

```

using System;
using Microsoft.SPOT;

using Microsoft.SPOT.Hardware;
using Microsoft.SPOT.IO;

using System.Threading;
using System.Text;
using System.IO;
using System.IO.Ports;

using GHI.Processor;

using BreakContinue.Diagnostics;

//TODO List
//*** Add watchdog timer
//*** Add try/catch/finally resulting in recovery mode starting in every method (that needs it) and
    bubbling up to main()
//*** Check bellows position for leaks in Surface Ops
//*** Investigate maximum file size limitation for XModem file transfer whole
//*** Organize directories with correct version of VS2012 and .NETMF downloads
//** Change h value in PID loop when in CP mode
//** Check Extend retract logic at beginning of ascend state
//** Turn EA state into low power recovery mode state
//** Continue implementing the errorHandler
//** Add a deterministic way to check for correct response from SBE41 everywhere
//** Get rid of Thread.Sleep anywhere in the Main loop
//** Think about using bluetooth instead of USB for download/debug port
//** Deal with the case on entering ascend state where desired depth is deeper than current depth
//** log queue count
//** quit logging
//** come up with an in-program way to write unit tests
//** fix start cpMode try count to be absolute time based and start command mode timer on startCP mode
    failure
//** think about what big chunks of code might be better off as methods (like picking which ascend/descend
    band)
//* Create methods and classes for things that get repeated like
//    queuing a message
//    Error handler error counters
//    State exit and state timeout
//    Serial Instrument class with standard Serial port event handler and error handlers
//* Deal with S> pressure response
//* pre-Encode and make static UTF8 all constant strings and stringbuilders. Put them all in a text class
    . Maybe convert to byte array.
//* Log instrument handler calls
//* Investigate whether we really need a lock for methods in the same thread
//* Add serial port error received error handlers
//* Figure out a solid way to start and stop CTD, Buoyancy control, and everything else that needs stopping
    /starting
//* Change state timeout functionality to force state transitions even if SM hangs up, maybe use execution
    constraints
//* To get a line count: ctrl-shift-F find what: "^:b*[^:b#/>+.*$" (no quotes), Find options "use regular
    expressions", Look at these file types *.cs
//* Maybe discard serial buffers at end of every serial port event handler
//* Add execution constraints to first part of program (before state machine timeouts kick in)
//* Harmonize the way anything that puts stuff on the queue formats the record. Maybe every class that
    puts something on the queue should format the whole record in one of the class methods as opposed to
    letting the processQueue case statement do it.
//* Harmonize the way stuff is written to the engineering log
//* Get rid of Depth PID stuff and moving state and velocity state
//* Make all the serial port handlers look like the SBE41 handler and the optode EH
//* Add comments describing public variables and methods to each class
//* Clean up verbosity logic for engineering log.

namespace CPF

```

```

{
    public class Program
    {
        //Need to make sure string descriptions of states are properly assigned to correct value every time
        states are changed
        //see setStateNames() in CPFUtilities
        public enum CPFStates : int
        {
            NotRunning = 0,
            RunEngrTest,
            InitializeMission,
            InitializeProfile,
            SurfaceOpsSetBellows,
            PreMissionDelay,
            SurfaceOps,
            AutoballastSetBellows,
            AutoballastRetract,
            Descend,
            Anchor,
            StartCPMode,
            DumpCPData,
            DeAnchor,
            Ascend,
            Surface,
            EmergencyAscend,
            ProcessErrors,
            Exit,
            lastState
        }
        public static CPFStates CPFState;

        private enum EngrTests : int
        {
            none = 0,
            noOps,
            runBellows,
            cycleBellows,
            stepBellows,
            sendSBD,
            holdDepth,
            readOptode,
            getDirectory,
            exitProgram
        }
        private static EngrTests engrTest = EngrTests.none;

        //put all the state variables in a structure so it's easy to check the list
        public struct SV
        {
            public static bool timeSynched = false;
            public static bool telemetryDone = false;
            public static bool stateTimedout = false;
            public static bool stateEntry = true;
            public static bool surfaceOpsGo = false;
            public static bool runPVPID = false;
            public static bool uploadLastEngrFile = false;
            public static bool uploadEngrFile = false;
            public static bool SORecoveryMode = true;
            public static bool checkStuckPressure = false;
            public static bool anchored = false;
            public static bool parked = false;
            public static bool openNewFile = true;
        }

        //Static variables
        #region
    }
}

```

```

    public static I2CDevice g400I2C = new I2CDevice(null);

    public static StructQueue sQueue = new StructQueue();
    private static StructQueue.qStruct qStructure = new StructQueue.qStruct();

    private static bool missionRun = true;
    private static bool firstEngrTest = true;
    private static bool engrTestExtend = true;
    private static bool ddRX = false;
    private static bool uploadComplete = false;
    private static bool dumpCPData = false;
    private static bool CPDataDumped = false;
    private static bool anchorBellowsExtend = false;

    private static Timer CPFStateTimer;
    private static Timer ADCTimer;
    private static Timer PTHTimer;

    private static TimeSpan mtParkStartTime;
    private static TimeSpan lastParkSampleTime;
    private static TimeSpan mtTimeNow;
    private static TimeSpan parkTime;
    private static TimeSpan engrTestStartTime;
    private static TimeSpan engrTestCyclePeriod = new TimeSpan(0, 0, 30);

    private static Object programLock = new Object();

    private static StringBuilder sbTemp = new StringBuilder(256);
    public static StringBuilder[] sbStateNames = new StringBuilder[(int)CPFStates.lastState + 1];
    private static readonly StringBuilder sbStateEntry = new StringBuilder("State Entry");
    private static readonly StringBuilder sbStateExit = new StringBuilder("Normal state exit");
    private static readonly StringBuilder sbStateTimedout = new StringBuilder("State timed out, exiting
state");
    private static readonly StringBuilder emergencyAscendComment = new StringBuilder("Max Pressure
Detected, Emergency Ascend");
    private static readonly StringBuilder surfaceOpsWaitLine = new StringBuilder("Waiting for Console
Command\r\n");
    private static readonly StringBuilder lastFileName = new StringBuilder(128);
    private static readonly StringBuilder testFileName = new StringBuilder(128);
    private static readonly StringBuilder UDF = new StringBuilder("Uploading Data File name = ");
    private static readonly StringBuilder WFCC = new StringBuilder("Waiting for Console Command");
    private static readonly StringBuilder BBVError = new StringBuilder("Battery Bus Voltage below
threshold, beginning Emergency Ascend");
    private static readonly StringBuilder BPEError = new StringBuilder("Bellows position exceeds limits,
beginning Emergency Ascend");
    private static readonly StringBuilder EPP = new StringBuilder("Exiting Park Period");
    private static readonly StringBuilder toDeepError = new StringBuilder("Current depth exceeds max
pressure, beginning Emergency Ascend");
    private static readonly StringBuilder BottomDetect = new StringBuilder("Bottom detected based on
pressure variance value");
    private static readonly StringBuilder TimerNull = new StringBuilder("Surface Ops set state timer
null after first profile");
    private static readonly StringBuilder sbEASStateExit = new StringBuilder("Exiting Emergency Ascend
State");
    private static readonly StringBuilder pressureStuckError = new StringBuilder("Pressure isn't
changing, beginning Emergency Ascend");
    private static readonly StringBuilder exitAnchorTripped = new StringBuilder("Anchor state exit
threshold tripped");

    public static int pressureTableNum = -1;
    private static int profileNum = 0;
    private static int verbosityLevel = 5;
    private static int engrTestSpeed = 0;
    private static int ABRetractThresholdCount = 0;
    public static int signalQuality = -1;
    private static int anchorPressureCount = 0;

```

```

private static int dumpCPDataAttempts = 0;
private static int numTrys = 0;
private static int returnValue = 0;

private static double PVSetPoint = 0.0;
private static double depthSetPoint = 0.0;
private static double IB = configFile.IB;
private static double OB = configFile.OB;
private static double initialAnchorPressure = double.NaN;
private static double initialAnchorBP = double.NaN;

// names for each of the ascend and descend substates
private static StringBuilder sbScendNameNone = new StringBuilder("None");
private static StringBuilder sbScendNameAboveBand = new StringBuilder("Above Band");
private static StringBuilder sbScendNameUpperBand = new StringBuilder("Upper Band");
private static StringBuilder sbScendNameInBand = new StringBuilder("In Band");
private static StringBuilder sbScendNameLowerBand = new StringBuilder("Lower Band");
private static StringBuilder sbScendNameBelowBand = new StringBuilder("Below Band");
private static StringBuilder sbScendStateName = new StringBuilder();

private static int surfaceOpsSubstate = -1;
private static TimeSpan surfaceOpsSubstateStartTime;
private static TimeSpan surfaceOpsSubstate1Timeout = new TimeSpan(0, 0, 2);
private static TimeSpan surfaceOpsSubstate2Timeout = new TimeSpan(0, 0, 5);
private static TimeSpan surfaceOpsSubstate3Timeout = new TimeSpan(0, 0, 20);
private static TimeSpan surfaceOpsSubstate4Timeout = new TimeSpan(0, 0, 10);

private static int initMissionState = 0;
private static int startCPModeState = 0;
private static int dumpCPDataState = 0;

#endregion

public static void Main()
{
    #region
    Debug.Print("Program Started");
    Utility.SetLocalTime(RealTimeClock.GetDateTime());           //TODO Add synch to GPS
    CPFUtilities.setStateNames();
    BTConsole.OpenConsolePort();

    for (int i = 0; i < 10; i++)
        Debug.Print("Uncomment Bellows Position Check");

    CPFState = CPFStates.NotRunning;
    CPFStateTimer = new Timer(new TimerCallback(CPFStateTimerCallback), null, 1000000, 1000000);

    EngrLogger.init();

    sbTemp.Clear();
    sbTemp.Append("Intialize Elmo");
    EngrLogger.writeToColumns(sbTemp);
    Elmo.init();

    sbTemp.Clear();
    sbTemp.Append("Reading Mission Configuration File");
    EngrLogger.writeToColumns(sbTemp);
    MissionConfigFile.read();

    sbTemp.Clear();
    sbTemp.Append("Intializing Instrument Handler");
    EngrLogger.writeToColumns(sbTemp);
    InstrumentHandler.init();

    sbTemp.Clear();
    sbTemp.Append("Intialize SBE41 (takes up to 10 seconds)");

```

```

    EngrLogger.writeToColumns(sbTemp);
    SBE41.Init();
    processStructQ();

    sbTemp.Clear();
    sbTemp.Append("Intialize Optode");
    EngrLogger.writeToColumns(sbTemp);
    Optode.init();
    Thread.Sleep(2000);
    processStructQ();

    //sbTemp.Clear();
    //sbTemp.Append("Power On uBlox");
    //EngrLogger.writeComment(sbTemp.ToString());
    //uBlox.PowerOn("COM5", 19200);

    sbTemp.Clear();
    sbTemp.Append("Initialize EVK7 GPS");
    EngrLogger.writeToColumns(sbTemp);
    EVK7GPS.init();
    //for (int i = 0; i < 5; i++)
    //{
    //    EVK7GPS.readI2C();
    //    Thread.Sleep(5000);
    //}

    sbTemp.Clear();
    sbTemp.Append("Initialize A3LA Iridium Modem");
    EngrLogger.writeToColumns(sbTemp);
    IModem.init();

    //sbTemp.Clear();
    //sbTemp.Append("Power On Iridium");
    //EngrLogger.writeComment();
    //Iridium.PowerOn("COM2", 19200);

    //Process anything that is still in the queue
    Thread.Sleep(3000);
    processStructQ();

    sbTemp.Clear();
    sbTemp.Append("Intializing PTH and starting PTH and ADC Timers");
    EngrLogger.writeToColumns(sbTemp);
    PTH.initPTH();
    ADCTimer = new Timer(new TimerCallback(ADCTimerCallback), null, 0, configFile.ADCSamplePeriod);
    PTHTimer = new Timer(new TimerCallback(PTHTimerCallback), null, 0, configFile.PTHSamplePeriod);

    //Comment out all but one of these
    //engrTest = EngrTests.stepBellows;
    //engrTest = EngrTests.runBellows;
    //engrTest = EngrTests.cycleBellows;
    //engrTest = EngrTests.readOptode;
    //engrTest = EngrTests.sendSBD;
    engrTest = EngrTests.noOps;
    //engrTest = EngrTests.getDirectory;
    //engrTest = EngrTests.holdDepth;
    //engrTest = EngrTests.none;

    //Uncomment these 2 to run engineering tests
    //CPFState = CPFStates.RunEngrTest;
    //CPFStateTimer.Change(configFile.RunEngrTestTimeout, configFile.timeoutDefaultPeriod);

    //Uncoment these 2 lines to jump to SurfaceOps
    //CPFStateTimer.Change(configFile.surfaceOpsTimeout, configFile.timeoutDefaultPeriod);
    //CPFState = CPFStates.SurfaceOps;

```

```

//Uncomment these 2 lines to run the program normally
CPFStateTimer.Change(configFile.InitializeMissionTimeout, configFile.timeoutDefaultPeriod);
CPFState = CPFStates.InitializeMission;

SV.SORecoveryMode = configFile.RecoveryMode;

#endregion

while (missionRun)
{
    try
    {
        switch (CPFState)
        {
            case (CPFStates.RunEngrTest):
                #region

                if (SV.stateEntry)
                {
                    basicStateEntry();
                    firstEngrTest = true;
                }

                switch (engrTest)
                {
                    case (EngrTests.none):
                        Elmo.stopMotor();
                        SV.stateEntry = true;
                        SV.stateTimedout = false;
                        CPFState = CPFStates.InitializeProfile;
                        EngrLogger.writeToColumns(sbStateExit);
                        break;
                    case (EngrTests.noOps):
                        //don't do anything just monitor stuff
                        Elmo.stopMotor();
                        SV.runPVPID = false;
                        break;
                    case (EngrTests.sendSBD):
                        Elmo.stopMotor();
                        SV.runPVPID = false;

                        sbTemp.Clear();
                        sbTemp.Append("Getting GPS PUBX Message");
                        EngrLogger.writeToColumns(sbTemp);
                        EVK7GPS.getPUBX();

                        engrTest = EngrTests.noOps;
                        break;
                    case (EngrTests.readOptode):
                        Optode.doSample();
                        break;
                    case (EngrTests.getDirectory):
                        EngrLogger.getDirectory();
                        testFileName.Clear();
                        testFileName.Append(@"SD\2014-9-25T20-23-14");
                        BTConsole.uploadDataFile(testFileName);
                        engrTest = EngrTests.noOps;
                        break;
                    case (EngrTests.runBellows):
                        if (firstEngrTest)
                        {
                            engrTestSpeed = 20000;

                            //Comment out extend or retract
                            //Elmo.extendBellows(engrTestSpeed);
                            Elmo.retractBellows(engrTestSpeed);
                        }
                    }
                }
            }
        }
    }
}

```

```

        firstEngrTest = false;
    }
    break;
case (EngrTests.cycleBellows):
    if (firstEngrTest)
    {
        engrTestSpeed = 5000;
        engrTestStartTime = Microsoft.SPOT.Hardware.Utility.GetMachineTime() -
        ());

        engrTestExtend = false;
        Elmo.retractBellows(engrTestSpeed);
        Debug.Print("Cycle Bellows: Retract");
        firstEngrTest = false;
    }

    if ((Microsoft.SPOT.Hardware.Utility.GetMachineTime() -
    engrTestStartTime) >= engrTestCyclePeriod)
    {
        engrTestExtend = !engrTestExtend;
        if (engrTestExtend)
        {
            engrTestStartTime = Microsoft.SPOT.Hardware.Utility.
            GetMachineTime();

            Elmo.extendBellows(engrTestSpeed);
            Debug.Print("Cycle Bellows: Extend");
        }
        else
        {
            engrTestStartTime = Microsoft.SPOT.Hardware.Utility.
            GetMachineTime();

            Elmo.retractBellows(engrTestSpeed);
            Debug.Print("Cycle Bellows: Retract");
        }
    }
    break;
case (EngrTests.stepBellows):
    if (firstEngrTest)
    {
        engrTestSpeed = 20000;
        engrTestStartTime = Microsoft.SPOT.Hardware.Utility.GetMachineTime() -
        ());

        engrTestExtend = true;
        Elmo.extendBellows(engrTestSpeed);
        firstEngrTest = false;
    }

    if ((Microsoft.SPOT.Hardware.Utility.GetMachineTime() -
    engrTestStartTime) >= engrTestCyclePeriod)
    {
        if (Elmo.extending)
        {
            Elmo.stopMotor();
            engrTestStartTime = Microsoft.SPOT.Hardware.Utility.
            GetMachineTime();

        }
        else
        {
            Elmo.extendBellows(engrTestSpeed);
            engrTestStartTime = Microsoft.SPOT.Hardware.Utility.
            GetMachineTime();

        }
    }
    break;
case (EngrTests.holdDepth):
    break;
case (EngrTests.exitProgram):

```

```

        Elmo.stopMotor();
        SV.stateEntry = true;
        SV.stateTimedout = false;
        CPFState = CPFStates.Exit;
        break;
    }

    if (SV.stateTimedout)
    {
        Elmo.stopMotor();
        SV.stateEntry = true;
        SV.stateTimedout = false;
        EngrLogger.writeToColumns(sbStateTimedout);
        //lock (programLock) { Program.MessageQueue.Enqueue(UTF8Encoding.UTF8.
        GetBytes(EngrLogger.getPrefix(sbStateTimedout).ToString())); }
        CPFState = CPFStates.Exit;
    }
    break;
#endregion
case (CPFStates.InitializeMission):
#region
    if (SV.stateEntry)
    {
        basicStateEntry();

        sbConsoleCommand.Clear();
        sbConsoleCommand.Append("NONE");
    }

    //Sequence through mission initialization steps
    switch (initMissionState)
    {
        case 0:
            //Get SBE41 setting
            //Debug.Print("\r\n" + DateTime.Now.TimeOfDay.ToString() + " Sending ds
            command");

            if (SBE41.waitForResponse(SBE41.commands.ds) > 0)
                initMissionState = 1;
            break;
        case 1:
            //Get Optode settings
            //Debug.Print("\r\n" + DateTime.Now.TimeOfDay.ToString() + " Starting
            CTD Timer and getting Optode setup");
            SBE41.startCommandModeTimer();
            if (Optode.waitForResponse(Optode.commands.getAll) > 0)
                initMissionState = 2;
            break;
        case 2:
            //Wait for Optode to finish sending settings
            SV.stateEntry = true;
            EngrLogger.writeToColumns(sbStateExit);
            CPFState = CPFStates.InitializeProfile;
            SV.checkStuckPressure = true;
            break;
    }
    break;
#endregion
case (CPFStates.InitializeProfile):
#region
    if (SV.stateEntry)
    {
        basicStateEntry();

        profileNum = profileNum + 1;
        pressureTableNum = 0;
    }

```

```

        anchorPressureCount = 0;

        BTConsole.BTSend = true;
    }

    SV.stateEntry = true;
    EngrLogger.writeToColumns(sbStateExit);
    //lock (programLock) { Program.MessageQueue.Enqueue(UTF8Encoding.UTF8.GetBytes
(EngrLogger.getPrefix(sbStateTimedout).ToString())); }
    CPFStateTimer.Change(configFile.SOSetBellowsTimeout, configFile.
timeoutDefaultPeriod);
    CPFState = CPFStates.SurfaceOpsSetBellows;

    break;
    #endregion
case (CPFStates.SurfaceOpsSetBellows):
    #region
    if (SV.stateEntry)
    {
        basicStateEntry();
        BTConsole.BTSend = true;
    }

    //Decide what direction to go and keep checking
    if (bellowsPosition > (configFile.SOSetBellowsPosition + 0.5))
    {
        if (!Elmo.retracting)
        {
            Elmo.retractBellows(configFile.SOSetBellowsJV);
        }
    }
    else if (bellowsPosition < (configFile.SOSetBellowsPosition - 0.5))
    {
        if (!Elmo.extending)
        {
            Elmo.extendBellows(configFile.SOSetBellowsJV);
        }
    }
    else if ((bellowsPosition <= configFile.SOSetBellowsPosition + 0.5) &&
(bellowsPosition >= configFile.SOSetBellowsPosition - 0.5)) //
Normal state exit criteria
    {
        Elmo.stopMotor();

        SV.stateEntry = true;
        SV.stateTimedout = false;
        EngrLogger.writeToColumns(sbStateExit);
        if (profileNum == 1)
        {
            CPFStateTimer.Change(configFile.preMissionDelayTimeout, configFile.
timeoutDefaultPeriod);
            CPFState = CPFStates.PreMissionDelay;
        }
        else
        {
            CPFStateTimer.Change(configFile.dumpCPDataTimeout, configFile.
timeoutDefaultPeriod);
            CPFState = CPFStates.DumpCPData;
        }
    }

    if (SV.stateTimedout)
    {
        Elmo.stopMotor();
        SV.stateEntry = true;
        SV.stateTimedout = false;
    }

```

```

        EngrLogger.writeToColumns(sbStateTimeout);
        //lock (programLock) { Program.MessageQueue.Enqueue(UTF8Encoding.UTF8.
        GetBytes(EngrLogger.getPrefix(sbStateTimeout).ToString())); }
        CPFStateTimer.Change(configFile.surfaceOpsTimeout, configFile.
        timeoutDefaultPeriod);
        CPFState = CPFStates.SurfaceOps;
    }
    break;
    #endregion
case CPFStates.DumpCPData:
    #region
    if (SV.stateEntry)
    {
        basicStateEntry();

        SV.checkStuckPressure = false;
        ddRX = false;
        uploadComplete = false;
        CPDataDumped = false;
        dumpCPDataState = 0;
        dumpCPDataAttempts = 0;
    }

    if (dumpCPData)
    {
        switch (dumpCPDataState)
        {
            case 0:
                //First make sure the little darling is awake
                sbTemp.Clear();
                sbTemp.Append(DateTime.Now.TimeOfDay.ToString() + " Trying dd
command number: " + dumpCPDataAttempts.ToString());
                EngrLogger.writeToColumns(sbTemp);
                if (SBE41.WaitForResponse(SBE41.commands.cr1f) > 0)
                    dumpCPDataState = 1;
                break;
            case 1:
                //Now send the dd command
                sbTemp.Clear();
                sbTemp.Append(DateTime.Now.TimeOfDay.ToString() + " Trying dd
command number: " + dumpCPDataAttempts.ToString());
                EngrLogger.writeToColumns(sbTemp);
                //dumpCPDataAttempts = dumpCPDataAttempts + 1;
                //SBE41.dumpData();
                if (SBE41.WaitForResponse(SBE41.commands.dd) > 0)
                    dumpCPDataState = 1;
                break;
            case 2:
                //Wait until we get the "S>dd" response
                if (ddRX)
                {
                    sbTemp.Clear();
                    sbTemp.Append(DateTime.Now.TimeOfDay.ToString() + " Got dd
command response");
                    EngrLogger.writeToColumns(sbTemp);
                    dumpCPDataState = 2;
                }
                break;
            case 3:
                //now wait for the "upload complete" response
                if (uploadComplete)
                {
                    sbTemp.Clear();
                    sbTemp.Append(DateTime.Now.TimeOfDay.ToString() + " Got upload
complete");
                    EngrLogger.writeToColumns(sbTemp);
                }
            }
        }
    }
}

```

```

        dumpCPDataState = 3;
    }
    break;
case 4:
    //restart the CTD command timer
    SBE41.sendCRLF();
    Thread.Sleep(1000);
    SBE41.stopSBE41Timer = false;
    SBE41.restartTimer(500, configFile.SBE41SamplePeriod);
    CPDataDumped = true;
    break;
}
}

//Normal state exit
if (CPDataDumped || !dumpCPData)
{
    if (CPDataDumped)
    {
        sbTemp.Clear();
        sbTemp.Append("CP data dumped successfully");
        EngrLogger.writeToColumns(sbTemp);
    }
    if (!dumpCPData)
    {
        sbTemp.Clear();
        sbTemp.Append("No CP data to dump");
        EngrLogger.writeToColumns(sbTemp);
    }
    SV.stateEntry = true;
    SV.stateTimedout = false;
    SV.checkStuckPressure = true;
    ddRX = false;
    uploadComplete = false;
    CPDataDumped = false;
    dumpCPDataAttempts = 0;
    EngrLogger.writeToColumns(sbStateExit);
    CPFStateTimer.Change(configFile.surfaceOpsTimeout, configFile.
timeoutDefaultPeriod);
    CPFState = CPFStates.SurfaceOps;
}

if (SV.stateTimedout)
{
    Elmo.stopMotor();
    SV.stateEntry = true;
    SV.stateTimedout = false;
    SV.checkStuckPressure = true;
    ddRX = false;
    uploadComplete = false;
    CPDataDumped = false;
    dumpCPDataAttempts = 0;
    EngrLogger.writeToColumns(sbStateTimedout);
    CPFStateTimer.Change(configFile.surfaceOpsTimeout, configFile.
timeoutDefaultPeriod);
    CPFState = CPFStates.SurfaceOps;
}
break;
#endregion
case (CPFStates.PreMissionDelay):
    #region
    if (SV.stateEntry)
    {
        basicStateEntry();
    }
}

```

```

        if (SV.surfaceOpsGo)
        {
            SV.stateEntry = true;
            SV.stateTimedout = false;
            SV.surfaceOpsGo = false;
            BTConsole.BTSend = true;
            EngrLogger.writeToColumns(sbStateExit);
            CPFStateTimer.Change(configFile.surfaceOpsTimeout, configFile.
timeoutDefaultPeriod);
            CPFState = CPFStates.SurfaceOps;
        }

        if (SV.stateTimedout)
        {
            SV.stateEntry = true;
            SV.stateTimedout = false;
            SV.surfaceOpsGo = false;
            EngrLogger.writeToColumns(sbStateTimedout);
            //lock (programLock) { Program.MessageQueue.Enqueue(UTF8Encoding.UTF8.
GetBytes(EngrLogger.getPrefix(sbStateTimedout).ToString())); }
            CPFStateTimer.Change(configFile.surfaceOpsTimeout, configFile.
timeoutDefaultPeriod);
            CPFState = CPFStates.SurfaceOps;
        }
        break;
    #endregion
    case (CPFStates.SurfaceOps):
        #region
        if (SV.stateEntry)
        {
            basicStateEntry();

            SV.timeSynched = false;
            SV.telemetryDone = false;
            SV.surfaceOpsGo = false;
            BTConsole.BTSend = true;
            //Probably should send an error if surfaceOpsSubstate isn't already 0
            surfaceOpsSubstate = 0;
            signalQuality = -1;

            Optode.doSample();

            //After the first profile close the old engr log file and open a new one
            //unless we're in SORecoveryMode and have timed out at least once
            if ((profileNum > 1) && (SV.openNewFile))
            {
                lastFileName.Clear();
                lastFileName.Append(EngrLogger.fileName);

                sbTemp.Clear();
                sbTemp.Append("Closing old engr log file: ");
                sbTemp.Append(lastFileName);
                EngrLogger.writeToColumns(sbTemp);

                EngrLogger.closeFile();
                EngrLogger.openFile();

                sbTemp.Clear();
                sbTemp.Append("Opened new engr log file: ");
                sbTemp.Append(EngrLogger.fileName);
                EngrLogger.writeToColumns(sbTemp);

                directoryValid = false;

                SV.uploadLastEngrFile = true;
            }
        }
    #endregion
}

```

```

    }
    else
        SV.openNewFile = true;
}

//Sequence through steps to send SBD message and then upload last profile engr
log
switch (surfaceOpsSubstate)
{
    case 0:
        //Clear modem buffers
        signalQuality = -1;

        IModem.sendSBDD2();

        sbTemp.Clear();
        sbTemp.Append("Sent SBDD2");
        EngrLogger.writeToColumns(sbTemp);

        surfaceOpsSubstateStartTime = Microsoft.SPOT.Hardware.Utility.
GetMachineTime();

        surfaceOpsSubstate = 1;
        break;
    case 1:
        //Get PUBX message from GPS
        if ((Microsoft.SPOT.Hardware.Utility.GetMachineTime() -
surfaceOpsSubstateStartTime) > surfaceOpsSubstate1Timeout)
        {
            //EVK7GPS.getPUBX();
            EVK7GPS.readI2C();

            sbTemp.Clear();
            sbTemp.Append("Sent getPUBX");
            EngrLogger.writeToColumns(sbTemp);

            surfaceOpsSubstateStartTime = Microsoft.SPOT.Hardware.Utility.
GetMachineTime();

            surfaceOpsSubstate = 2;
        }
        break;
    case 2:
        //Get signal quality from IModem
        if ((Microsoft.SPOT.Hardware.Utility.GetMachineTime() -
surfaceOpsSubstateStartTime) > surfaceOpsSubstate2Timeout)
        {
            IModem.sendCSQ();

            sbTemp.Clear();
            sbTemp.Append("Sent CSQ");
            EngrLogger.writeToColumns(sbTemp);

            surfaceOpsSubstate = 3;
            surfaceOpsSubstateStartTime = Microsoft.SPOT.Hardware.Utility.
GetMachineTime();

        }
        break;
    case 3:
        //Load SBDWT buffer with PUBX message
        if ((Microsoft.SPOT.Hardware.Utility.GetMachineTime() -
surfaceOpsSubstateStartTime) > surfaceOpsSubstate3Timeout)
        {
            sbTemp.Clear();
            sbTemp.Append(timeNow.ToString("yyyy-MM-dd"));
            sbTemp.Append("T");
            sbTemp.Append(timeNow.ToString("HH:mm:ss"));
            sbTemp.Append("Z,");

```

```

        sbTemp.Append("SQ = ");
        sbTemp.Append(signalQuality.ToString());
        sbTemp.Append(",");
        sbTemp.Append(UTF8Encoding.UTF8.GetChars(pubxByteArray));
        Debug.Print(DateTime.Now + "SBDWT message = " + sbTemp.ToString());

        IModem.sendSBDWT(UTF8Encoding.UTF8.GetBytes(sbTemp.ToString()));

        sbTemp.Clear();
        sbTemp.Append("Sent SBDWT");
        EngrLogger.writeToColumns(sbTemp);

        surfaceOpsSubstate = 4;
        surfaceOpsSubstateStartTime = Microsoft.SPOT.Hardware.Utility.
GetMachineTime();
    }
    break;
case 4:
    //Send the message with SBDDI
    if ((Microsoft.SPOT.Hardware.Utility.GetMachineTime() -
surfaceOpsSubstateStartTime) > surfaceOpsSubstate4Timeout)
    {
        IModem.sendSBDI();

        sbTemp.Clear();
        sbTemp.Append("Sent SBDDI");
        EngrLogger.writeToColumns(sbTemp);

        surfaceOpsSubstate = -1;
        signalQuality = -1;
    }
    break;
default:
    break;
}

//Upload engineering log file here
if ((SV.uploadLastEngrFile || SV.uploadEngrFile) && (surfaceOpsSubstate == -1))
{
    if (SV.uploadLastEngrFile)
    {
        sbTemp.Clear();
        sbTemp.Append(UDF);
        sbTemp.Append(lastFileName);
        EngrLogger.writeToColumns(sbTemp);
        BTConsole.uploadDataFile(lastFileName);
        //TODO The BTConsole
        might need to go in a separate thread
        SV.uploadLastEngrFile = false;
    }
    else
    {
        if (directoryValid)
        {
            switch (uploadEngrFileNum)
            {
                #region
                case 0:
                    BTConsole.uploadDataFile(EngrLogger.sbFileNames[0]);
                    break;
                case 1:
                    BTConsole.uploadDataFile(EngrLogger.sbFileNames[1]);
                    break;
                case 2:
                    BTConsole.uploadDataFile(EngrLogger.sbFileNames[2]);
                    break;
                case 3:

```

```

        BTConsole.uploadDataFile(EngrLogger.sbFileNames[3]);
        break;
    case 4:
        BTConsole.uploadDataFile(EngrLogger.sbFileNames[4]);
        break;
    case 5:
        BTConsole.uploadDataFile(EngrLogger.sbFileNames[5]);
        break;
    case 6:
        BTConsole.uploadDataFile(EngrLogger.sbFileNames[6]);
        break;
    case 7:
        BTConsole.uploadDataFile(EngrLogger.sbFileNames[7]);
        break;
    case 8:
        BTConsole.uploadDataFile(EngrLogger.sbFileNames[8]);
        break;
    case 9:
        BTConsole.uploadDataFile(EngrLogger.sbFileNames[9]);
        break;
    #endregion
}
SV.uploadEngrFile = false;
}
else
{
    sbTemp.Clear();
    sbTemp.Append("Need a valid directory, execute $GETDIR command");
    BTConsole.SendLine(sbTemp);
    SV.uploadEngrFile = false;
}
}
}

//Normal exit in SORecovery Mode only if surfaceOpsGo = true
//Normal exit if not in SORecovery mode when (timeSynched AND SV.telemetryDone)
OR surfaceOPSgo = true
//No matter what, wait for telemetry to finish sequence
if (surfaceOpsSubstate == -1)
{
    if ((SV.SORecoveryMode && SV.surfaceOpsGo) || (!SV.SORecoveryMode && (SV.
timeSynched && SV.telemetryDone) || SV.surfaceOpsGo)))
    {
        SV.timeSynched = false;
        SV.telemetryDone = false;
        SV.surfaceOpsGo = false;
        SV.stateEntry = true;
        SV.stateTimeout = false;
        SV.openNewFile = true;

        EngrLogger.writeToColumns(sbStateExit);
        CPFStateTimer.Change(configFile.ABSetBellowsTimeout, configFile.
timeoutDefaultPeriod);

        CPFState = CPFStates.AutoballastSetBellows;

        sbTemp.Clear();
        sbTemp.Append("Free Memory = ");
        sbTemp.Append(Debug.GC(false).ToString());
        EngrLogger.writeToColumns(sbTemp);
    }

    if (SV.stateTimeout)
    {
        if (SV.SORecoveryMode)
        {
            SV.stateEntry = true;

```

```

        SV.stateTimeout = false;
        SV.surfaceOpsGo = false;
        SV.openNewFile = false;
        EngrLogger.writeToColumns(sbStateTimeout);
        CPFStateTimer.Change(configFile.surfaceOpsTimeout, configFile.
timeoutDefaultPeriod);

        CPFState = CPFStates.SurfaceOps;

        sbTemp.Clear();
        sbTemp.Append("Free Memory = ");
        sbTemp.Append(Debug.GC(false).ToString());
        EngrLogger.writeToColumns(sbTemp);
    }
    else
    {
        SV.timeSynched = false;
        SV.telemetryDone = false;
        SV.surfaceOpsGo = false;
        SV.stateEntry = true;
        SV.stateTimeout = false;
        SV.openNewFile = true;
        EngrLogger.writeToColumns(sbStateTimeout);
        CPFStateTimer.Change(configFile.ABSetBellowsTimeout, configFile.
timeoutDefaultPeriod);

        CPFState = CPFStates.AutoballastSetBellows;

        sbTemp.Clear();
        sbTemp.Append("Free Memory = ");
        sbTemp.Append(Debug.GC(false).ToString());
        EngrLogger.writeToColumns(sbTemp);
    }
}
}
break;
#endregion
case (CPFStates.AutoballastSetBellows):
#region
if (SV.stateEntry)
{
    basicStateEntry();
}

//Decide what direction to go and keep checking
if (bellowsPosition > (configFile.ABSetBellowsPosition + 0.5))
{
    if (!Elmo.retracting)
    {
        Elmo.retractBellows(configFile.ABSetBellowsJV);
    }
}
else if (bellowsPosition < (configFile.ABSetBellowsPosition - 0.5))
{
    if (!Elmo.extending)
    {
        Elmo.extendBellows(configFile.ABSetBellowsJV);
    }
}
else //Normal state exit criteria
{
    Elmo.stopMotor();
    SV.stateEntry = true;
    SV.stateTimeout = false;
    EngrLogger.writeToColumns(sbStateExit);
    CPFStateTimer.Change(configFile.ABRetractTimeout, configFile.
timeoutDefaultPeriod);

    CPFState = CPFStates.AutoballastRetract;

```

```

    }

    if (SV.stateTimedout)
    {
        Elmo.stopMotor();
        SV.stateEntry = true;
        SV.stateTimedout = false;
        EngrLogger.writeToColumns(sbStateTimedout);
        //lock (programLock) { Program.MessageQueue.Enqueue(UTF8Encoding.UTF8.
GetBytes(EngrLogger.getPrefix(sbStateTimedout).ToString())); }
        CPFStateTimer.Change(configFile.ABRetractTimeout, configFile.
timeoutDefaultPeriod);
        CPFState = CPFStates.AutoballastRetract;
    }
    break;
#endregion
case (CPFStates.AutoballastRetract):
#region
    if (SV.stateEntry)
    {
        basicStateEntry();

        ABRetractThresholdCount = 0;
        Elmo.retractBellows(configFile.ABRetractJV);
    }

    //Increment threshold counter on successive tests otherwise reset counter to 0
    if (pressure >= configFile.ABDepthThreshold)
        ABRetractThresholdCount = ABRetractThresholdCount + 1;
    else
        ABRetractThresholdCount = 0;

    //Normal state exit criteria
    if (ABRetractThresholdCount >= 3)
    {
        Elmo.stopMotor();
        SV.stateEntry = true;
        EngrLogger.writeToColumns(sbStateExit);
        //CPFStateTimer.Change(configFile.RunEngrTestTimeout, configFile.
timeoutDefaultPeriod);
        //CPFState = CPFStates.RunEngrTest;
        CPFStateTimer.Change(configFile.descendTimeout, configFile.
timeoutDefaultPeriod);
        CPFState = CPFStates.Descend;
        SV.stateTimedout = false;
    }

    if (SV.stateTimedout)
    {
        Elmo.stopMotor();
        SV.stateEntry = true;
        SV.stateTimedout = false;
        EngrLogger.writeToColumns(sbStateTimedout);
        //lock (programLock) { Program.MessageQueue.Enqueue(UTF8Encoding.UTF8.
GetBytes(EngrLogger.getPrefix(sbStateTimedout).ToString())); }
        CPFStateTimer.Change(configFile.descendTimeout, configFile.
timeoutDefaultPeriod);
        CPFState = CPFStates.Descend;
    }
    break;
#endregion
case (CPFStates.Descend):
#region
    if (SV.stateEntry)
    {

```

```

        basicStateEntry();
        SV.parked = false;
        pressureTableNum = 0;
        BuoyancyControl.resetPVPID();
        SV.runPVPID = true;
        //GM Moved these 2 lines from if block directly below to here. Seems like ✓
they only need to run once on state entry
        IB = configFile.IB;
        OB = configFile.OB;
        sbScendStateName.Clear();
        sbScendStateName.Append(sbScendNameNone);
        BTConsole.BTSend = false;
    }

    depthSetPoint = MissionConfigFile.descendTable[pressureTableNum].depth;
    parkTime = MissionConfigFile.descendTable[pressureTableNum].parkTime;

    //Not parking at target depth just taking a sample
    if (parkTime == TimeSpan.Zero)
    {
        if (pressure < depthSetPoint)
        {
            PVSetPoint = -1.0 * System.Math.Abs(MissionConfigFile.descendTable
[pressureTableNum].PV); ✓
            SV.runPVPID = true;
        }

        if (pressure >= depthSetPoint)
        {
            InstrumentHandler.sampleInstruments(pressureTableNum);
            pressureTableNum = pressureTableNum + 1;
        }
    }
    //we are parking at this depth check to see which of the 5 depth bands we're ✓
in
    else if (parkTime > TimeSpan.Zero)
    {
        //Above outer band
        if (pressure <= (depthSetPoint - OB))
        {
            PVSetPoint = -1.0 * System.Math.Abs(MissionConfigFile.descendTable
[pressureTableNum].PV); ✓
            SV.runPVPID = true;
            sbScendStateName.Clear();
            sbScendStateName.Append(sbScendNameAboveBand);
        }
        //In upper outer band
        else if ((pressure > (depthSetPoint - OB)) && (pressure < (depthSetPoint -
IB))) ✓

        {
            PVSetPoint = -1 * System.Math.Abs(configFile.approachVel);
            SV.runPVPID = true;
            sbScendStateName.Clear();
            sbScendStateName.Append(sbScendNameUpperBand);
        }
        //In inner band
        else if ((pressure >= (depthSetPoint - IB)) && (pressure <= (depthSetPoint
+ IB))) ✓

        {
            PVSetPoint = 0.0;
            SV.runPVPID = true;
            sbScendStateName.Clear();
            sbScendStateName.Append(sbScendNameInBand);

            //set up the sample and park time start times and take a sample
            //the first time we are in-band

```

```

        if (!SV.parked)
        {
            //used as reference for exiting park mode
            mtParkStartTime = Microsoft.SPOT.Hardware.Utility.GetMachineTime();

            //used as reference for when to sample during park mode
            lastParkSampleTime = mtParkStartTime;

            SV.parked = true;

            sbTemp.Clear();
            sbTemp.Append("Starting park period = ");
            sbTemp.Append(parkTime.ToString());
            EngrLogger.writeToColumns(sbTemp);

            InstrumentHandler.sampleInstruments(pressureTableNum);
        }
    }
    //in lower outer band
    else if ((pressure > (depthSetPoint + IB)) && (pressure < (depthSetPoint + OB)))
    {
        PVSetPoint = System.Math.Abs(configFile.approachVel);
        SV.runPVPID = true;
        sbScendStateName.Clear();
        sbScendStateName.Append(sbScendNameLowerBand);
    }
    //Below lower band
    else if (pressure >= depthSetPoint + OB)
    {
        PVSetPoint = 1.0 * System.Math.Abs(MissionConfigFile.descendTable
        SV.runPVPID = true;
        sbScendStateName.Clear();
        sbScendStateName.Append(sbScendNameBelowBand);
    }
}

//check if it's time to sample or if it is time to exit park mode
if (SV.parked)
#region
{
    mtTimeNow = Microsoft.SPOT.Hardware.Utility.GetMachineTime();

    //check to see if we need to sample
    if ((mtTimeNow - lastParkSampleTime) >= MissionConfigFile.parkSamplePeriod)
    {
        lastParkSampleTime = mtTimeNow;
        InstrumentHandler.sampleInstruments(pressureTableNum);
    }

    //check to see if park time has expired
    if ((mtTimeNow - mtParkStartTime) >= parkTime)
    {
        pressureTableNum = pressureTableNum + 1;
        EngrLogger.writeToColumns(EPP);
        SV.parked = false;
    }
}
#endregion

//Check for anchor condition as long as we haven't hit the inner band i.e. we
're parked
#region
if (!SV.parked)
{

```

```

        //Don't check for anchoring when really shallow
        if (pressure > configFile.anchorPressureMinimum)
        {
            if (lpfResults.pressureVariance < configFile.
anchorPressureVarianceLimit)
            {
                anchorPressureCount = anchorPressureCount + 1;
                if (anchorPressureCount >= configFile.anchorPressureCountThreshold)
                {
                    EngrLogger.writeToColumns(BottomDetect);
                    Elmo.stopMotor();
                    SV.runPVPID = false;
                    SV.stateEntry = true;
                    anchorPressureCount = 0;
                    //Anchor mode doesn't have it's own state timeout is uses the
descend state timeout
                    CPFState = CPFStates.Anchor;
                }
            }
            //Reset the counter if the threshold true results aren't sequential
            else
            {
                if (anchorPressureCount > 0)
                    anchorPressureCount = 0;
            }
        }
    }
}
#endregion

//Check to see if we're near the bellows limits
if ((bellowsPosition < configFile.lowerDescendBellowsLimit) || (bellowsPosition >
> configFile.upperDescendBellowsLimit))
{
    Elmo.stopMotor();
    SV.runPVPID = false;
    SV.stateEntry = true;
    sbTemp.Clear();
    sbTemp.Append("Hit descend bellows limit, transition to anchor state");
    EngrLogger.writeToColumns(sbTemp);
    //Anchor mode doesn't have it's own state timeout is uses the descend state
timeout
    CPFState = CPFStates.Anchor;
}

//Check to see if we're at the end of the descend table
//Normal state exit
if (pressureTableNum >= MissionConfigFile.descendTable.Length)
{
    Elmo.stopMotor();
    SV.runPVPID = false;
    SV.stateEntry = true;
    EngrLogger.writeToColumns(sbStateExit);
    CPFStateTimer.Change(configFile.startCPModeTimeout, configFile.
timeoutDefaultPeriod);
    CPFState = CPFStates.StartCPMode;
    sbScendStateName.Clear();
    SV.stateTimedout = false;
}

if (SV.stateTimedout)
{
    PVSetPoint = 0.0;
    SV.runPVPID = false;
    SV.stateEntry = true;
    SV.stateTimedout = false;
    EngrLogger.writeToColumns(sbStateTimedout);
}

```

```

        //lock (programLock) { Program.MessageQueue.Enqueue(UTF8Encoding.UTF8.
GetBytes(EngrLogger.getPrefix(sbStateTimedout).ToString())); }
        CPFStateTimer.Change(configFile.startCPModeTimeout, configFile.
timeoutDefaultPeriod);
        sbScendStateName.Clear();
        CPFState = CPFStates.StartCPMode;
    }
    break;
    #endregion
    case (CPFStates.Ascend):
        #region
        if (SV.stateEntry)
        {
            basicStateEntry();
            SV.parked = false;

            // Find the first depth in the ascend table less than current depth
            // Eventually need to deal with the case where the first ascend station is
deeper than last descend station
            // and we aren't anchored
            pressureTableNum = 0;
            for (int i = 0; i < MissionConfigFile.ascendTable.Length; i++)
            {
                if (pressure < MissionConfigFile.ascendTable[pressureTableNum].depth)
                {
                    sbTemp.Clear();
                    sbTemp.Append("Skipping ascend table depth = ");
                    sbTemp.Append(MissionConfigFile.ascendTable[pressureTableNum].depth
.ToString());

                    EngrLogger.writeToColumns(sbTemp);
                    pressureTableNum = pressureTableNum + 1;
                }
            }

            IB = configFile.IB;
            OB = configFile.OB;
            BuoyancyControl.resetPVPID();
            PVSetPoint = 0.0;
            SV.runPVPID = false;
        }

        depthSetPoint = MissionConfigFile.ascendTable[pressureTableNum].depth;
        parkTime = MissionConfigFile.ascendTable[pressureTableNum].parkTime;

        if (parkTime == TimeSpan.Zero)
        {
            if (pressure > depthSetPoint)
            {
                PVSetPoint = System.Math.Abs(MissionConfigFile.ascendTable
[pressureTableNum].PV);
                SV.runPVPID = true;
            }

            if (pressure <= depthSetPoint)
            {
                InstrumentHandler.sampleInstruments(pressureTableNum);
                pressureTableNum = pressureTableNum + 1;
            }
        }
        //we are parking at this depth check to see which of the 5 depth bands we're
in
        else if (parkTime > TimeSpan.Zero)
        {
            //Above band
            if (pressure <= (depthSetPoint - OB))
            {

```

```

        PVSetPoint = -1.0 * System.Math.Abs(MissionConfigFile.ascendTable
[pressureTableNum].PV);
        SV.runPVPID = true;
        sbScendStateName.Clear();
        sbScendStateName.Append(sbScendNameAboveBand);
    }
    //In upper outer band
    else if ((pressure > (depthSetPoint - OB)) && (pressure < (depthSetPoint -
IB)))
    {
        PVSetPoint = -1 * System.Math.Abs(configFile.approachVel);
        SV.runPVPID = true;
        sbScendStateName.Clear();
        sbScendStateName.Append(sbScendNameUpperBand);
    }
    //In inner band
    else if ((pressure >= (depthSetPoint - IB)) && (pressure <= (depthSetPoint
+ IB)))
    {
        PVSetPoint = 0.0;
        SV.runPVPID = true;
        sbScendStateName.Clear();
        sbScendStateName.Append(sbScendNameInBand);

        //set up the sample and park time start times and take a sample
        if (!SV.parked)
        {
            //used as reference for exiting park mode
            mtParkStartTime = Microsoft.SPOT.Hardware.Utility.GetMachineTime();

            //used as reference for when to sample during park mode
            lastParkSampleTime = mtParkStartTime;

            SV.parked = true;

            sbTemp.Clear();
            sbTemp.Append("Starting park period = ");
            sbTemp.Append(parkTime.ToString());
            EngrLogger.writeToColumns(sbTemp);

            InstrumentHandler.sampleInstruments(pressureTableNum);
        }
    }
    //in lower outer band
    else if ((pressure > (depthSetPoint + IB)) && (pressure < (depthSetPoint +
OB)))
    {
        PVSetPoint = System.Math.Abs(configFile.approachVel);
        SV.runPVPID = true;
        sbScendStateName.Clear();
        sbScendStateName.Append(sbScendNameLowerBand);
    }
    //Below band
    else if (pressure >= depthSetPoint + OB)
    {
        PVSetPoint = 1.0 * System.Math.Abs(MissionConfigFile.ascendTable
[pressureTableNum].PV);
        SV.runPVPID = true;
        sbScendStateName.Clear();
        sbScendStateName.Append(sbScendNameBelowBand);
    }
}

//check if it's time to sample or if it is time to exit park mode
if (SV.parked)
#region

```

```

    {
        mtTimeNow = Microsoft.SPOT.Hardware.Utility.GetMachineTime();

        //check to see if we need to sample
        if ((mtTimeNow - lastParkSampleTime) >= MissionConfigFile.parkSamplePeriod)
        {
            lastParkSampleTime = mtTimeNow;
            InstrumentHandler.sampleInstruments(pressureTableNum);
        }

        //check to see if park time has expired
        if ((mtTimeNow - mtParkStartTime) >= parkTime)
        {
            pressureTableNum = pressureTableNum + 1;
            EngrLogger.writeToColumns(EPP);
            SV.parked = false;
        }
    }
    #endregion

    //check to see if we've hit the surface
    if (pressure < (configFile.cpPressureCutoff + 0.5))
    {
        BTConsole.BTSend = true;
        Elmo.stopMotor();
        if (configFile.CPMode)
        {
            sbTemp.Clear();
            sbTemp.Append("Stopping CP mode");
            //sbTemp.Append("Stopping CP mode and dumping data");
            EngrLogger.writeToColumns(sbTemp);
            SBE41.stopProfile();
            dumpCPData = true;
            //Thread.Sleep(5000);
            //SBE41.dumpData();
            //Thread.Sleep(10000);
            //SBE41.sendCRLF();
            //Thread.Sleep(1000);
            //SBE41.stopSBE41Timer = false;
            //SBE41.restartTimer(500, configFile.SBE41SamplePeriod);
        }
        SV.runPVPID = false;
        SV.stateEntry = true;
        sbScendStateName.Clear();
        EngrLogger.writeToColumns(sbStateExit);
        CPFState = CPFStates.InitializeProfile;
        SV.stateTimedout = false;
    }

    if (SV.stateTimedout)
    {
        Elmo.stopMotor();
        if (configFile.CPMode)
        {
            SBE41.stopProfile();
            Thread.Sleep(1000);
            SBE41.dumpData();
            Thread.Sleep(5000);
            SBE41.sendCRLF();
            Thread.Sleep(1000);
            SBE41.stopSBE41Timer = false;
            SBE41.restartTimer(500, configFile.SBE41SamplePeriod);
        }
        SV.runPVPID = false;
        SV.stateEntry = true;
        SV.stateTimedout = false;
    }

```

```

        sbScendStateName.Clear();
        EngrLogger.writeToColumns(sbStateTimedout);
        //lock (programLock) { Program.MessageQueue.Enqueue(UTF8Encoding.UTF8.
GetBytes(EngrLogger.getPrefix(sbStateTimedout).ToString())); }
        CPFState = CPFStates.InitializeProfile;
    }
    break;
    #endregion
case (CPFStates.Anchor):
    #region
    if (SV.stateEntry)
    {
        basicStateEntry();
        SV.anchored = true;
        initialAnchorPressure = pressure;
        initialAnchorBP = bellowsPosition;
        anchorBellowsExtend = false;
        sbTemp.Clear();
        sbTemp.Append("Starting anchor period = ");
        sbTemp.Append(parkTime.ToString());
        EngrLogger.writeToColumns(sbTemp);
        if (!SV.parked)
        {
            //used as reference for exiting park mode
            mtParkStartTime = Microsoft.SPOT.Hardware.Utility.GetMachineTime();

            //used as reference for when to sample during park mode
            lastParkSampleTime = mtParkStartTime;

            SV.parked = true;

            sbTemp.Clear();
            sbTemp.Append("Starting park period = ");
            sbTemp.Append(parkTime.ToString());
            EngrLogger.writeToColumns(sbTemp);

            InstrumentHandler.sampleInstruments(pressureTableNum);
        }
    }

    mtTimeNow = Microsoft.SPOT.Hardware.Utility.GetMachineTime();

    //check to see if we need to sample
    if ((mtTimeNow - lastParkSampleTime) >= MissionConfigFile.parkSamplePeriod)
    {
        lastParkSampleTime = mtTimeNow;
        InstrumentHandler.sampleInstruments(pressureTableNum);
    }

    //check to see if park time has expired
    if ((mtTimeNow - mtParkStartTime) >= parkTime)
    {
        SV.stateEntry = true;
        SV.stateTimedout = false;
        EngrLogger.writeToColumns(sbStateExit);
        //Leave SV.anchored true; startCPMode needs to know
        CPFStateTimer.Change(configFile.startCPModeTimeout, configFile.
timeoutDefaultPeriod);
        CPFState = CPFStates.StartCPMode;
    }

    //Check to see if we aren't really parked, if not go back to descend as if
nothing had happened
    //This may need more logic if there is more than one descend entry in the
descend table
    if ((pressure - initialAnchorPressure) >= configFile.

```

```

exitAnchorPressureThreshold)
    {
        //SV.state entry needs to be false to make sure we re-enter descend state
as if we haven't left
        SV.stateEntry = false;
        SV.stateTimedout = false;
        SV.anchored = false;
        EngrLogger.writeToColumns(exitAnchorTripped);
        CPFStateTimer.Change(configFile.descendTimeout, configFile.
timeoutDefaultPeriod);
        CPFState = CPFStates.Descend;
    }

//Check to see if the bellows have retracted more than the limit due to oil
leakage
//Start extending if it has
if (bellowsPosition < (initialAnchorBP - 1.0))
{
    Elmo.extendBellows(2000);
    anchorBellowsExtend = true;
    sbTemp.Clear();
    sbTemp.Append("Extending bellows back to initial anchor position");
    EngrLogger.writeToColumns(sbTemp);
}
if (anchorBellowsExtend)
{
    if (bellowsPosition > (initialAnchorBP - 0.3))
    {
        Elmo.stopMotor();
        anchorBellowsExtend = false;
        sbTemp.Clear();
        sbTemp.Append("Stop extending bellows");
        EngrLogger.writeToColumns(sbTemp);
    }
}

if (SV.stateTimedout)
{
    SV.stateEntry = true;
    SV.stateTimedout = false;
    EngrLogger.writeToColumns(sbStateTimedout);
    //Leave SV.anchored true; startCPMode needs to know
    CPFStateTimer.Change(configFile.startCPModeTimeout, configFile.
timeoutDefaultPeriod);
    CPFState = CPFStates.StartCPMode;
}
break;
#endregion
case (CPFStates.StartCPMode):
    #region
    if (SV.stateEntry)
    {
        basicStateEntry();
        SV.runPVPID = false;
        startCPModeState = 0;
        numTrys = 0;
    }

    //Start CP Mode
    switch (startCPModeState)
    {
        case 0:
            //Stop SBE41 timer
            sbTemp.Clear();
            sbTemp.Append("Stopping SBE41 Timer");
            EngrLogger.writeToColumns(sbTemp);

```

```

        SBE41.stopTimer();
        //startCPModeStartTime = Microsoft.SPOT.Hardware.Utility.GetMachineTime ✓
    ());

    startCPModeState = 1;
    numTrys = 0;
    break;
case 1:
    //Wait for SBE41 Timer to Stop then Wakeup the SBE41
    if (numTrys < 10)
    {
        if (SBE41.SBE41TimerStopped)
        {
            sbTemp.Clear();
            sbTemp.Append("SBE41 Timer Stopped, Sending SBE41 CR/LF");
            EngrLogger.writeToColumns(sbTemp);
            startCPModeState = 2;
        }
        else
            numTrys = numTrys + 1;
    }
    else
    {
        sbTemp.Clear();
        sbTemp.Append("Exceed number for SBE41 Timer Stopped tryes, exiting ✓

startCPMode");

        EngrLogger.writeToColumns(sbTemp);
        startCPModeState = 4;
        numTrys = 0;
        //probably should start command mode timer
    }
    break;
case 2:
    returnValue = SBE41.waitForResponse(SBE41.commands.crlf);
    if (returnValue > 0)
    {
        sbTemp.Clear();
        sbTemp.Append("Got CR/LF response, sending startprofile");
        EngrLogger.writeToColumns(sbTemp);
        startCPModeState = 3;
    }
    else if (returnValue < 0)
    {
        sbTemp.Clear();
        sbTemp.Append("Didn't get CR/LF response, exiting startCPMode");
        EngrLogger.writeToColumns(sbTemp);
        startCPModeState = 4;
    }
    break;
case 3:
    //Start CP mode
    returnValue = SBE41.waitForResponse(SBE41.commands.startProfile);
    if (returnValue > 0)
    {
        sbTemp.Clear();
        sbTemp.Append("Got startprofile response");
        EngrLogger.writeToColumns(sbTemp);
        startCPModeState = 4;
    }
    else if (returnValue < 0)
    {
        sbTemp.Clear();
        sbTemp.Append("Didn't get startprofile response, exiting ✓

startCPMode");

        EngrLogger.writeToColumns(sbTemp);
        startCPModeState = 4;
    }
}

```

```

        break;
    case 4:
        //CP Mode should be started, transition to next state
        sbTemp.Clear();
        sbTemp.Append("transitioning to next state");
        EngrLogger.writeToColumns(sbTemp);

        Elmo.stopMotor();
        SV.runPVPID = false;
        SV.stateEntry = true;
        sbScendStateName.Clear();
        SV.stateTimedout = false;

        if (SV.anchored)
        {
            CPFStateTimer.Change(configFile.deAnchorTimeout, configFile.
timeoutDefaultPeriod);
            CPFState = CPFStates.DeAnchor;
        }
        else
        {
            CPFStateTimer.Change(configFile.ascendTimeout, configFile.
timeoutDefaultPeriod);
            CPFState = CPFStates.Ascend;
        }

        EngrLogger.writeToColumns(sbStateExit);
        break;
    }

    if (SV.stateTimedout)
    {
        PVSetPoint = 0.0;
        SV.runPVPID = false;
        SV.stateEntry = true;
        SV.stateTimedout = false;
        sbScendStateName.Clear();

        if (SV.anchored)
        {
            CPFStateTimer.Change(configFile.deAnchorTimeout, configFile.
timeoutDefaultPeriod);
            CPFState = CPFStates.DeAnchor;
        }
        else
        {
            CPFStateTimer.Change(configFile.ascendTimeout, configFile.
timeoutDefaultPeriod);
            CPFState = CPFStates.Ascend;
        }

        EngrLogger.writeToColumns(sbStateTimedout);
    }
    break;
#endregion
case (CPFStates.DeAnchor):
#region
    if (SV.stateEntry)
    {
        basicStateEntry();
        Elmo.extendBellows(configFile.deanchorJV);
    }

    //normal state exit criteria goes here
    if (lpfResults.LPFVel >= configFile.deanchorVelocityThreshold)
    {

```

```

        Elmo.stopMotor();
        SV.anchored = false;
        SV.stateEntry = true;
        EngrLogger.writeToColumns(sbStateExit);
        //CPFStateTimer.Change(configFile.RunEngrTestTimeout, configFile.
timeoutDefaultPeriod);
        //CPFState = CPFStates.RunEngrTest;
        CPFStateTimer.Change(configFile.ascendTimeout, configFile.
timeoutDefaultPeriod);
        CPFState = CPFStates.Ascend;
        SV.stateTimedout = false;
    }

    if (SV.stateTimedout)
    {
        SV.stateEntry = true;
        SV.stateTimedout = false;
        EngrLogger.writeToColumns(sbStateTimedout);
        //lock (programLock) { Program.MessageQueue.Enqueue(UTF8Encoding.UTF8.
GetBytes(EngrLogger.getPrefix(sbStateTimedout).ToString())); }
        CPFStateTimer.Change(configFile.ascendTimeout, configFile.
timeoutDefaultPeriod);
        CPFState = CPFStates.Ascend;
    }
    break;
    #endregion
case (CPFStates.Surface):
    #region
    break;
    #endregion
case (CPFStates.EmergencyAscend):
    #region
    if (SV.stateEntry)
    {
        basicStateEntry();
        SV.checkStuckPressure = false;
    }
    SV.SORRecoveryMode = true;

    if (bellowsPosition < (configFile.bellowsEAPosition - 1.0))
    {
        if (!Elmo.extending)
            Elmo.extendBellows(configFile.EAJV);
    }
    else if (bellowsPosition > (configFile.bellowsEAPosition + 1.0))
    {
        if (!Elmo.retracting)
            Elmo.retractBellows(configFile.EAJV);
    }
    else
    {
        Elmo.stopMotor();
        SV.stateEntry = true;
        SV.stateTimedout = false;
        profileNum = profileNum + 1;
        EngrLogger.writeToColumns(sbEASStateExit);
        CPFStateTimer.Change(configFile.surfaceOpsTimeout, configFile.
timeoutDefaultPeriod);
        CPFState = CPFStates.SurfaceOps;
    }
    break;
    #endregion
case (CPFStates.ProcessErrors):
    #region
    basicStateEntry();
    break;

```

```

        #endregion
        case (CPFStates.Exit):
            #region
            basicStateEntry();        //This calls Elmo.StopMotor

            missionRun = false;
            //Make sure Elmo.stopMotor finishes
            Thread.Sleep(1000);
            break;
            #endregion
    }
    writeSystemStateMessage();

    if (sQueue.Count > 0)
    {
        processStructQ();
    }

    //uBlox.Parse_NMEA_Messages();

    Thread.Sleep(configFile.SMSleepPeriod);
}
catch (Exception currentException)
{
    sbTemp.Clear();
    sbTemp.Append("*** Exception in State Machine loop: ");
    sbTemp.Append(currentException.HResultToString());
    EngrLogger.writeToColumns(sbTemp);

    CPFStateTimer.Change(configFile.EATimeout, configFile.timeoutDefaultPeriod);
    CPFState = CPFStates.EmergencyAscend;
}
}

EngrLogger.closeFile();
EngrLogger.unMount();
//Iridium.PowerOff();
}

//Static variables relevant to queue processor
#region
private static BuoyancyControl.PIDLogValues pidLogValues;

private static byte[] dequeueMessage = new byte[256];

private static double pressure = double.NaN;
public static double bellowsPosition = double.NaN;
public static double batteryBusVolts = double.NaN;
public static double batteryBusAmps = double.NaN;
public static double amps12V = double.NaN;
public static double amps33V = double.NaN;
private static double[] CN0254Volts = new double[8];

private static SBE41.LPFResults lpfResults;

private static readonly StringBuilder SBE41Header = new StringBuilder("SBE41 Message");
private static readonly StringBuilder unparsedSBE41Header = new StringBuilder("Unparsed SBE41
Message");
private static readonly StringBuilder velPIDHeader = new StringBuilder("Vel PID Values");
private static readonly StringBuilder sbPMQ = new StringBuilder(256);
private static readonly StringBuilder sbConsoleCommand = new StringBuilder(32);
private static readonly StringBuilder consoleCommandNONE = new StringBuilder("NONE");
private static readonly StringBuilder CN0254Header = new StringBuilder("CN0254 Message");
private static readonly StringBuilder PTHHeader = new StringBuilder("PTH Message");
private static readonly StringBuilder OptodeHeader = new StringBuilder("Optode Message");
private static readonly StringBuilder EVK7Header = new StringBuilder("EVK7 GPS Message");

```

```

private static readonly StringBuilder IModemHeader = new StringBuilder("A3LA Message");
private static readonly StringBuilder DefaultHeader = new StringBuilder("Unknown Message: ");

private static byte[] returnedByteArray = new byte[StructQueue.qRecordWidth];
private static byte[] sbdwtByteArray = new byte[1024];
private static byte[] pubxByteArray = new byte[512];
private static byte[] sbdwtHeader = new byte[9] { (byte)65, (byte)84, (byte)43, (byte)83, (byte)66, (byte)68, (byte)87, (byte)84, (byte)61 }; // AT+SBDWT=
private static DateTime timeNow;

private static PTH.PTHData pthData;

private static int byteArrayLength = -1;

private static int msgLength = 0;

#endregion

private static void processStructQ()
{
    while (Program.sQueue.Count > 0)
    {
        lock (programLock) { qStructure = sQueue.Dequeue(); }
        switch (qStructure.qRecordType)
        {
            case (StructQueue.QRecordType.SBE41):
                #region
                byteArrayLength = Array.IndexOf(qStructure.byteArray, 0);
                //check for pressure only message from SBE41. This means message starts with a
                blank space and 8<=length=>10
                if ((qStructure.byteArray[0] == 32) && (byteArrayLength <= 10) && (byteArrayLength
                >= 8))
                {
                    pressure = SBE41.parsePOnly(qStructure.byteArray);
                    lpfResults = SBE41.LPFPPressure(pressure);
                    if (SV.runPVPID)
                    {
                        if (SV.runPVPID)
                        {
                            pidLogValues = BuoyancyControl.PIDPlatformVelocity(PVSetPoint,
                            lpfResults.diffVel);
                            if (verbosityLevel > 2)
                            {
                                EngrLogger.writeToColumns(EngrLogger.ColumnNums.dateTime,
                                EngrLogger.ColumnNums.state, qStructure
                                .state,
                                EngrLogger.ColumnNums.comment,
                                velPIDHeader,
                                EngrLogger.ColumnNums.PID_setPoint,
                                pidLogValues.setPoint,
                                EngrLogger.ColumnNums.PID_PV,
                                pidLogValues.PV,
                                EngrLogger.ColumnNums.PID_y1,
                                pidLogValues.y1,
                                EngrLogger.ColumnNums.PID_y2,
                                pidLogValues.y2,
                                EngrLogger.ColumnNums.PID_I,
                                pidLogValues.I,
                                EngrLogger.ColumnNums.PID_v,
                                pidLogValues.v,
                                EngrLogger.ColumnNums.PID_u,
                                pidLogValues.u,
                                EngrLogger.ColumnNums.PID_uCPS,
                                pidLogValues.uCPS); //TODO Should do this in a way that the message is time stamped at the source
                            }
                        }
                    }
                }
            }
        }
    }
}

```

```

    }
}
if (verbosityLevel > 2)
{
    EngrLogger.writeToColumns(EngrLogger.ColumnNums.dateTime, qStructure.
timeStamp,
                                EngrLogger.ColumnNums.state, qStructure.state,
                                EngrLogger.ColumnNums.comment, SBE41Header,
                                EngrLogger.ColumnNums.pressure, pressure,
                                EngrLogger.ColumnNums.CTDSalinity, qStructure.
byteArray,
                                EngrLogger.ColumnNums.platformLPFVel,
lpfResults.LPFVel,
                                EngrLogger.ColumnNums.platformDiffVel,
lpfResults.diffVel,
                                EngrLogger.ColumnNums.platformLPFAcc,
lpfResults.LPFAcc,
                                EngrLogger.ColumnNums.pressureAverage,
lpfResults.pressureAverage,
                                EngrLogger.ColumnNums.pressureVariance,
lpfResults.pressureVariance); //TODO Should do this in a way that the message is time stamped at the
source
    }

    //Check depth limit
    #region
    if (lpfResults.LPFPressure > configFile.maxPressure)
    {
        Elmo.stopMotor();
        SV.runPVPID = false;
        EngrLogger.writeToColumns(toDeepError);
        CPFStateTimer.Change(configFile.EATimeout, configFile.timeoutDefaultPeriod)
;
        CPFState = CPFStates.EmergencyAscend;
    }
    #endregion

}
else
{
    //log any non-pressure only messages
    EngrLogger.writeToColumns(EngrLogger.ColumnNums.dateTime, qStructure.timeStamp,
                                EngrLogger.ColumnNums.state, qStructure.state,
                                EngrLogger.ColumnNums.dateTime, qStructure.
timeStamp,
                                EngrLogger.ColumnNums.state, qStructure.state,
                                EngrLogger.ColumnNums.comment, unparsedSBE41Header,
                                EngrLogger.ColumnNums.CTDSalinity, qStructure.
byteArray);
}

//Check for SBE41 not updating pressure
#region
if (SV.checkStuckPressure)
{
    if (ErrorHandler.isPressureStuck(pressure))
    {
        Elmo.stopMotor();
        SV.runPVPID = false;
        EngrLogger.writeToColumns(pressureStuckError);
        CPFStateTimer.Change(configFile.EATimeout, configFile.timeoutDefaultPeriod)
;
        CPFState = CPFStates.EmergencyAscend;
    }
}
#endregion

```

```

//Check for expected response
#region
msgLength = qStructure.byteArray.Length;

//check for expectedResponse anywhere in the SBE41 response message
if (SBE41.waitingForResponse)
{
    if (msgLength > SBE41.erLength)
    {
        for (int j = 0; j < msgLength - SBE41.erLength; j++)
        {
            for (int k = 0; k < SBE41.erLength; k++)
            {
                if (qStructure.byteArray[j + k] != SBE41.expectedResponse[k])
                {
                    SBE41.gotResponse = false;
                    break;
                }
            }
        }
        SBE41.gotResponse = true;
    }
}

//check for "S>dd" anywhere in the SBE41 response message
if (msgLength > 4)
{
    for (int j = 0; j < msgLength - 3; j++)
    {
        if ((qStructure.byteArray[j] == 83) && (qStructure.byteArray[j + 1] == 62)
            && (qStructure.byteArray[j + 2] == 100) && (qStructure.byteArray[j + 3] == 100))
        {
            ddRX = true;
            break;
        }
    }
}

//check for "upload command" anywhere in the SBE41 response message
if (msgLength > 14)
{
    for (int j = 0; j < msgLength - 13; j++)
    {
        //117 112 108 111 097 100 032 099 111 109 112 108 101 116 101
        if ((qStructure.byteArray[j] == 117) && (qStructure.byteArray[j + 1] == 112)
            && (qStructure.byteArray[j + 2] == 108) && (qStructure.byteArray[j + 3] == 111)
            && (qStructure.byteArray[j + 4] == 097) && (qStructure.byteArray[j + 5] == 100)
            && (qStructure.byteArray[j + 6] == 032) && (qStructure.byteArray[j + 7] == 099)
            && (qStructure.byteArray[j + 8] == 111) && (qStructure.byteArray[j + 9] == 109)
            && (qStructure.byteArray[j + 10] == 112) && (qStructure.byteArray[j + 11] == 108)
            && (qStructure.byteArray[j + 12] == 101) && (qStructure.byteArray[j + 13] == 116)
            && (qStructure.byteArray[j + 14] == 101))
        {
            uploadComplete = true;
            break;
        }
    }
}

```

```

    }
}
#endregion

break;
#endregion
case (StructQueue.QRecordType.Optode):
    #region
    for (int i = 0; i < qStructure.byteArray.Length; i++)
    {
        if (qStructure.byteArray[i] == 9)
            qStructure.byteArray[i] = 44;
    }

    EngrLogger.writeToColumns(EngrLogger.ColumnNums.dateTime, qStructure.timeStamp,
                             EngrLogger.ColumnNums.state, qStructure.state,
                             EngrLogger.ColumnNums.comment, OptodeHeader,
                             EngrLogger.ColumnNums.TOpt, qStructure.byteArray);

    //check for expectedResponse anywhere in the SBE41 response message
    msgLength = qStructure.byteArray.Length;

    if (Optode.waitingForResponse)
    {
        if (msgLength > Optode.erLength)
        {
            for (int j = 0; j < msgLength - Optode.erLength; j++)
            {
                for (int k = 0; k < Optode.erLength; k++)
                {
                    if (qStructure.byteArray[j + k] != Optode.expectedResponse[k])
                    {
                        Optode.gotResponse = false;
                        break;
                    }
                }
            }
            Optode.gotResponse = true;
        }
    }

    break;
#endregion
case (StructQueue.QRecordType.EVK7):
    #region
    Array.Clear(pubxByteArray, 0, pubxByteArray.Length);
    Array.Copy(qStructure.byteArray, pubxByteArray, Array.IndexOf(qStructure.byteArray, ✎
0));

    EngrLogger.writeToColumns(EngrLogger.ColumnNums.dateTime, qStructure.timeStamp,
                             EngrLogger.ColumnNums.state, qStructure.state,
                             EngrLogger.ColumnNums.comment, EVK7Header,
                             EngrLogger.ColumnNums.GPS, qStructure.byteArray);

    break;
#endregion
case (StructQueue.QRecordType.IModem):
    #region
    EngrLogger.writeToColumns(EngrLogger.ColumnNums.dateTime, qStructure.timeStamp,
                             EngrLogger.ColumnNums.state, qStructure.state,
                             EngrLogger.ColumnNums.comment, IModemHeader,
                             EngrLogger.ColumnNums.IModem, qStructure.byteArray);

    if (surfaceOpsSubstate == 3)
        IModem.parseSQ(qStructure.byteArray);

    break;

```

```

        #endregion
        case (StructQueue.QRecordType.PTH):
            #region
            pthData = PTH.getPTH();
            timeNow = DateTime.Now;
            EngrLogger.writeToColumns(EngrLogger.ColumnNums.dateTime, timeNow,
                                     EngrLogger.ColumnNums.state, CPFState,
                                     EngrLogger.ColumnNums.comment, PTHHeader,
                                     EngrLogger.ColumnNums.LPS331Pressure, pthData. ✓
            LPS331Pressure,
                                     EngrLogger.ColumnNums.LPS331Temperature, pthData. ✓
            LPS331Temperature,
                                     EngrLogger.ColumnNums.SHT21Humidity, pthData. ✓
            SHT21Humidity,
                                     EngrLogger.ColumnNums.SHT21Temperature, pthData. ✓
            SHT21Temperature);
            break;
            #endregion
        case (StructQueue.QRecordType.BTConsole):
            #region
            processConsoleCommand(qStructure.byteArray);
            break;
            #endregion
        case (StructQueue.QRecordType.CN0254):
            #region
            CN0254Volts = CN0254.scanAll();
            timeNow = DateTime.Now;
            bellowsPosition = (CN0254Volts[0] * configFile.cn0254CH0Scale) + configFile. ✓
            cn0254CH0offset;
            batteryBusVolts = (CN0254Volts[1] * configFile.cn0254CH1Scale) + configFile. ✓
            cn0254CH1offset;
            batteryBusAmps = (CN0254Volts[2] * configFile.cn0254CH2Scale) + configFile. ✓
            cn0254CH2offset;
            amps12V = (CN0254Volts[3] * configFile.cn0254CH3Scale) + configFile.cn0254CH3Offset ✓
            ;
            amps33V = (CN0254Volts[4] * configFile.cn0254CH4Scale) + configFile.cn0254CH4Offset ✓
            ;
            if (verbosityLevel > 2)
            {
                EngrLogger.writeToColumns(EngrLogger.ColumnNums.dateTime, timeNow,
                                         EngrLogger.ColumnNums.state, CPFState,
                                         EngrLogger.ColumnNums.comment, CN0254Header,
                                         EngrLogger.ColumnNums.bellowsPosition, ✓
            bellowsPosition,
                                         EngrLogger.ColumnNums.batteryBusVolts, ✓
            batteryBusVolts,
                                         EngrLogger.ColumnNums.batteryBusAmps, ✓
            batteryBusAmps,
                                         EngrLogger.ColumnNums.Amps12V, amps12V,
                                         EngrLogger.ColumnNums.Amps33V, amps33V);
            }

            if ((CPFState != CPFStates.EmergencyAscend) && (CPFState != CPFStates.NotRunning))
            {
                //if ((bellowsPosition > configFile.bellowsUpperLimit) || (bellowsPosition < ✓
            configFile.bellowsLowerLimit))
                //{
                //    Elmo.stopMotor();
                //    SV.runPVPID = false;
                //    EngrLogger.writeToColumns(BPError);
                //    CPFStateTimer.Change(configFile.EATimeout, configFile. ✓
            timeoutDefaultPeriod);
                //    CPFState = CPFStates.EmergencyAscend;
                //}

            if (batteryBusVolts < configFile.minBatteryBusVolts)

```

```

        {
            Elmo.stopMotor();
            SV.runPVPID = false;
            EngrLogger.writeToColumns(BBVError);
            CPFStateTimer.Change(configFile.EATimeout, configFile.timeoutDefaultPeriod) ↵
;

            CPFState = CPFStates.EmergencyAscend;
        }
    }
    break;
    #endregion
default:
    #region
    timeNow = DateTime.Now;
    sbTemp.Clear();
    sbTemp.Append(DefaultHeader);
    sbTemp.Append(qStructure.qRecordType);
    sbTemp.Append(" ");
    sbTemp.Append(UTF8Encoding.UTF8.GetChars(qStructure.byteArray));
    EngrLogger.writeToColumns(EngrLogger.ColumnNums.dateTime, timeNow,
                                EngrLogger.ColumnNums.state, CPFState,
                                EngrLogger.ColumnNums.comment, DefaultHeader);

    break;
    #endregion
}
//sbTemp.Clear();
//sbTemp.Append(UTF8Encoding.UTF8.GetChars(qStructure.byteArray));
//Debug.Print(sbTemp.ToString());
}
}

private static void CPFStateTimerCallback(object o)
{
    SV.stateTimedout = true;
}

private static StructQueue.qStruct ADCQStruct = new StructQueue.qStruct()
{
    qRecordType = StructQueue.QRecordType.CN0254,
    byteArray = new byte[StructQueue.qRecordWidth]
};
private static object ADCTimerLock = new Object();

private static void ADCTimerCallback(object o)
{
    lock (ADCTimerLock) { Program.sQueue.Enqueue(ADCQStruct); }
}

private static StructQueue.qStruct PTHQStruct = new StructQueue.qStruct()
{
    qRecordType = StructQueue.QRecordType.PTH,
    byteArray = new byte[StructQueue.qRecordWidth]
};
private static object PHTimerLock = new Object();

private static void PHTimerCallback(object o)
{
    PTHQStruct.timeStamp = DateTime.Now;
    PTHQStruct.state = CPFState;
    lock (PHTimerLock) { Program.sQueue.Enqueue(PTHQStruct); }
}

private static StringBuilder consoleCommandComment = new StringBuilder();
private static bool directoryValid = false;
private static int uploadEngrFileNum = -1;

```

```

private static void processConsoleCommand(byte[] consoleCommandMessage)
{
    //Ignore command if it doesn't have a CR
    if (Array.IndexOf(consoleCommandMessage, 13) >= 0)
    {
        BTConsole.BTSend = true;
        sbConsoleCommand.Clear();
        sbConsoleCommand.Append(UTF8Encoding.UTF8.GetChars(consoleCommandMessage, 1, (Array.IndexOf
(consoleCommandMessage, 13) - 1)));
        consoleCommandComment.Clear();
        consoleCommandComment.Append("Processing Console Command: ");
        consoleCommandComment.Append(sbConsoleCommand);
        EngrLogger.writeToColumns(consoleCommandComment);

        switch (sbConsoleCommand.ToString().ToUpper())
        {
            case "DIR":
                break;
            case "DOWNENGR":
                break;
            case "DOWNMIS":
                break;
            case "GETDIR":
                EngrLogger.getDirectory();
                directoryValid = true;
                break;
            case "UPDATA":
                break;
            case "GO":
                SV.surfaceOpsGo = true;
                break;
            case "RECOVERYTRUE":
                SV.SORecoveryMode = true;
                break;
            case "RECOVERYFALSE":
                SV.SORecoveryMode = false;
                break;
            case "EXIT":
                CPFState = CPFStates.Exit;
                break;
            case "UPENGR0":
                SV.uploadEngrFile = true;
                uploadEngrFileNum = 0;
                break;
            case "UPENGR1":
                SV.uploadEngrFile = true;
                uploadEngrFileNum = 1;
                break;
            case "UPENGR2":
                SV.uploadEngrFile = true;
                uploadEngrFileNum = 2;
                break;
            case "UPENGR3":
                SV.uploadEngrFile = true;
                uploadEngrFileNum = 3;
                break;
            case "UPENGR4":
                SV.uploadEngrFile = true;
                uploadEngrFileNum = 4;
                break;
            case "UPENGR5":
                SV.uploadEngrFile = true;
                uploadEngrFileNum = 5;
                break;
            case "UPENGR6":
                SV.uploadEngrFile = true;

```

```

        uploadEngrFileNum = 6;
        break;
    case "UPENGR7":
        SV.uploadEngrFile = true;
        uploadEngrFileNum = 7;
        break;
    case "UPENGR8":
        SV.uploadEngrFile = true;
        uploadEngrFileNum = 8;
        break;
    case "UPENGR9":
        SV.uploadEngrFile = true;
        uploadEngrFileNum = 9;
        break;
    default:
        sbTemp.Clear();
        sbTemp.Append("Invalid command, try again");
        BTConsole.SendLine(sbTemp);
        break;
    }
}
else
{
    sbConsoleCommand.Append(UTF8Encoding.UTF8.GetChars(consoleCommandMessage, 1, (Array.IndexOf
(consoleCommandMessage, 0) - 1)));
    consoleCommandComment.Clear();
    consoleCommandComment.Append("Received non-CR terminated Console Command: ");
    consoleCommandComment.Append(sbConsoleCommand);
    EngrLogger.writeToColumns(consoleCommandComment);
}

sbConsoleCommand.Clear();
sbConsoleCommand.Append("NONE");
}

private static void writeSystemStateMessage()
{
    timeNow = DateTime.Now;
    sbTemp.Clear();
    sbTemp.Append("SYSTEM STATE MSG (pres, bel pos, bat v) = ");
    sbTemp.Append(pressure.ToString("f2"));
    sbTemp.Append(", ");
    sbTemp.Append(bellowsPosition.ToString("f2"));
    sbTemp.Append(", ");
    sbTemp.Append(batteryBusVolts.ToString("f1"));
    sbTemp.Append(", ");

    if (SV.runPVPID)
        sbTemp.Append("PVPID Running,");
    sbTemp.Append(",");

    sbTemp.Append(sbScendStateName);
    sbTemp.Append(",");

    if (Elmo.extending)
        sbTemp.Append("Extending");
    sbTemp.Append(",");
    if (Elmo.retracting)
        sbTemp.Append("Retracting");
    sbTemp.Append(",");
    if (Elmo.motorStopped)
        sbTemp.Append("MotorStopped");
    sbTemp.Append(",");

    sbTemp.Append("depthNum = ");
    sbTemp.Append(pressureTableNum.ToString());

```

```

        EngrLogger.writeToColumns(EngrLogger.ColumnNums.dateTime, timeNow,
                                   EngrLogger.ColumnNums.state, CPFState,
                                   EngrLogger.ColumnNums.comment, sbTemp,
                                   EngrLogger.ColumnNums.pressure, pressure,
                                   EngrLogger.ColumnNums.bellowsPosition, bellowsPosition,
                                   EngrLogger.ColumnNums.batteryBusVolts, batteryBusVolts);
    }

    private static void basicStateEntry()
    {
        Elmo.stopMotor();
        EngrLogger.writeToColumns(sbStateEntry);
        SV.runPVPID = false;
        SV.stateEntry = false;
        SV.stateTimedout = false;
        sbScendStateName.Clear();
        sbScendStateName.Append(sbScendNameNone);
    }

    private static StringBuilder sbReturnArray = new StringBuilder(64);

    private static byte[] getField(byte[] inArray, byte token, int desiredField)
    {
        byte[] returnArray = new byte[64];

        int startIndex = 0;
        int endIndex = inArray.Length;
        int fieldNum = 0;

        bool gotField = false;

        for (int i = 0; i < inArray.Length; i++)
        {
            if (inArray[i] == token)
            {
                if (desiredField == 0)
                {
                    endIndex = i;
                    gotField = true;
                    break;
                }
                else
                {
                    fieldNum = fieldNum + 1;
                    if (fieldNum == desiredField)
                    {
                        startIndex = i + 1;
                        endIndex = Array.IndexOf(inArray, token, startIndex);
                        if (endIndex < 0)
                            endIndex = inArray.Length;
                        gotField = true;
                        break;
                    }
                }
            }
        }

        Array.Clear(returnArray, 0, returnArray.Length);
        if (gotField || (desiredField == 0))
            Array.Copy(inArray, startIndex, returnArray, 0, endIndex - startIndex);

        sbReturnArray.Clear();
        sbReturnArray.Append(UTF8Encoding.UTF8.GetChars(returnArray));
        Debug.Print(sbReturnArray.ToString());
        return (returnArray);
    }

```

```
    }  
}  
}
```