ON THEN
                BEGIN SEQUENCE;
                WHEN INTERRUPT <TIMER> EQUAL TO 1 SEC OCCURS
                GOTO STEP 2210;
                DELAY 2 SEC;
                END SEQUENCE;
             END SEQUENCE;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
// DE-ENERGIZE PV3 OPENING SOLENOID, LV85 //

      STATEMENT GROUP FOR (RECONFIGURE) = (70)
     BEGIN SEQUENCE;

          RECORD TEXT (CHECKING LV85...)
               TO <PAGE-B> LINE 8 COLUMN 3 CYAN;

      VERIFY <V41X1345E1> IS ON THEN
         BEGIN SEQUENCE;
               WHEN INTERRUPT <TIMER> EQUAL TO 1 SEC OCCURS
               GOTO STEP 2230;
               DELAY 2 SEC;
         END SEQUENCE;
     END SEQUENCE;

// DE-ENERGIZE PV3 CLOSING SOLENOID, LV17 //

          STATEMENT GROUP FOR (RECONFIGURE) = (71)
             BEGIN SEQUENCE;

             RECORD TEXT (CHECKING LV17...)
                    TO <PAGE-B> LINE 8 COLUMN 3 CYAN;

             VERIFY <V41X1332E1> IS ON THEN
                BEGIN SEQUENCE;
                WHEN INTERRUPT <TIMER> EQUAL TO 1 SEC OCCURS
                GOTO STEP 2220;
                DELAY 2 SEC;
                END SEQUENCE;
             END SEQUENCE;

// DE-ENERGIZE PV3 CLOSING SOLENOID, LV82 //

          STATEMENT GROUP FOR (RECONFIGURE) = (72)
             BEGIN SEQUENCE;

             RECORD TEXT (CHECKING LV82...)
                    TO <PAGE-B> LINE 8 COLUMN 3 CYAN;

             VERIFY <V41X1344E1> IS ON THEN
                BEGIN SEQUENCE;
                WHEN INTERRUPT <TIMER> EQUAL TO 1 SEC OCCURS
                GOTO STEP 2240;
                DELAY 2 SEC;
                END SEQUENCE;
             END SEQUENCE;

// OPEN PV7 BY DE-ENERGIZING CLOSING SOLENOID, LV24 //

          STATEMENT GROUP FOR (RECONFIGURE) = (73)
             BEGIN SEQUENCE;

             RECORD TEXT (CHECKING PV7/LV24...)
                    TO <PAGE-B> LINE 8 COLUMN 3 CYAN;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
            IF (PV7 OP) = OFF THEN
               BEGIN SEQUENCE;
               WHEN INTERRUPT <TIMER> EQUAL TO 1 SEC OCCURS
               GOTO STEP 2430;
               DELAY 2 SEC;
               END SEQUENCE;
            END SEQUENCE;

// OPEN PV9 //

         STATEMENT GROUP FOR (RECONFIGURE) = (74)
            BEGIN SEQUENCE;

            RECORD TEXT (CHECKING PV9 ...)
                   TO <PAGE-B> LINE 8 COLUMN 3 CYAN;

            IF (PV9 OP) = OFF THEN
               BEGIN SEQUENCE;
               WHEN INTERRUPT <TIMER> EQUAL TO 1 SEC OCCURS
               GOTO STEP 2300;
               DELAY 2 SEC;
               END SEQUENCE;
            END SEQUENCE;

// DE-ENERGIZE PV9 CLOSING SOLENOID, LV29 //

         STATEMENT GROUP FOR (RECONFIGURE) = (75)
            BEGIN SEQUENCE;

            RECORD TEXT (CHECKING LV29...)
                   TO <PAGE-B> LINE 8 COLUMN 3 CYAN;

            VERIFY <V41X1507E1> = ON THEN
               BEGIN SEQUENCE;
               WHEN INTERRUPT <TIMER> EQUAL TO 1 SEC OCCURS
               GOTO STEP 2320 ;
               DELAY 2 SEC;
               END SEQUENCE;
            END SEQUENCE;

// DE-ENERGIZE PV9 OPENING SOLENOID, LV28 //

         STATEMENT GROUP FOR (RECONFIGURE) = (76)
            BEGIN SEQUENCE;

            RECORD TEXT (CHECKING LV28...)
                   TO <PAGE-B> LINE 8 COLUMN 3 CYAN;

            VERIFY <V41X1508E1> = ON THEN
               BEGIN SEQUENCE;
               WHEN INTERRUPT <TIMER> EQUAL TO 1 SEC OCCURS
               GOTO STEP 2310;
               DELAY 2 SEC;
               END SEQUENCE;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
            END SEQUENCE;

// OPEN PV10 //

          STATEMENT GROUP FOR (RECONFIGURE) = (77)
            BEGIN SEQUENCE;

            RECORD TEXT (CHECKING PV10...)
                   TO <PAGE-B> LINE 8 COLUMN 3 CYAN;

            IF (PV10 OP) = OFF THEN
               BEGIN SEQUENCE;
               WHEN INTERRUPT <TIMER> EQUAL TO 1 SEC OCCURS
               GOTO STEP 2350;
               DELAY 2 SEC;
               END SEQUENCE;
            END SEQUENCE;

// DE-ENERGIZE PV10 CLOSING SOLENOID, LV31 //

          STATEMENT GROUP FOR (RECONFIGURE) = (78)
            BEGIN SEQUENCE;

            RECORD TEXT (CHECKING LV31...)
                   TO <PAGE-B> LINE 8 COLUMN 3 CYAN;

            VERIFY <V41X1505E1> = ON THEN
               BEGIN SEQUENCE;
               WHEN INTERRUPT <TIMER> EQUAL TO 1 SEC OCCURS
               GOTO STEP 2370 ;
               DELAY 2 SEC;
               END SEQUENCE;
            END SEQUENCE;

// DE-ENERGIZE PV10 OPENING SOLENOID, LV30 //

         STATEMENT GROUP FOR (RECONFIGURE) = (79)
            BEGIN SEQUENCE;

            RECORD TEXT (CHECKING LV30...)
                   TO <PAGE-B> LINE 8 COLUMN 3 CYAN;

            VERIFY <V41X1506E1> = ON THEN
               BEGIN SEQUENCE;
               WHEN INTERRUPT <TIMER> EQUAL TO 1 SEC OCCURS
               GOTO STEP 2360;
               DELAY 2 SEC;
               END SEQUENCE;
            END SEQUENCE;

// OPEN PV19 BY DE-ENERGIZING CLOSING SOLENOID, LV76 //

          STATEMENT GROUP FOR (RECONFIGURE) = (80)
            BEGIN SEQUENCE;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
            RECORD TEXT (CHECKING PV19/LV76...)
                   TO <PAGE-B> LINE 8 COLUMN 3 CYAN;

            IF (PV19 OP) = OFF THEN
               BEGIN SEQUENCE;
               WHEN INTERRUPT <TIMER> EQUAL TO 1 SEC OCCURS
               GOTO STEP 2405;
               DELAY 2 SEC;
               END SEQUENCE;
            END SEQUENCE;

// OPEN PV20 BY DE-ENERGIZING CLOSING SOLENOID, LV77 //

         STATEMENT GROUP FOR (RECONFIGURE) = (81)
            BEGIN SEQUENCE;

            RECORD TEXT (CHECKING PV20/LV77...)
                   TO <PAGE-B> LINE 8 COLUMN 3 CYAN;

            IF (PV20 OP) = OFF THEN
               BEGIN SEQUENCE;
               WHEN INTERRUPT <TIMER> EQUAL TO 1 SEC OCCURS
               GOTO STEP 2455;
               DELAY 2 SEC;
               END SEQUENCE;
            END SEQUENCE;

// OPEN PV21 BY DE-ENERGIZING CLOSING SOLENOID, LV78 //

         STATEMENT GROUP FOR (RECONFIGURE) = (82)
            BEGIN SEQUENCE;

            RECORD TEXT (CHECKING PV21/LV78...)
                   TO <PAGE-B> LINE 8 COLUMN 3 CYAN;

            IF (PV21 OP) = OFF THEN
               BEGIN SEQUENCE;
               WHEN INTERRUPT <TIMER> EQUAL TO 1 SEC OCCURS
               GOTO STEP 2505;
               DELAY 2 SEC;
               END SEQUENCE;
            END SEQUENCE;


// ENERGIZE LV60 //

         STATEMENT GROUP FOR (RECONFIGURE) = (83)
            BEGIN SEQUENCE;

            RECORD TEXT (CHECKING LV60... )
                   TO <PAGE-B> LINE 8 COLUMN 3 CYAN;

         // CHECK LV7 AND LV8 ARE ENERGIZE BEFORE LV60 IS ENERGIZED //

               VERIFY <V41X1645E1> IS ON AND
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
                        <V41X1646E1> IS ON AND
                        <V41X1170E1> IS OFF  THEN
                  BEGIN SEQUENCE;
                  WHEN INTERRUPT <TIMER> EQUAL TO 1 SEC OCCURS GOTO STEP 3091;
                  DELAY 2 SEC;
                  END SEQUENCE;
               END SEQUENCE;

// ENERGIZE LV62 //

            STATEMENT GROUP FOR (RECONFIGURE) = (84)
            BEGIN SEQUENCE;

               RECORD TEXT (CHECKING LV62... )
                      TO <PAGE-B> LINE 8 COLUMN 3 CYAN;

           // CHECK LV7 AND LV8 ARE ENERGIZE BEFORE LV62 IS ENERGIZED //

                  VERIFY <V41X1645E1> IS ON AND
                           <V41X1646E1> IS ON AND
                           <V41X1270E1> IS OFF THEN
                  BEGIN SEQUENCE;
                  WHEN INTERRUPT <TIMER> EQUAL TO 1 SEC OCCURS GOTO STEP 3092;
                  DELAY 2 SEC;

                 END SEQUENCE;

            END SEQUENCE;


// ENERGIZE LV64 //

            STATEMENT GROUP FOR (RECONFIGURE) = (85)
            BEGIN SEQUENCE;

               RECORD TEXT (CHECKING LV64... )
                      TO <PAGE-B> LINE 8 COLUMN 3 CYAN;

                  // CHECK LV7 AND LV8 ARE ENERGIZE BEFORE LV64 IS ENERGIZED //

                  VERIFY <V41X1645E1> IS ON AND
                           <V41X1646E1> IS ON AND
                           <V41X1370E1> IS OFF  THEN
                  BEGIN SEQUENCE;
                  WHEN INTERRUPT <TIMER> EQUAL TO 1 SEC OCCURS GOTO STEP 3093;
                  DELAY 2 SEC;
                  END SEQUENCE;
            END SEQUENCE;

// RECONFIGURATION COMPLETE //

            STATEMENT GROUP FOR (RECONFIGURE) = (86)
            BEGIN SEQUENCE;

               CLEAR <PAGE-B> LINE 7 TO LINE 10 COLUMN 1 TO COLUMN 70;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
RECORD TEXT (ORBITER STAND-BY RECONFIGURATION COMPLETE   )
       TO <PAGE-B> LINE 7 COLUMN 1 GREEN;

DELAY 3 SEC;

CLEAR <LED2>;
TURN OFF <PFPK2-L1>;

IF (ALL TESTS) = FALSE THEN
   BEGIN SEQUENCE;
   RECORD TEXT (TERM PROGRAM     ) TO <LED5>;
   TURN ON <PFPK5-L1>;
   END SEQUENCE;

IF (RECONFIG ERROR) = TRUE THEN
   BEGIN SEQUENCE;
   RECORD TEXT(ERRORS WERE ENCOUNTERED DURING THE RECONFIGURE)
          TEXT( PROCESS.) TO <PAGE-B> LINE 8 COLUMN 1 RED;
   RECORD TEXT(AFFECTED AREAS ARE INDICATED BY THE RED ARROWS)
          TO <PAGE-B> LINE 9 COLUMN 1 RED,
          GRAPHIC (36L) TO <PAGE-B> LINE 9 COLUMN 48 INVERT RED,
          TEXT(.) TO <PAGE-B> LINE 9 COLUMN 49 RED;

IF (TEST IN PROGRESS) = ON THEN
   BEGIN SEQUENCE;
   ASSIGN (LED1) = TEXT(CONTINUE         );
   TURN ON <PFPK1-L1>;

   RECORD TEXT (        TERM TEST) TO <LED2>;
   TURN ON <PFPK2-L1>;

   RECORD TEXT (PFPK2: TERM TEST)
          TO <PAGE-B> LINE 10 COLUMN 55 YELLOW;
   END SEQUENCE;

END SEQUENCE;

ASSIGN (RECONFIG IN PROGRESS) = FALSE;
LET (RECONFIGURE)= 0;

IF (PROGRAM TERMINATING) = TRUE THEN
   BEGIN SEQUENCE;

   CLEAR <PAGE-B> LINE 7 TO LINE 10 COLUMN 1 TO COLUMN 70;

   RECORD TEXT (PROGRAM TERMINATING                          )
          TO <PAGE-B> LINE 7 COLUMN 1 CYAN;

   DELAY 2 SEC;

   GOTO STEP 9999;

   END SEQUENCE;
END SEQUENCE;
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
          END STATEMENT GROUPS FOR (RECONFIGURE);

          GOTO STEP 10;


//*************************************************************************
**//
// END OF STANDBY CONFIGURATION SETUP //
//*************************************************************************
**//
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.6.4.3.1.5. Post-Requisites
3.6.4.3.1.6. State Diagram
3.6.4.3.1.7. State Table
3.6.4.3.1.8. Descriptive Text
3.6.4.3.2. DATA_ANALYSIS()
3.6.4.3.2.1. Parameters
3.6.4.3.2.2. Effectivity
3.6.4.3.2.3. Prerequisites
3.6.4.3.2.4. Sequence
3.6.4.3.2.5. Post-Requisites
3.6.4.3.2.6. State Diagram
3.6.4.3.2.7. State Table
3.6.4.3.2.8. Descriptive Text
3.6.4.3.3. START_DECAY()
3.6.4.3.3.1. Parameters
3.6.4.3.3.2. Effectivity
3.6.4.3.3.3. Prerequisites
3.6.4.3.3.4. Sequence
3.6.4.3.3.5. Post-Requisites
3.6.4.3.3.6. State Diagram
3.6.4.3.3.7. State Table
3.6.4.3.3.8. Descriptive Text
3.6.4.3.4. CHECK_STANDBY()
3.6.4.3.4.1. Parameters
3.6.4.3.4.2. Effectivity
3.6.4.3.4.3. Prerequisites
3.6.4.3.4.4. Sequence
3.6.4.3.4.5. Post-Requisites
3.6.4.3.4.6. State Diagram
3.6.4.3.4.7. State Table
3.6.4.3.4.8. Descriptive Text

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.6.4.3.5. MONITOR_LOOP()
3.6.4.3.5.1. Parameters
3.6.4.3.5.2. Effectivity
3.6.4.3.5.3. Prerequisites
3.6.4.3.5.4. Sequence
3.6.4.3.5.5. Post-Requisites
3.6.4.3.5.6. State Diagram
3.6.4.3.5.7. State Table
3.6.4.3.5.8. Descriptive Text
3.6.4.3.6. SET_PRESSURE_LIMITS()
3.6.4.3.6.1. Parameters
3.6.4.3.6.2. Effectivity
3.6.4.3.6.3. Prerequisites
3.6.4.3.6.4. Sequence
3.6.4.3.6.5. Post-Requisites
3.6.4.3.6.6. State Diagram
3.6.4.3.6.7. State Table
3.6.4.3.6.8. Descriptive Text
3.6.4.3.7. INITIAL_SETUP()
3.6.4.3.7.1. Parameters
3.6.4.3.7.2. Effectivity
3.6.4.3.7.3. Prerequisites
3.6.4.3.7.4. Sequence
3.6.4.3.7.5. Post-Requisites
3.6.4.3.7.6. State Diagram
3.6.4.3.7.7. State Table
3.6.4.3.7.8. Descriptive Text
3.6.4.3.8. VALVE_CONFIGURATION_A()
3.6.4.3.8.1. Parameters
3.6.4.3.8.2. Effectivity
3.6.4.3.8.3. Prerequisites

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.6.4.3.8.4. Sequence

3.6.4.3.8.5. Post-Requisites

3.6.4.3.8.6. State Diagram

3.6.4.3.8.7. State Table

3.6.4.3.8.8. Descriptive Text

3.6.4.3.9. VALVE_CONFIGURATION_B()

3.6.4.3.9.1. Parameters

3.6.4.3.9.2. Effectivity

3.6.4.3.9.3. Prerequisites

3.6.4.3.9.4. Sequence

```
//V41AZ0.100B

CURRENT_TEST(B);  //ASSINGNS WHICH TEST IS IN PROGRESS
CURRENT_TEST(TELL)   // RUNS CURRENT_TEST() AND RECORDS WHICH TEST IS
CURRENTLY PERFORMED.

         ASSIGN (TEST IN PROGRESS) = ON;
         (TEST LOCATION B) = 0;
         (REG OUT ALLOWABLE) = 2.4 PSI/MIN;
         (ACCUM ALLOWABLE)   = 1.6 PSI/MIN;

         RECORD TEXT (TEST V41AZ0.100-B IN PROGRESS. STAND-BY...)
                TO <PAGE-B>
         RECORD <GMT> ,TEXT (   TEST: V41AZ0.100-B)
                TO (PRINTERS)
         DELAY 3 SEC;

         (REG OUT ALLOWABLE),(ACCUM ALLOWABLE) TO <PAGE-B

// INITIATE TIMERS

            (DELAY REMAINING) = (DELAY);
             (DECAY REMAINING) = (DECAY);

            RECORD (DELAY REMAINING)
                   (DECAY REMAINING) TO <PAGE-B>

//RUN STANDBY CONFIGURATION CHECK

CHECK_ORBITER_STBY_CONFIG();

//ENERGIZE LV75

            RECORD TEXT (CONFIGURING VALVES/SOLENOIDS FOR DECAY TEST B )
                   TO <PAGE-B>
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
            READ <V41P1433C1> AND SAVE AS (LH2 MANIFOLD BEFORE);

            VERIFY <V41X1902E1> IS OFF THEN

               BEGIN SEQUENCE;

            OPEN.MPS_LV75();
            DELAY 3 SEC

               END SEQUENCE;

//CHECK LH2 MANIFOLD FOR SUDDEN INCREASE OF PRESSURE.

            VERIFY <V41P1433C1> IS GREATER THAN (LH2 MANIF PLUS1) THEN
               BEGIN SEQUENCE;

               RECORD TEXT (LH2 MANIFOLD PRESSURE INCREASED AFTER ENERGIZING
LV75)
                      TO <PAGE-B>

                VERIFY <V41X1902E1> IS ON THEN
                    BEGIN SEQUENCE;

                    CLOSE.MPS_LV75()
                 DELAY 3 SEC

                     END SEQUENCE;

               VERIFY <V41X1902E1> IS OFF THEN

RECORD TEXT (PROGRAM DE-ENERGIZED LV75. EVALUATE & ENERGIZE LV75 BEFORE
CONTINUING) TO <PAGE-B> RED;

END SEQUENCE

//ENERGIZE LV41

            READ <V41P1533C1> AND SAVE AS (LO2 MANIFOLD BEFORE);

            VERIFY <V41X1539E1> IS OFF THEN
               BEGIN SEQUENCE;

               OPEN.MPS_LV41();

               END SEQUENCE;

// CHECK LO2 MANIFOLD FOR SUDDEN INCREASE IN PRESSURE.

            LET (LO2 MANIF PLUS1) = (LO2 MANIFOLD BEFORE) +1 PSIA;
            VERIFY <V41P1533C1> IS GREATER THAN (LO2 MANIF PLUS1) THEN

BEGIN SEQUENCE;
               RECORD TEXT (LO2 MANIFOLD PRESSURE INCREASED AFTER ENERGIZING
LV41)
                      TO <PAGE-B>
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
                  VERIFY <V41X1539E1> IS ON THEN
                  BEGIN SEQUENCE;

CLOSE.MPS_LV41();

END SEQUENCE;

               VERIFY <V41X1539E1> IS OFF THEN
                  BEGIN SEQUENCE;

                  RECORD TEXT (PROGRAM DE-ENERGIZED LV41.  EVALUATE &
ENERGIZE LV41 BEFORE CONTINUING) TO <PAGE-B> LINE 9 COLUMN  1 RED;

                  END SEQUENCE;
               END SEQUENCE;

//ENERGIZE LV43

            READ <V41P1433C1> AND SAVE AS (LH2 MANIFOLD BEFORE);

            VERIFY <V41X1438E1> IS OFF THEN OPEN.MPS_LV43()

//CHECK LH2 MANIFOLD FOR SUDDEN INCREASE IN PRESSURE

            LET (LH2 MANIF PLUS1) = (LH2 MANIFOLD BEFORE) +1 PSIA;

            VERIFY <V41P1433C1> IS GREATER THAN (LH2 MANIF PLUS1) THEN
               BEGIN SEQUENCE;

               RECORD TEXT (LH2 MANIFOLD PRESSURE INCREASED AFTER ENERGIZING
LV43)
                      TO <PAGE-B> 1 RED;

               VERIFY <V41X1438E1> IS ON THEN

                  BEGIN SEQUENCE;
                  CLOSE.MPS_LV43();
                  DELAY 2 SEC;
                  END SEQUENCE;

               VERIFY <V41X1438E1> IS OFF THEN
                  BEGIN SEQUENCE;
                  DELAY 5 SEC;
RECORD TEXT (PROGRAM DE-ENERGIZED LV43.  EVALUATE & ENERGIZE LV43
BEFORE CONTINUING) TO <PAGE-B> RED;

                  END SEQUENCE;
               END SEQUENCE;

//ENERGIZE LV27
            VERIFY <V41X1634E1> IS OFF THEN
               BEGIN SEQUENCE;

               OPEN.MPS_LV27();
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
               END SEQUENCE;

//ENERGIZE LV80
            VERIFY <V41X1144E1> IS OFF THEN
               BEGIN SEQUENCE;

               OPEN.MPS_LV80();

               END SEQUENCE;

//ENERGIZE LV81

            VERIFY <V41X1244E1> IS OFF THEN
               BEGIN SEQUENCE;

               OPEN.MPS_LV81();
               DELAY 2 SEC;

               END SEQUENCE;

//ENERGIZE LV82

            VERIFY <V41X1344E1> IS OFF THEN

               OPEN.MPS_LV82();

//DELAY FOR THERMAL STABILIZATION

RECORD (DELAY), (MINUTE DELAY FOR THERMAL STABILIZATION IN PROGRESS. PLEASE
STANDBY..) TO <PAGE-B>

THERMAL_STABILIZATION(15 MIN);

//DE-ENERGIZE LV7 .

      VERIFY <V41X1645E1> IS ON THEN
CLOSE.MPS_LV7();

//DE-ENERGIZE LV8

        VERIFY <V41X1646E1> IS ON THEN

      CLOSE.MPS_LV8();

//ENERGIZE LV32

VERIFY <V41X1386E1> IS OFF THEN

OPEN.MPS_LV32();

//DE-ENERGIZE LV32

      VERIFY <V41X1386E1> IS ON THEN
CLOSE.MPS_LV32();
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
//CHECK PRESSURE DIFFERENCE ACROSS (CV9)

            RECORD TEXT (VERIFYING PRESSURE DROP ACROSS CV9)
                   TO <PAGE-B>
DELAY 3 SEC;

            READ <V41P1605A1> AND SAVE AS (PREG);
            READ <V41P1650A1> AND SAVE AS (PACCUM);

            LET (PRESS DIFF) = (PACCUM) - (PREG);

            IF (PRESS DIFF) IS LESS THAN 50 PSIA THEN
               BEGIN SEQUENCE;

RECORD TEXT ("PRESSURE DIFFERENCE BETWEEN ACCUM AND REGOUT IS"),(PRESS DIFF)
,(" .A MINIMUM OF 50 PSIA IS REQUIRED. EVALUATE BEFORE PROCEEDING. )
                      TO <PAGE-B> RED;

MONITOR_LOOP();
              END SEQUENCE;

// 2 MINUTE DELAY

RECORD ("TWO [2] MINUTE DELAY IN PROGRESS [BEGAN AT:"),<GMT>,("PLEASE STAND-
BY...)"                    TO <PAGE-B>
            DELAY 2 SEC;

THERMAL_STABILIZATION(2 MIN);

// START DECAY

            RECORD (DECAY) ,"MINUTE DECAY TEST IN PROGRESS..."  TO <PAGE-B>

            ASSIGN (START DECAY)= ON;
            START_DECAY();

// DECAY TEST COMPLETE

             RECORD TEXT (TEST V41AZ0.100-[B] COMPLETE. END TIME: ),(REG OUT
GMT END)
                    TEXT (RESULTS DISPLAYED BELOW. MAKE HARD-COPY)
                   TO <PAGE-B>

//  RECONFIGURE TO STANDBY MODE $

            RECONFIGURE_TO_STANDBY();

// TEST STEPS COMPLETE

RECORD <TEST B COMPLETE FLAG> TO <PAGE-B>

            ASSIGN (TEST IN PROGRESS) = OFF;
            RECORD TEXT(PROGRAM IN MONITOR LOOP. SELECT AN OPTION) TO <PAGE-
B>
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.6.4.3.9.5. Post-Requisites

3.6.4.3.9.6. State Diagram

3.6.4.3.9.7. State Table

3.6.4.3.9.8. Descriptive Text

3.6.4.3.10. VALVE_CONFIGURATION_C()

3.6.4.3.10.1. Parameters

3.6.4.3.10.2. Effectivity

3.6.4.3.10.3. Prerequisites

3.6.4.3.10.4. Sequence

```
//V41AZ0.100C

{
         ASSIGN (TEST) = TEXT(C);
         ASSIGN (TEST IN PROGRESS) = ON;

         LET (REG OUT ALLOWABLE) = 2.0 PSI/MIN;
         LET (ACCUM ALLOWABLE)   = 1.6 PSI/MIN;

         RECORD TEXT (TEST V41AZ0.100-C IN PROGRESS. STAND-BY...)
                TO <PAGE-B>
         RECORD <GMT> , TEXT (TEST: V41AZ0.100-C)
                TO (PRINTERS)
         DELAY 3 SEC;

         RECORD TEXT (C),(REG OUT ALLOWABLE), (ACCUM ALLOWABLE) TO <PAGE-B>

//INITIALIZE TIMERS

            LET (DELAY REMAINING) = (DELAY);
            LET (DECAY REMAINING) = (DECAY);

            RECORD (DELAY REMAINING)
                   (DECAY REMAINING)
                   TO <PAGE-B>

//RUN STANDBY CONFIGURATION CHECK

CHECK_ORBITER_STBY_CONFIG();

//ENERGIZE LV49

            RECORD TEXT (CONFIGURING VALVES/SOLENOIDS FOR DECAY TEST C )
                   TO <PAGE-B>
            VERIFY <V41X1381E1> IS OFF THEN
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
OPEN.MPS_LV49();


//ENERGIZE LV48

            VERIFY <V41X1382E1> IS OFF THEN

OPEN.MPS_LV48();

//DE-ENERGIZE LV49

            VERIFY <V41X1381E1> IS ON THEN
               CLOSE.MPS_LV49();


//ENERGIZE LV67

         STATEMENT GROUP FOR (TEST LOCATION C) = (6)
            BEGIN SEQUENCE;

OPEN.MPS_LV67();


//ENERGIZE LV50

OPEN.MPS_LV50()

//ENERGIZE LV18

OPEN.MPS_LV18();


//ENERGIZE LV20


OPEN.MPS_LV20();


// ENERGIZE LV22


OPEN.MPS_LV22()


// ENERGIZE LV34

OPEN.MPS_LV34();

//ENERGIZE LV39

OPEN.MPS_LV39();
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
//ENERGIZE LV36

OPEN.MPS_LV36();

//ENERGIZE LV72

OPEN.MPS_LV72();

//ENERGIZE LV79

OPEN.MPS_LV79();

//ENERGIZE LV47

OPEN.MPS_LV47();

//ENERGIZE LV46

OPEN.MPS_LV46();

//DE-ENERGIZE LV47

OPEN.MPS_LV47();

//ENERGIZE LV65

OPEN.MPS_LV65();

//ENERGIZE LV12

   OPEN.MPS_LV12();

//ENERGIZE LV83


   OPEN.MPS_LV83();

//ENERGIZE LV14

OPEN.MPS_LV14();

//ENERGIZE LV84

OPEN.MPS_LV84();

//ENERGIZE LV16
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
OPEN.MPS_LV16();

//ENERGIZE LV85

OPEN.MPS_LV85();

//ENERGIZE LV28

OPEN.MPS_LV28();


// ENERGIZE LV30

OPEN.MPS_LV30();

//THERMAL STABILIZATION DELAY

RECORD (DELAY), (MINUTE DELAY FOR THERMAL STABILIZATION IN PROGRESS. PLEASE
STANDBY..) TO <PAGE-B>

THERMAL_STABILIZATION(15 MIN);

//DE-ENERGIZE LV7 .

      VERIFY <V41X1645E1> IS ON THEN
CLOSE.MPS_LV7();

//DE-ENERGIZE LV8

        VERIFY <V41X1646E1> IS ON THEN

      CLOSE.MPS_LV8();

//ENERGIZE LV32

            VERIFY <V41X1386E1> IS OFF THEN

CLOSE.MPS_LV8();

//CHECK PRESSURE DIFFERENCE ACROSS (CV9)

            RECORD TEXT (VERIFYING PRESSURE DROP ACROSS CV9)
                   TO <PAGE-B>
DELAY 3 SEC;

            READ <V41P1605A1> AND SAVE AS (PREG);
            READ <V41P1650A1> AND SAVE AS (PACCUM);

            LET (PRESS DIFF) = (PACCUM) - (PREG);

            IF (PRESS DIFF) IS LESS THAN 50 PSIA THEN
               BEGIN SEQUENCE;

RECORD TEXT (“PRESSURE DIFFERENCE BETWEEN ACCUM AND REGOUT IS”),(PRESS DIFF)
,(” .A MINIMUM OF 50 PSIA IS REQUIRED. EVALUATE BEFORE PROCEEDING. )
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

```
                    TO <PAGE-B> RED;

MONITOR_LOOP();
             END SEQUENCE;

// 2 MINUTE DELAY

RECORD (“TWO [2] MINUTE DELAY IN PROGRESS [BEGAN AT:”),<GMT>,(”PLEASE STAND-
BY...)”                   TO <PAGE-B>
            DELAY 2 SEC;

THERMAL_STABILIZATION(2 MIN);

// START DECAY
            RECORD (DECAY) ,”MINUTE DECAY TEST IN PROGRESS...”  TO <PAGE-B>

            ASSIGN (START DECAY)= ON;
            START_DECAY();

// DECAY TEST COMPLETE

             RECORD TEXT (TEST V41AZ0.100-[C] COMPLETE. END TIME: ),(REG OUT
GMT END)
                   TEXT (RESULTS DISPLAYED BELOW. MAKE HARD-COPY)
                  TO <PAGE-B>

//  RECONFIGURE TO STANDBY MODE $

            RECONFIGURE_TO_STANDBY();

// TEST STEPS COMPLETE

RECORD <TEST C COMPLETE FLAG> TO <PAGE-B>

            ASSIGN (TEST IN PROGRESS) = OFF;
            RECORD TEXT(PROGRAM IN MONITOR LOOP. SELECT AN OPTION) TO <PAGE-
B>
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.6.4.3.10.5. Post-Requisites
3.6.4.3.10.6. State Diagram
3.6.4.3.10.7. State Table
3.6.4.3.10.8. Descriptive Text
3.6.4.3.11. VALVE_CONFIGURATION_D()
3.6.4.3.11.1. Parameters
3.6.4.3.11.2. Effectivity
3.6.4.3.11.3. Prerequisites
3.6.4.3.11.4. Sequence
3.6.4.3.11.5. Post-Requisites
3.6.4.3.11.6. State Diagram
3.6.4.3.11.7. State Table
3.6.4.3.11.8. Descriptive Text
3.6.4.3.12. VALVE_CONFIGURATION_E()
3.6.4.3.12.1. Parameters
3.6.4.3.12.2. Effectivity
3.6.4.3.12.3. Prerequisites
3.6.4.3.12.4. Sequence
3.6.4.3.12.5. Post-Requisites
3.6.4.3.12.6. State Diagram
3.6.4.3.12.7. State Table
3.6.4.3.12.8. Descriptive Text
3.6.4.3.13. VALVE_CONFIGURATION_ALL()
3.6.4.3.13.1. Parameters
3.6.4.3.13.2. Effectivity
3.6.4.3.13.3. Prerequisites
3.6.4.3.13.4. Sequence
3.6.4.3.13.5. Post-Requisites
3.6.4.3.13.6. State Diagram
3.6.4.3.13.7. State Table
3.6.4.3.13.8. Descriptive Text

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.6.4.3.14. TROUBLE_SHOOTING()

3.6.4.3.14.1. Parameters

3.6.4.3.14.2. Effectivity

3.6.4.3.14.3. Prerequisites

3.6.4.3.14.4. Sequence

3.6.4.3.14.5. Post-Requisites

3.6.4.3.14.6. State Diagram

3.6.4.3.14.7. State Table

3.6.4.3.14.8. Descriptive Text

3.6.4.3.15. DECAY_RATE_RESULTS()

3.6.4.3.15.1. Parameters

3.6.4.3.15.2. Effectivity

3.6.4.3.15.3. Prerequisites

3.6.4.3.15.4. Sequence

3.6.4.3.15.5. Post-Requisites

3.6.4.3.15.6. State Diagram

3.6.4.3.15.7. State Table

3.6.4.3.15.8. Descriptive Text

3.6.4.3.16. LEG_VOLUME()

3.6.4.3.16.1. Parameters

3.6.4.3.16.2. Effectivity

3.6.4.3.16.3. Prerequisites

3.6.4.3.16.4. Sequence

```
(GROUND LEG) = 185

// PV9 //
             VERIFY <V41X1508E1> IS ON  AND
                <V41X1507E1> IS OFF THEN
                 (GROUND LEG) = (GROUND LEG) + 61.6

             VERIFY <V41X1507E1> IS ON  AND
                <V41X1508E1> IS OFF THEN
                 (GROUND LEG) = (GROUND LEG) + 91.2
// PV10 //
             VERIFY <V41X1506E1> IS ON  AND
                 <V41X1505E1> IS OFF THEN
                  (GROUND LEG) = (GROUND LEG) + 61.6
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
             VERIFY <V41X1505E1> IS ON  AND
                 <V41X1506E1> IS OFF THEN
                  (GROUND LEG) = (GROUND LEG) + 91.2
// PV11 //
             VERIFY <V41X1386E1> IS ON  AND
                 <V41X1385E1> IS OFF THEN
                  (GROUND LEG) = (GROUND LEG) + 31.4

             VERIFY <V41X1385E1> IS ON  AND
                 <V41X1386E1> IS OFF THEN
                  (GROUND LEG) = (GROUND LEG) + 61.4
// PV12 //
             VERIFY <V41X1406E1> IS ON  AND
                 <V41X1405E1> IS OFF THEN
                  (GROUND LEG) = (GROUND LEG) + 31.4

             VERIFY <V41X1405E1> IS ON  AND
                 <V41X1406E1> IS OFF THEN
                  (GROUND LEG) = (GROUND LEG) + 61.4
// PV13 //
             VERIFY <V41X1458E1> IS ON THEN
                  (GROUND LEG) = (GROUND LEG) + 2.8
// PV14,15,16 //
             VERIFY <V41X1112E1> IS ON THEN
                  (GROUND LEG) = (GROUND LEG) + 15.8
// PV20 //
             VERIFY <V41X1813E1> IS ON THEN
                  (GROUND LEG) = (GROUND LEG) + 1.8
// PV21 //
             VERIFY <V41X1823E1> IS ON THEN
                  (GROUND LEG) = (GROUND LEG) + 1.8


             RECORD (GROUND LEG) TO <PAGE-B>


// FLIGHT LEG - PV1, PV2, PV3, PV4, PV5, PV6, PV7, PV8, PV17, PV18, PV19,
PV22, PD1, PD2, & PD3 //


              (FLIGHT LEG) = 1175
// PV1 //
             VERIFY <V41X1133E1> IS ON AND
                <V41X1145E1> IS ON THEN
                 (FLIGHT LEG) = (FLIGHT LEG) + 27.8

             VERIFY <V41X1132E1> IS ON THEN
                 (FLIGHT LEG) = (FLIGHT LEG) + 27.8

             VERIFY <V41X1144E1> IS ON THEN
                 (FLIGHT LEG) = (FLIGHT LEG) + 27.8
// PV2 //
             VERIFY <V41X1233E1> IS ON AND
                 <V41X1245E1> IS ON THEN
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
                (FLIGHT LEG) = (FLIGHT LEG) + 27.8

             VERIFY <V41X1232E1> IS ON THEN
                 (FLIGHT LEG) = (FLIGHT LEG) + 27.8

             VERIFY <V41X1244E1> IS ON THEN
                 (FLIGHT LEG) = (FLIGHT LEG) + 27.8
// PV3 //
             VERIFY <V41X1333E1> IS ON AND
                 <V41X1345E1> IS ON THEN
                (FLIGHT LEG) = (FLIGHT LEG) + 27.8

             VERIFY <V41X1332E1> IS ON THEN
                 (FLIGHT LEG) = (FLIGHT LEG) + 27.8

             VERIFY <V41X1344E1> IS ON THEN
                 (FLIGHT LEG) = (FLIGHT LEG) + 27.8
// PV4 //
             VERIFY <V41X1103E1> IS ON THEN
                 (FLIGHT LEG) = (FLIGHT LEG) + 27.8

             VERIFY <V41X1102E1> IS ON THEN
                 (FLIGHT LEG) = (FLIGHT LEG) + 27.8
// PV5 //
             VERIFY <V41X1203E1> IS ON THEN
                 (FLIGHT LEG) = (FLIGHT LEG) + 27.8

             VERIFY <V41X1202E1> IS ON THEN
                 (FLIGHT LEG) = (FLIGHT LEG) + 27.8
// PV6 //
             VERIFY <V41X1303E1> IS ON THEN
                 (FLIGHT LEG) = (FLIGHT LEG) + 27.8

             VERIFY <V41X1302E1> IS ON THEN
                 (FLIGHT LEG) = (FLIGHT LEG) + 27.8
// PV7 //
             VERIFY <V41X1549E1> IS ON THEN
                 (FLIGHT LEG) = (FLIGHT LEG) + 3.4
// PV8 //
             VERIFY <V41X1449E1> IS ON THEN
                 (FLIGHT LEG) = (FLIGHT LEG) + 3.4
// PV17 //
             VERIFY <V41X1921E1> IS ON THEN
                 (FLIGHT LEG) = (FLIGHT LEG) + 2.8
// PV18 //
             VERIFY <V41X1911E1> IS ON THEN
                 (FLIGHT LEG) = (FLIGHT LEG) + 2.8
// PV19 //
             VERIFY <V41X1582E1> IS ON THEN
                 (FLIGHT LEG) = (FLIGHT LEG) + 2.8
// PV22 //
             VERIFY <V41X1467E1> IS ON THEN
                 (FLIGHT LEG) = (FLIGHT LEG) + 2.8
// PD1 //
             VERIFY <V41X1807E1> IS ON  AND
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

```
                <V41X1806E1> IS OFF THEN
                (FLIGHT LEG) = (FLIGHT LEG) + 17.0

            VERIFY <V41X1806E1> IS ON  AND
                <V41X1807E1> IS OFF THEN
                (FLIGHT LEG) = (FLIGHT LEG) + 17.7
// PD2 //
            VERIFY <V41X1382E1> IS ON  AND
                <V41X1381E1> IS OFF THEN
                (FLIGHT LEG) = (FLIGHT LEG) + 17.0

            VERIFY <V41X1381E1> IS ON  AND
                <V41X1382E1> IS OFF THEN
                (FLIGHT LEG) = (FLIGHT LEG) + 17.7
// PD3 //
            VERIFY <V41X1440E1> IS ON  AND
                <V41X1439E1> IS OFF THEN
                (FLIGHT LEG) = (FLIGHT LEG) + 1.0

            VERIFY <V41X1439E1> IS ON  AND
                <V41X1440E1> IS OFF THEN
                (FLIGHT LEG) = (FLIGHT LEG) + 1.0

            RETURN(FLIGHT LEG) TO <PAGE-B>
```


Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for
current approved version of this document before using it for work

3.6.4.3.16.5. Post-Requisites
3.6.4.3.16.6. State Diagram
3.6.4.3.16.7. State Table
3.6.4.3.16.8. Descriptive Text
3.6.4.3.17. CV9_DELTA_PRESSURE()
3.6.4.3.17.1. Parameters
3.6.4.3.17.2. Effectivity
3.6.4.3.17.3. Prerequisites
3.6.4.3.17.4. Sequence
3.6.4.3.17.5. Post-Requisites
3.6.4.3.17.6. State Diagram
3.6.4.3.17.7. State Table
3.6.4.3.17.8. Descriptive Text
3.6.4.3.18. HIGH_MANIFOLD_PRESS()
3.6.4.3.18.1. Parameters
3.6.4.3.18.2. Effectivity
3.6.4.3.18.3. Prerequisites
3.6.4.3.18.4. Sequence
3.6.4.3.18.5. Post-Requisites
3.6.4.3.18.6. State Diagram
3.6.4.3.18.7. State Table
3.6.4.3.18.8. Descriptive Text
3.6.4.3.19. F1()
3.6.4.3.19.1. Parameters
3.6.4.3.19.2. Effectivity
3.6.4.3.19.3. Prerequisites
3.6.4.3.19.4. Sequence
3.6.4.3.19.5. Post-Requisites
3.6.4.3.19.6. State Diagram

<insert state diagram here>

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

3.6.4.3.19.7. State Table

3.6.4.3.19.8. Descriptive Text

3.6.4.4. User Interfaces

3.6.4.5. Safety Requirements

4. DISPLAYS

4.1. GENERAL

1. MPS Orbiter systems will be displayed in schematic, tabular, and/or enumerated state format as applicable and specified in this SRS. The operator interface to CLCS controlled components will provide command and control capablitity with methods specified in this section, and will generally be accessed via the system schematic display. Tabular displays shall provide information necessary to support troubleshooting and configuration verification, giving the status of all specified indicators (valve position, RPCs, power, switch scans, and system pressure and temperature measurements), including fail bit and bypass indications of these measurements. Enumerated state displays will provide a tabular view (arranged similarly to the tabular display) of component position indicator state, command state, and over-all state as specified in section 3 of this SRS.
2. The MPS orbiter system is broken down into three major subsystems: LH2, LO2 and GHe. Each of this systems will have schematic, tabular and enumerated state displays for use during orbiter test and check-out (OPF and VAB/PAD).
3. For MPS electrical system test and check-out, a display to support MDM command redundancy, copper path, and bus redundancy testing (bus 'drops') will be generated as detailed in this SRS.
4. MPS displays depicting reactive and prerequisite control logic will be generated as detailed in this section.
5. A display to support ETOI functional check-out will be required. The display will support MPS File II and file III OMRSD testing of flow control valves and external tank instrumentation electronics in the MPS point sensor electronics box.
6. All MPS component methods will be a two step operation (unless otherwise specifically specified) to prevent against inadvertent method invocation.
7. Command control of MPS GSE components (reference 84K01450-100 Vol 1) will be included on orbiter system displays where necessary to provide the operator control and safing of GSE directly interfacing with the vehicle.
8. The capability to hide/show GSE interfacing to the vehicle will be provided to the operator to reflect actual vehicle configuration.
9. The capability to hide/show applicable SSME components/systems will be provided to the operator on MPS LO2 and LH2 system schematic displays to reflect actual vehicle configuration (SSMEs installed/not installed).
10. MPS orbiter system test and check-out displays will have an "OPF" and a "VAB/PAD" view, selectable by the operator to depict vehicle location/configuration. This allows for 1 common display to be used for both horizontal and vertical processing. The applicable GSE supporting vehicle location will be diplayed in these views.
11. Color codes for MPS displays will conform to 84K01720 unless otherwise specified in this SRS.
12. MPS displays shall be cyclically updated with a period equal to (TBD).
13. The methods displayed for software debug and validation testing will be the full method list (as detailed in section 3 of this SRS) when the MPSMASTERTESTFLAG pseudo is ON. Otherwise, the method list for MPS orbiter valve components will include open/close (or latch/unlatch for PD1/2 latch components) and energize/deenergize solenoids only.
14. MPS displays designed specifically to support vehicle load and launch operations (including LCC monitoring and enunciations) will be detailed in a separate SRS to support CLCS phase 3 development.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

4.2. MPS LH2 SYSTEM TEST AND CHECK-OUT DISPLAY - SCHEMATIC

4.2.1. Description

The MPS liquid hydrogen (LH2) and gaseous hydrogen (GH2) systems will be displayed in schematic format per engineering direction and guidelines (Ref V070-415102).
In addition to the below listed CLCS components, the LH2 system test and checkout display will provide the status of LH2 system propellant valve actuator solenoids. The display will depict the solenoid command state of energized or de-energized.

4.2.2. Control Points

4.2.2.1. LH2/GH2 Manifold Repress Regulator Control

The display will provide a target box for control (activation/securing) of the LH2 manifold repress regulator (PR6) and provide access to the methods of the MPS_RepressRegCntrl class.

4.2.2.2. RTLS Press Control

The display will provide a target box for control of the LH2 RTLS press valves LV74 and LV75 and provide access to the methods of the MPS_Valve_ORB_RTLSPress class.

4.2.2.3. Auto Valve Checkout

Auto valve checkout will perform a functional cycle test of all MPS LH2 orbiter cryogenic valves on the LH2 System Test and Check-Out Display (less PD2 disconnect). The auto checkout function will be available from the pull-down menu and will invoke the autoValveCheckout() method of the MPS_LH2_Subsystem Class.

4.2.2.4. ETOI Control

The display will provide a target box for access to methods for control and check-out of orbiter-ET point sensor electronics. In addition to power-up/down methods, the user will have access to the following:
1. Launch of the ETOI test and check-out display.
2.
3.

4.2.2.5. Terminate Display

This option will allow the operator to terminate the LH2 System Test and Checkout Display. The option will be available from a target box in the lower left hand corner of the display.

4.2.3. Objects

4.2.3.1. Transducers/Level Sensors

The following transducers and ET point sensor measurements will be displayed in addition to the CLCS controlled cryogenic valves and system piping.

INSTANCE	NOMENCLATURE
MPS_MT43 (V41P1433C1)	LH2 Manifold Pressure
MPS_MT47 (V41P1490A1)	GH2 Disconnect Pressure
MPS_MT11 (V41P1160A1)	E1 GH2 Pressure

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

MPS_MT24 (V41P1260A1)	E2 GH2 Pressure
MPS_MT37 (V41P1360A1)	E3 GH2 Pressure
MPS_MT12 (V41T1161A1)	E1 GH2 Temperature
MPS_MT25 (V41T1261A1)	E2 GH2 Temperature
MPS_MT38 (V41T1361A1)	E3 GH2 Temperature
MPS_MT42 (V41T1428A1)	LH2 Feedline Temperature
MPS_MT44 (V41P1564A1)	LH2/GH2 Delta Pressure
GLHP4116A	LH2 Transfer Line Pressure (GSE) (VAB/PAD view)
GLHP4525A	LH2 ET Prepress Line Pressure (GSE) (VAB/PAD view)
MPS_PP1 (V41R1115A)	E1 LH2 Recirc Pump Speed (VAB/PAD view)
MPS_PP2 (V41R1215A)	E2 LH2 Recirc Pump Speed (VAB/PAD view)
MPS_PP3 (V41R1315A)	E3 LH2 Recirc Pump Speed (VAB/PAD view)
MPS_MT54 (V41P1100C1)	LH2 E1 Feedline Pressure
MPS_MT14 (V41P1200C1)	LH2 E2 Feedline Pressure
MPS_MT27 (V41P1300C1)	LH2 E3 Feedline Pressure
E41H1061D	SSME 1 LH2 Bleed Valve Position PCT (SSME view)
E41H2061D	SSME 2 LH2 Bleed Valve Position PCT (SSME view)
E41H3061D	SSME 3 LH2 Bleed Valve Position PCT (SSME view)
T41P1700C1	ET Ullage Pressure No.1 (VAB/PAD view)
T41P1701C1	ET Ullage Pressure No.2 (VAB/PAD view)
T41P1702C1	ET Ullage Pressure No.3 (VAB/PAD view)
T41P1703C1	ET Ullage Pressure - Spare (VAB/PAD view)
T41X1730X1	ET LH2 Low Level Sensor No. 1 (VAB/PAD view)
T41X1731X1	ET LH2 Low Level Sensor No. 2 (VAB/PAD view)
T41X1732X1	ET LH2 Low Level Sensor No. 3 (VAB/PAD view)
T41X1733X1	ET LH2 Low Level Sensor No. 4 (VAB/PAD view)

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

T41X1712E1	ET LH2 5% Level Sensor (VAB/PAD view)
T41X1715E1	ET LH2 98% Sensor No. 1 (VAB/PAD view)
T41X1716E1	ET LH2 98% Sensor No. 2 (VAB/PAD view)
T41X1718E1	ET LH2 100% Sensor No. 1 (VAB/PAD view)
T41X1719E1	ET LH2 100% Sensor No. 2 (VAB/PAD view)
T41X1717E1	ET LH2 100% - Sensor (VAB/PAD view)
T41X1720E1	ET LH2 100% + Sensor (VAB/PAD view)
T41X1721E1	ET LH2 Overfill Sensor (VAB/PAD view)
T41X1724E1	ET LH2 Vent Valve Closed Indication
T41X1727E1	ET LH2 Vent Valve Open Indication
MPS_A83108 (GHEP2074A)	LH2 Feedline GSE Supply Pressure (GSE) (OPF view)
MPS_A83114 (GHEP2094A)	GH2 Prepress GSE Supply Pressure (GSE) (OPF view)

4.2.3.2. Static Components

4.2.3.3. CLCS Components

INSTANCE	NOMENCLATURE	METHODS
MPS_PV4	E1 LH2 Prevalve	open/close, energize/deenergize open&close solenoid
MPS_PV5	E2 LH2 Prevalve	open/close, energize/deenergize open&close solenoid
MPS_PV6	E3 LH2 Prevalve	open/close, energize/deenergize open&close solenoid
MPS_PV8	LH2 Manifold Relief Isolation Valve	open/close, energize/deenergize open&close solenoid
MPS_PV11	LH2 Outboard Fill and Drain Valve	open/close, energize/deenergize open&close solenoid
MPS_PV12	LH2 Inboard Fill and Drain Valve	open/close, energize/deenergize open&close solenoid
MPS_PV13	LH2 Topping Valve	open/close, energize/deenergize open&close solenoid

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

MPS_PV14	LH2 E1 Recirc Isolation Valve	open/close, energize/deenergize open&close solenoid
MPS_PV15	LH2 E2 Recirc Isolation Valve	open/close, energize/deenergize open&close solenoid
MPS_PV16	LH2 E3 Recirc Isolation Valve	open/close, energize/deenergize open&close solenoid
MPS_PV17	LH2 RTLS Dump Valve	open/close, energize/deenergize open&close solenoid
MPS_PV18	LH2 RTLS Dump Valve	open/close, energize/deenergize open&close solenoid
MPS_PV22	LH2 High Point Bleed Valve	open/close, energize/deenergize open&close solenoid
MPS_PD2	LH2 17" Disconnect	open/close, energize/deenergize open&close solenoid
MPS_PD2_Latch	LH2 17" Disconnect Latch	lock/unlock, energize/deenergize open&close solenoid
MPS_LV36	LH2 Recirc Isolation VLV Control Solenoid	open/close

4.3. MPS LH2 SYSTEM TEST AND CHECK-OUT DISPLAY - TABULAR

4.3.1. Description

The MPS LH2 system tabular display will show MPS propellant valve open/close position indicators, power and RPC indicators (where applicable), and cockpit switch position indicators (where applicable) arranged in a tabular view by component per engineering instructions and guidelines. In addition to FD ON/OFF indicator blocks, the display shall have FD fail bit indication and engineering bypass indications for each FD.

4.3.2. Control Points

The display will have a push button target to launch the LH2 system enumerated state display as well as a display QUIT button to terminate the display.

4.3.3. Objects

There are no CLCS control objects associated with this status display.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

4.4. MPS LH2 SYSTEM TEST AND CHECK-OUT DISPLAY - ENUMERATED STATE

4.4.1. Description

The MPS LH2 system enumerated state display will show MPS propellant valve command state, indicator state, and over-all eneumerated state arranged in a tabular view by component per engineering instructions and guidelines.

4.4.2. Control Points

The display will have a push button target to launch the LH2 system tabular display as well as a display QUIT button to terminate the display.

4.4.3. Objects

There are no CLCS control objects associated with this status display.

4.5. MPS LO2 SYSTEM TEST AND CHECK-OUT DISPLAY - SCHEMATIC

4.5.1. Description

The MPS liquid oxygen (LO2) and gaseous oxygen (GO2) systems will be displayed in schematic format per engineering direction and guidelines (Ref V070-415113).

4.5.2. Control Points

4.5.2.1. LO2/GO2 Manifold Repress Regulator Control

The display will provide a target box for control (activation/securing) of the LO2 manifold repress regulator (PR5) and provide access to the methods of the MPS_RepressRegCntrl class.

4.5.2.2. Auto Valve Checkout

Auto valve checkout will perform a functional cycle test of all MPS LO2 orbiter cryogenic valves on the LO2 System Test and Check-Out Display (less PD1 disconnect). The auto checkout function will be available from the pull-down menu and will invoke the autoValveCheckout() method of the MPS_LO2_Subsystem Class.

4.5.2.3. Terminate Display

This option will allow the operator to terminate the LO2 System Test and Checkout Display. The option will be available from a target box in the lower left hand corner of the display.

4.5.3. Objects

4.5.3.1. Transducers/Level Sensors

The following transducers and ET point sensor measurements will be displayed in addition to the CLCS controlled cryogenic valves and system piping.

INSTANCE	NOMENCLATURE
MPS_MT49 (V41P1533C1)	LO2 Manifold Pressure
MPS_MT53 (V41P1590A1)	GO2 Disconnect Pressure

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

MPS_MT11 (V41P1171A1)	E1 GO2 Temperature
MPS_MT26 (V41T1271A1)	E2 GO2 Temperature
MPS_MT39 (V41T1371A1)	E3 GO2 temperature
MPS_MT512 (V41T1527A1)	LO2 Manifold Temperature B
MPS_MT48 (V41T1528A1)	LO2 Manifold Temperature A
MPS_MT6 (V41T1131C1)	E1 LO2 Feedline Temperature
MPS_MT19 (V41T1231C1)	E2 LO2 Feedline Temperature
MPS_MT32 (V41T1331C1)	E3 LO2 Feedline Temperature
MPS_MT5 (V41P1130C1)	LO2 E1 Feedline Pressure
MPS_MT18 (V41P1230C1)	LO2 E2 Feedline Pressure
MPS_MT31 (V41P1330C1)	LO2 E3 Feedline Pressure
E41H1062D	SSME 1 LO2 Bleed Valve Position PCT (SSME view)
E41H2062D	SSME 2 LO2 Bleed Valve Position PCT (SSME view)
E41H3062D	SSME 3 LO2 Bleed Valve Position PCT (SSME view)
GLOP2026A	LO2 Transfer Line Pressure (GSE) (VAB/PAD view)
GLOT2027A	LO2 Orbiter Inlet Temperature 1 (GSE) (VAB/PAD view)
GLOT3027A	LO2 Orbiter Inlet Temperature 2 (GSE) (VAB/PAD view)
GLOP2015A	LO2 ET Prepress Line Pressure (GSE) (VAB/PAD view)
T41P1750C1	ET LO2 Ullage Pressure No.1 (VAB/PAD view)
T41P1751C1	ET LO2 Ullage Pressure No.2 (VAB/PAD view)
T41P1752C1	ET LO2 Ullage Pressure No.3 (VAB/PAD view)
MPS_LO2_ECO1 (V41X1555X1)	LO2 Left ECO Sensor No. 1
MPS_LO2_ECO2 (V41X1556X1)	LO2 Left ECO Sensor No. 2
MPS_LO2_ECO3	LO2 Right ECO Sensor No. 2

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

(V41X1557X1)	
MPS_LO2_ECO4 (T41X1558X1)	LO2 Right ECO Sensor No. 1
T41T1755A1	ET LO2 Ullage Temperature (VAB/PAD view)
T41X1765E1	ET LO2 98% Sensor No. 1 (VAB/PAD view)
T41X1766E1	ET LO2 98% Sensor No. 2 (VAB/PAD view)
T41X1767E1	ET LO2 100% - Sensor (VAB/PAD view)
T41X1768E1	ET LO2 100% Sensor No. 1 (VAB/PAD view)
T41X1769E1	ET LO2 100% Sensor No. 2 (VAB/PAD view)
T41X1770E1	ET LO2 100% + Sensor (VAB/PAD view)
T41X1771E1	ET LO2 Overfill Sensor (VAB/PAD view)
T41X1774E1	ET LO2 Vent Valve Closed Indication
MPS_A82920 (GHEP2214A)	LO2 Feedline GSE Supply Pressure (GSE) (OPF view)
MPS_A82919 (GHEP2104A)	GO2 Prepress GSE Supply Pressure (GSE) (OPF view)

4.5.3.2. Static Components

4.5.3.3. CLCS Components

INSTANCE	NOMENCLATURE	METHODS
MPS_PV1	E1 LO2 Prevalve	open/close, energize/deenergize solenoids
MPS_PV2	E2 LO2 Prevalve	open/close, energize/deenergize solenoids
MPS_PV3	E3 LO2 Prevalve	open/close, energize/deenergize solenoids
MPS_PV7	LH2 Manifold Relief Isolation Valve	open/close, energize/deenergize solenoids
MPS_PV9	LH2 Outboard Fill and Drain Valve	open/close, energize/deenergize solenoids
MPS_PV10	LH2 Inboard Fill and Drain Valve	open/close, energize/deenergize solenoids
MPS_PV19	LO2 Bleed Valve	open/close, energize/deenergize solenoid
MPS_PV20	LO2 Pogo No. 1	open/close, energize/deenergize solenoid
MPS_PV21	LO2 Pogo No. 2	open/close, energize/deenergize solenoid
MPS_PD1	LO2 17" Disconnect	open/close, energize/deenergize solenoids

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

MPS_PD1_Latch	LO2 17" Disconnect	lock/unlock, energize/deenergize solenoids
MPS_LV40	LO2 Repress Sol 1	accessible thru MPS_RepressRegCntrl
MPS_LV41	LO2 Repress Sol 2	accessible thru MPS_RepressRegCntrl

4.6. MPS LO2 SYSTEM TEST AND CHECK-OUT DISPLAY - TABULAR

4.6.1. Description

The MPS LO2 system tabular display will show MPS propellant valve open/close position indicators, power and RPC indicators (where applicable), and cockpit switch position indicators (where applicable) arranged in a tabular view by component per engineering instructions and guidelines. In addition to FD ON/OFF indicator blocks, the display shall have FD fail bit indication and engineering bypass indications for each FD.

4.6.2. Control Points

The display will have a push button target to launch the LO2 system enumerated state display as well as a display QUIT button to terminate the display.

4.6.3. Objects

There are no CLCS control objects associated with this status display.

4.6.3.1. Static Components

4.6.3.2. CLCS Components

4.7. MPS LO2 SYSTEM TEST AND CHECK-OUT DISPLAY - ENUMERATED STATE

4.7.1. Description

The MPS LO2 system enumerated state display will show MPS propellant valve command state, indicator state, and over-all eneumerated state arranged in a tabular view by component per engineering instructions and guidelines.

4.7.2. Control Points

The display will have a push button target to launch the LO2 system tabular display as well as a display QUIT button to terminate the display.

4.7.3. Objects

There are no CLCS control objects associated with this status display.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

4.8. MPS GHE SYSTEM TEST AND CHECK-OUT DISPLAY - SCHEMATIC

4.8.1. Description

The MPS high pressure helium (GHe) supply and MPS/SSME 750 psi pneumatic systems will be displayed in schematic format per engineering direction and guidelines (Ref V070-415141 and V070-415770).

4.8.2. Control Points

4.8.2.1. Terminate Display

This option will allow the operator to terminate the LH2 System Test and Checkout Display. The option will be available from a target box in the lower left hand corner of the display.

4.8.3. Objects

4.9. MPS GHE SYSTEM TEST AND CHECK-OUT DISPLAY - TABULAR

4.9.1. Description

4.9.2. Control Points

4.9.3. Objects

4.10. MPS GHE SYSTEM TEST AND CHECK-OUT DISPLAY - ENUMERATED STATE

4.10.1. Description

The MPS GHe system enumerated state display will show MPS propellant valve command state, indicator state, and over-all eneumerated state (where applicable) arranged in a tabular view by component per engineering instructions and guidelines.

4.10.2. Control Points

The display will have a push button target to launch the GHe system tabular display as well as a display QUIT button to terminate the display.

4.10.3. Objects

There are no CLCS control objects associated with this status display.

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work

4.11. MPS SYSTEM MESSAGE DISPLAY

4.11.1. Description

A display will be generated to field MPS system software messaging. The type of messages to be sent to this display will be similar to that in LPS system page messaging. This display will be the default page for MPS/MPL/SSME system error messages. This display will field software system errors.

4.11.2. Control Points

There are no control points associated with this display other than a display terminate button.

4.11.3. Objects

There are no objects associated with this display.

5. INSTANCES

Printed documents may be obsolete. Check the CLCS Documentation Baseline web pages for current approved version of this document before using it for work