```
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.Premium.IO;
using GHI.Premium.System;
using GHI.Premium.Hardware;
using GHI.Hardware.EMX;

namespace miniCPF
{
    public class Program
    {
        public static PersistentStorage sdCard = new PersistentStorage("SD");
        private static Timer CPFStateTimer = null;

        public static string fileName;

        public static DateTime timeNow;
        public static DateTime synchDateTime;

        public static byte[] writeByteArray = new byte[1024];

        public static UInt32 todTicks;
        private static int GPSMessageNum = 0;

        private static int surfaceOpsTimeout = 10000;

        private static bool buffer1Ready = false;
        private static bool timeSynched = false;
        private static bool missionRun = true;
        private static bool RMCReady = false;
        private static bool stateTimedout = false;
        private static bool SurfaceOpsTimeoutStarted = false;

        private static byte[] GPSInByteArray = new byte[102400];
        private static char[] GPSInCharArray = new char[204800];
        private static byte[] RMCByteArray = new byte[102400];
        private static char[] RMCCharArray = new char[204800];
        private static int GPSByteNum = 0;

        public struct CPFDateTime
        {
            public static int year;
            public static int month;
            public static int day;
            public static int second;
        }
        CPFDateTime CPFDateTimeNow;

        public enum CPFStates : int
        {
            StartInitialize = 0,
            Initialize,
            StartPreMissionDelay,
            PreMissionDelay,
            StartSurfaceOps,
            SurfaceOps,
            StartAutoballast,
```

```
            Autoballast,
            StartDescend,
            Descend,
            StartDescendPark,
            DescendPark,
            StartAscend,
            Ascend,
            StartAscendPark,
            AscendPark,
            StartEmergencyAscend,
            EmergencyAscend,
            StartRecovery,
            Recovery,
            ProcessErrors,
            Exit,
        }
        public static CPFStates CPFState;

        public static SerialPort GPSPort = new SerialPort("COM3", 9600, Parity.None);

        public static void Main()
        {
            Debug.Print("Program Started     " + DateTime.Now.ToString() + "." + DateTime.Now.Millisecond. ↙
    ToString());
            CPFStateTimer = new Timer(new TimerCallback(CPFStateTimerCallback), null, 1000000, 1000000);


            CPFState = CPFStates.StartInitialize;

            sdCard.MountFileSystem();
            VolumeInfo sdVol = new VolumeInfo("SD");
            Debug.Print(sdVol.Name.ToString());

            fileName = sdVol.RootDirectory + "\\fileStreamTest";
            FileStream testFile = new FileStream(fileName, FileMode.OpenOrCreate, FileAccess.Write);

            while (missionRun)
            {
                switch (CPFState)
                {
                    case (CPFStates.StartInitialize):
                        Debug.Print("State = Start Intialize");
                        CPFState = CPFStates.Initialize;
                        break;
                    case (CPFStates.Initialize):
                        Debug.Print("State = Intialize");
                        CPFState = CPFStates.StartPreMissionDelay;
                        break;
                    case (CPFStates.StartPreMissionDelay):
                        Debug.Print("State = Start PremissionDelay");
                        CPFState = CPFStates.PreMissionDelay;
                        break;
                    case (CPFStates.PreMissionDelay):
                        Debug.Print("State = PreMissionDelay");
                        CPFState = CPFStates.StartSurfaceOps;
                        break;
                    case (CPFStates.StartSurfaceOps):
                        Debug.Print("State = Start SurfaceOps " + DateTime.Now.ToString());

                        GPSPort.Open();
                        GPSPort.DiscardInBuffer();
                        GPSPort.DiscardOutBuffer();
                        GPSPort.DataReceived += new SerialDataReceivedEventHandler(GPSPortDataReceived);

                        CPFState = CPFStates.SurfaceOps;
                        break;
```

```
                    case (CPFStates.SurfaceOps):
                        //Debug.Print("State = SurfaceOps");

                        if (timeSynched && !SurfaceOpsTimeoutStarted)
                        {
                            CPFStateTimer.Change(surfaceOpsTimeout, 10000);
                            Debug.Print("Starting SurfaceOpsTimeout Timer " + DateTime.Now.ToString() + "." ↙
     + DateTime.Now.Millisecond.ToString());
                            SurfaceOpsTimeoutStarted = true;                //TODO need to clean up this    ↙
    logic
                        }

                        if (RMCReady)
                        {
                            parseRMC(RMCByteArray);
                            RMCReady = false;
                        }
                        if (stateTimedout)
                        {
                            GPSPort.DiscardInBuffer();
                            GPSPort.Close();
                            CPFStateTimer.Change(1000000, 1000000);
                            CPFState = CPFStates.StartAutoballast;
                        }
                        break;
                    case (CPFStates.StartAutoballast):
                        Debug.Print("State = Start Autoballast");
                        CPFState = CPFStates.StartAutoballast;
                        break;
                    case (CPFStates.Autoballast):
                        Debug.Print("State = Autoballast");
                        CPFState = CPFStates.StartDescend;
                        break;
                    case (CPFStates.StartDescend):
                        Debug.Print("State = Start Descend");
                        CPFState = CPFStates.Descend;
                        break;
                    case (CPFStates.Descend):
                        Debug.Print("State = Descend");
                        CPFState = CPFStates.StartDescendPark;
                        break;
                    case (CPFStates.StartDescendPark):
                        Debug.Print("State = Start DescendPark");
                        CPFState = CPFStates.DescendPark;
                        break;
                    case (CPFStates.DescendPark):
                        Debug.Print("State = Descendpark");
                        CPFState = CPFStates.StartAscend;
                        break;
                    case (CPFStates.StartAscend):
                        Debug.Print("State = Start Ascend");
                        CPFState = CPFStates.Ascend;
                        break;
                    case (CPFStates.Ascend):
                        Debug.Print("State = Ascend");
                        CPFState = CPFStates.StartAscendPark;
                        break;
                    case (CPFStates.StartAscendPark):
                        Debug.Print("State = Start AscendPark");
                        CPFState = CPFStates.AscendPark;
                        break;
                    case (CPFStates.AscendPark):
                        Debug.Print("State = AscendPark");
                        CPFState = CPFStates.SurfaceOps;
                        break;
                    case (CPFStates.StartEmergencyAscend):
```

```
                    Debug.Print("State = Start EmergencyAscend");
                    CPFState = CPFStates.EmergencyAscend;
                    break;
                case (CPFStates.EmergencyAscend):
                    Debug.Print("State = EmergencyAscend");
                    CPFState = CPFStates.StartRecovery;
                    break;
                case (CPFStates.StartRecovery):
                    Debug.Print("State = Start Recovery");
                    CPFState = CPFStates.Recovery;
                    break;
                case (CPFStates.Recovery):
                    Debug.Print("State = Recovery");
                    break;
                case (CPFStates.ProcessErrors):
                    Debug.Print("State = ProcessErrors");
                    break;
                case (CPFStates.Exit):
                    Debug.Print("State = Exit");
                    missionRun = false;
                    break;
            }

            if (buffer1Ready)
            {
                timeNow = DateTime.Now;
                todTicks = (UInt32)(timeNow.TimeOfDay.Ticks / 10000);
                Microsoft.SPOT.Hardware.Utility.InsertValueIntoArray(writeByteArray, 0, 4, todTicks);

                testFile.Write(writeByteArray, 0, writeByteArray.Length);
                Debug.Print(todTicks.ToString());
                Thread.Sleep(1000);
            }

            Thread.Sleep(100);
        }
        testFile.Close();
        GPSPort.Close();
        sdCard.UnmountFileSystem();
    }

    private static void GPSPortDataReceived(object sender, SerialDataReceivedEventArgs e)
    {
        int GPSPortBytesToRead = 0;

        GPSPortBytesToRead = GPSPort.BytesToRead;
        byte[] inBytes = new byte[GPSPortBytesToRead];

        try
        {
            GPSPort.Read(inBytes, 0, GPSPortBytesToRead);
        }
        catch
        {
            Debug.Print("GPS Read Exception");          //TODO do something smarter here
        }

        for (int i = 0; i < GPSPortBytesToRead; i++)
        {
            GPSInByteArray[GPSByteNum] = inBytes[i];
            if (GPSByteNum == 0)    //Check to make sure first character is = $ (NMEA standard start of ↙
string)
            {
                if (GPSInByteArray[0] == 36)
                {
                    GPSByteNum = GPSByteNum + 1;
```

```
                }
            }
            else
            {
                if (GPSInByteArray[GPSByteNum] == 10)
                {
                    GPSMessageNum = GPSMessageNum + 1;
                    Debug.Print("Got GPS Message Number = " + GPSMessageNum.ToString());
                    //GPSInCharArray = Encoding.UTF8.GetChars(GPSInByteArray);
                    if (!timeSynched)
                    {
                        if ((GPSInByteArray[1] == (byte)'G') && (GPSInByteArray[2] == (byte)'P') &&
(GPSInByteArray[3] == (byte)'R') &&
                            (GPSInByteArray[4] == (byte)'M') && (GPSInByteArray[5] == (byte)'C'))
                        {
                            for (int j = 0; j < GPSByteNum; j++)
                            {
                                RMCByteArray[j] = GPSInByteArray[j];
                            }
                            //RMCCharArray = Encoding.UTF8.GetChars(RMCByteArray);
                            RMCReady = true;
                        }
                    }
                    GPSByteNum = 0;
                }
                else
                {
                    GPSByteNum = GPSByteNum + 1;
                }
            }
        }
    }


    private static void parseRMC(byte[] RMCByteArray)
    {
        int tokenCounter = 0;
        int hour = 0;
        int minute = 0;
        int second = 0;
        int year = 0;
        int month = 0;
        int day = 0;


        for (int i = 0; i < 80; i++)
        {
            if (RMCByteArray[i] == 44)
            {
                tokenCounter = tokenCounter + 1;
                if (tokenCounter == 1)
                {
                    hour = (RMCByteArray[i + 1] - 48) * 10 + (RMCByteArray[i + 2] - 48);
                    minute = (RMCByteArray[i + 3] - 48) * 10 + (RMCByteArray[i + 4] - 48);
                    second = (RMCByteArray[i + 5] - 48) * 10 + (RMCByteArray[i + 6] - 48);
                }
                if (tokenCounter == 9)
                {
                    day = (RMCByteArray[i + 1] - 48) * 10 + (RMCByteArray[i + 2] - 48);
                    month = (RMCByteArray[i + 3] - 48) * 10 + (RMCByteArray[i + 4] - 48);
                    year = 2000 + (RMCByteArray[i + 5] - 48) * 10 + (RMCByteArray[i + 6] - 48);
                    synchTime(year, month, day, hour, minute, second);
                }
            }
        }
```

```
        }

        private static void synchTime(int year, int month, int day, int hour, int minute, int second)
        {
            synchDateTime = new DateTime(year, month, day, hour, minute, second);

            //Utility.SetLocalTime(synchDateTime);
            //RealTimeClock.SetTime(synchDateTime); //This will set the hardware Real-time Clock to what is ↙
     in DT

            //Thread.Sleep(2000);
            Debug.Print("DateTime.Now = " + DateTime.Now.ToString());
            Debug.Print("RTC.Now = " + RealTimeClock.GetTime().ToString());
            timeSynched = true;
        }

        private static void CPFStateTimerCallback(object o)
        {
            Debug.Print("CPF State Timed Out " + DateTime.Now.ToString() + "." + DateTime.Now.Millisecond. ↙
    ToString());
           stateTimedout = true;
        }
    }
}
```