Affordable, Flexible, and Modular: a Guide to Open-Source Membrane-Based Water Treatment Systems

Adam Slade, David Jassby*

Department of Chemical and Environmental Engineering, University of California – Riverside, Riverside, CA 92521, USA

**Corresponding author email: djassby@engr.ucr.edu*

Appendix C

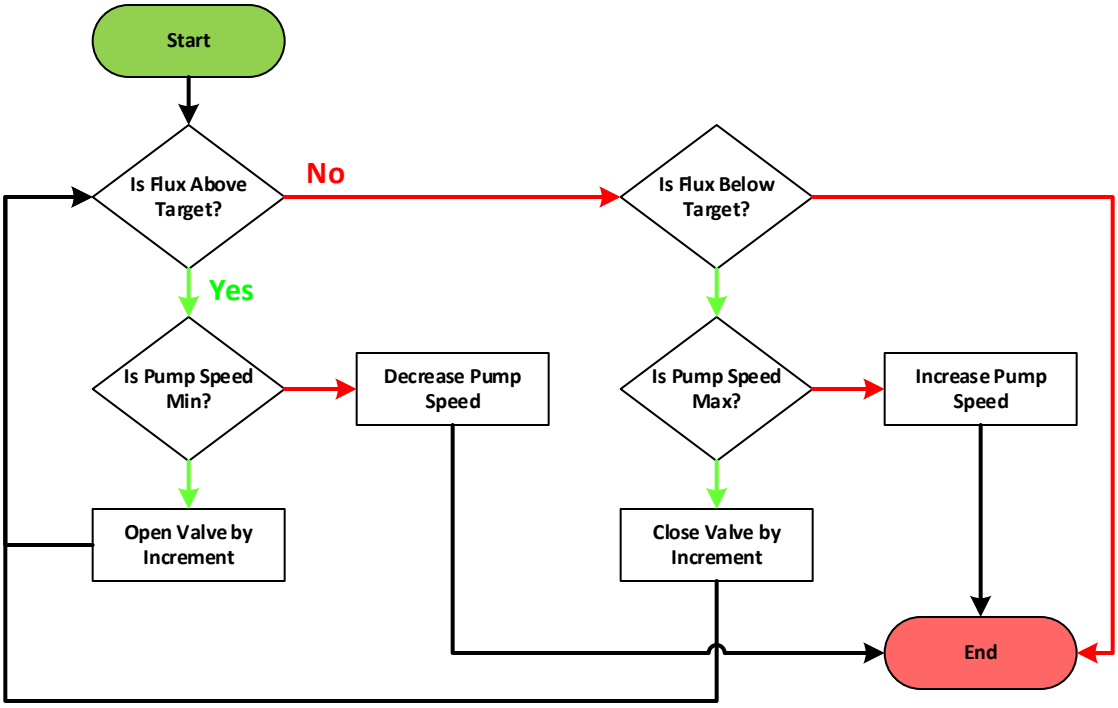
Flux/Pressure Control Code

The following is the algorithm used for flux control, taken from the computer software code. The computer program gathers environmental data from the Arduino (which retrieves and conditions it from the sensors), uses it within the control algorithms, and sends commands to the Arduino which initiate control actuations (such as opening a valve, or changing the pump speed). The software was written in VB.NET, though can be easily translated to any programming language provided it interfaces with the Arduino's serial communications.

Two distinct control algorithms are used: (1) flux control via varying pump speed, and (2) flux control via varying valve opening. Both are used, as varying the valve opening was a coarse flux adjustment, while varying the pump speed was a fine flux adjustment. In practice, whether an adjustment is coarse or fine will depend on the resolution of the actuators used.

Finally, a selection of all control codes are included for reference. Other supporting functions and subroutines are omitted for brevity.

Software Flow Diagram for Flux Control



Flux Control Algorithm: Pump Speed

```
'match the flux with the pump speed

    Private Sub MatchFluxDOIPump()


        'flux matching I (for pump adjustment)
        Dim newval As Decimal
        If isPressureHold Then
            If Pressure(1) > DOIPressureTarget Then
                newval = CDec(numPumpFreq_Value - (60.0 / 255))
            Else
                newval = CDec(numPumpFreq_Value + (60.0 / 255))
            End If
        Else
```

```
            If Flowrate(1) > GetFlowrateSet() Then
                newval = CDec(numPumpFreq_Value - (60.0 / 255))
            Else
                newval = CDec(numPumpFreq_Value + (60.0 / 255))
            End If
        End If

        'Check Bounds
        If newval > minPumpFreq Then
            If newval < maxPumpFreq Then
                SetNumeric(numMainPump, newval) 'okay
                StatusColor = Color.Lime
            Else
                SetNumeric(numMainPump, maxPumpFreq) 'pump speed is too high and filtrate
flow is too low (close valve if possible)
                StatusColor = Color.Red
                DOI_High_Count = 0 'give control to valve
            End If
        Else
            SetNumeric(numMainPump, minPumpFreq) 'pump speed too low and filtrate flow is
too high (open valve if possible)
            StatusColor = Color.Red
            DOI_High_Count = 0 'give control to valve
        End If

    End Sub
```

1.1 Flux Control Algorithm: Valve Position

```
'match the flux with the valve
```

```
    Private Sub MatchFluxDOIValve()


        'Flux Matching
        Dim newval As Decimal
        If isPressureHold Then
            If Pressure(1) > DOIPressureTarget Then
                newval = CDec(numDOI_Value + (100.0 / 255.0))
                DOI_High_Count += 1 'count as one against valve control (switches to pump
after 10)
            Else
                newval = CDec(numDOI_Value - (100.0 / 255.0))
            End If
        Else
            If Flowrate(1) > GetFlowrateSet() Then
                newval = CDec(numDOI_Value + (100.0 / 255.0))
                DOI_High_Count += 1 'count as one against valve control (switches to pump
after 10)
            Else
                newval = CDec(numDOI_Value - (100.0 / 255.0))
            End If
        End If



        'Check Bounds
        If newval > 0 Then
            If newval < 100 Then
                SetNumeric(numDOI, newval) 'okay
                StatusColor = Color.Lime
            Else
```

```
                SetNumeric(numDOI, 100) 'valve opening is too high and filtrate flow is
too high (slow down pump if possible)
                StatusColor = Color.Red
                DOI_High_Count = 10 'give control to pump
            End If
        Else
            SetNumeric(numDOI, 0) 'valve opening is too small and filtrate flow is too
low (speed up pump if possible)
            StatusColor = Color.Red
            DOI_High_Count = 10 'give control to pump
        End If

    End Sub
```

1.2 Software Code for Main Winform Form

```
Imports System.Threading.Thread 'sleep access
Imports System.IO.Ports 'arduino access

Public Class frmMain

#Region "Common Variables"

    Dim filepath As String = "" 'data filepath
    Dim starthour As Double 'time of process start
    Dim isLoaded As Boolean 'has the GUI finished loading?
    Dim isBackwashPending As Boolean 'DOI/AF backwash trigger
    Dim isCleaningPending As Boolean 'DOI cleaning trigger
    Dim isNFCleaningPending As Boolean 'NF cleaning trigger
    Public Shared isDOI As Boolean 'is DOI project?
```

```
    Dim isPaused As Boolean 'is data collection active?
    Dim isUpdaterRunning As Boolean 'is data storage active?
    Dim isSimulation As Boolean 'is a simulation (no hardware)?
    Public Shared pps As Integer 'pulses/s (flowmeter calibration)
    Public Shared isAnnotationPending As Boolean 'annotation trigger
    Public Shared AnnotationText As String 'annotation
    Dim isPressureHold As Boolean 'is control mode pressure?

    '2nd-order polynomial fit coefficients for pressure (DOI)
    Dim poly(2) As Double

    '2nd-order polynomial fit coefficients for pressure (AF)
    Dim poly_UF(2) As Double
    Dim poly_NF(2) As Double
    Dim poly_Lev(2) As Double

#End Region

#Region "Sensor Parameters"

    '*** COMMON PARAMETERS ***
    Public Shared PulsePerLiter(2) As Double 'number of flowmeter pulses per liter
    Public Shared FlowmeterOffset(2) As Double 'offset of flowmeter
    Public Shared MaxFlow As Double = My.Settings.MaxFlowrate '[L/min] - maximum rated
flowrate for flowmeter
    Dim PressureSamples As Integer = 100 'number of analog pressure samples to average
    Dim SensorRespond As ULong = 1000 '[ms] - time the sensors take to update (auto-
updates)
```

```
    Dim ValveMoveTime As ULong = 353 '[ms] - time for the valve to move 1/255 of the
distance (90/255)
    Public Shared FaultTimeF As ULong = My.Settings.HighFlowTime * 1000 '[ms] - time
before an over-flowrate shuts down the system
    Public Shared FaultTimeP As ULong = My.Settings.HighPressureTime * 1000 '[ms] - time
before an over-pressure shuts down the system
    Public Shared FilterArea As Double = My.Settings.MembraneArea '[m^2] - area of single
membrane
    Public Shared Flowmeters As Double = My.Settings.Flowmeters '[] - number of parallel
flowmeters


    '*** DOI PARAMETERS ***
    Public Shared p_psi() As Double = {My.Settings.DOIPP0, My.Settings.DOIPP1,
My.Settings.DOIPP2} '[psi] - calibration pressures for pressure transducer
    Public Shared p_sig() As Double = {My.Settings.DOIPC0, My.Settings.DOIPC1,
My.Settings.DOIPC2} '[mA] - calibration signal for pressure transducer
    Public Shared p_resistor As Double = My.Settings.DOIPR '[Ohm] - resistance between
pressure transducer terminals
    Dim BackwashFrequency As ULong '[ms] - time between backwashses
    Dim BackwashDuration As ULong '[ms] - duration of a single backwash
    Public Shared MembraneCountDOI As Double = My.Settings.DOIMembraneCount 'number of
membranes in UF module
    Dim DOIPressureMax As Double '[psi] - "do not exceed" pressure
    Dim DOIFluxTarget As Double '[LMH] - flux match setting
    Dim DOIPressureTarget As Double '[psi] - pressure match setting
    Dim StatusColor As Color = Color.Lime
    Public Shared minPumpFreq As Double = My.Settings.MinPump '[Hz] - minimum pump speed
before control is given back to valve change
    Public Shared maxPumpFreq As Double = My.Settings.MaxPump '[Hz] - maximum pump speed
```

```
    Dim DOI_HighPTimer As New Stopwatch 'times high pressure
    Dim DOI_HighFTimer As New Stopwatch 'times high flow
    Dim DOI_FaultCondition As Boolean 'has an error occured?
    Dim A_Flat As Double = 0.007 * (9 * 0.0254) '[m^2] - flow area inside flat sheet
module
    Dim A_Hollow As Double = Math.PI * ((1.522 * 0.0254) / 2) ^ 2 - Math.PI * ((1 *
0.0254) / 2) ^ 2 '[m^2] - flow area inside hollow fiber module
    Dim A_Spiral As Double = 0.001393 '[m^2] - flow area inside spiral wound module

    '*** AF PARAMETERS ***
    Public Shared pUF_psi() As Double = {My.Settings.AFUFPP0, My.Settings.AFUFPP1,
My.Settings.AFUFPP2} '[psi] - calibration pressures for UF pressure transducer
    Public Shared pUF_sig() As Double = {My.Settings.AFUFPC0, My.Settings.AFUFPC1,
My.Settings.AFUFPC2} '[mA] - calibration signal for UF pressure transducer
    Public Shared pUF_resistor As Double = My.Settings.AFUFPR '[Ohm] - resistance between
UF pressure transducer terminals
    Public Shared pNF_psi() As Double = {My.Settings.AFNFPP0, My.Settings.AFNFPP1,
My.Settings.AFNFPP2} '[psi] - calibration pressures for NF pressure transducer
    Public Shared pNF_sig() As Double = {My.Settings.AFNFPC0, My.Settings.AFNFPC1,
My.Settings.AFNFPC2} '[mA] - calibration signal for NF pressure transducer
    Public Shared pNF_resistor As Double = My.Settings.AFNFPR '[Ohm] - resistance between
NF pressure transducer terminals
    Public Shared Lev_depth() As Double = {My.Settings.AFDD0, My.Settings.AFDD1,
My.Settings.AFDD2} '[ft] - calibration depths for depth transducer
    Public Shared Lev_sig() As Double = {My.Settings.AFDC0, My.Settings.AFDC1,
My.Settings.AFDC2} '[mA] - calibration signal for depth transducer
    Public Shared Lev_resistor As Double = My.Settings.AFDR '[Ohm] - resistance between
depth transducer terminals
```

```
    Public Shared maxdepth As Double = My.Settings.MaxDepth / 12 '[ft] - UF feed full
depth
    Public Shared mindepth As Double = My.Settings.MinDepth / 12 '[ft] - UF feed empty
depth (10%)
    Dim BackwashFrequencyAF As ULong '[ms] - time between backwashses
    Dim BackwashDurationAF As ULong '[ms] - duration of a single backwash
    Dim CleanFrequency As ULong '[ms] - time between cleanings
    Dim CleanDuration As ULong '[ms] - duration of a single cleaning
    Dim NFCleanFrequency As ULong '[ms] - time between NF cleanings
    Dim NFCleanDuration As ULong '[ms] - duration of a single NF cleaning
    Dim ValveTime As ULong = 90 * 1000 '[ms] - time for valve to move full range
    Public Shared MembraneCountAF As Double = My.Settings.AFMembraneCount 'number of
membranes on each side of UF/NF module
    Dim AFPressureMaxUF As Double '[psi] - "do not exceed" pressure
    Dim AFPressureMaxNF As Double '[psi] - "do not exceed" pressure
    Dim AFFluxTargetHigh As Double '[LMH] - flux setting when full
    Dim AFFluxTargetLow As Double '[LMH] - flux setting when empty
    Dim AFPressureTarget As Double '[psi] - pressure matching setting
    Dim AF_UF_HighPTimer As New Stopwatch 'times high UF pressure
    Dim AF_NF_HighPTimer As New Stopwatch 'times high NF pressure
    Dim AF_UF_HighFTimer As New Stopwatch 'times high UF flowrate
    Dim AF_NF_HighFTimer As New Stopwatch 'times high NF flowrate
    Dim AF_UF_FaultCondition As Boolean 'has a UF error occured?
    Dim AF_NF_FaultCondition As Boolean 'has a NF error occured?
    Dim isPressureAdjusting As Boolean 'is pressure actively adjusting?
    Dim isFluxAdjusting As Boolean 'is flux actively adjusting?
    Public Shared SettingRange As Double = My.Settings.Accuracy / 100 'acceptable range
for set point
```

```
#End Region

#Region "Main GUI Form Changes"

    'DOI
    Dim chkMainPump_isChanged As Boolean
    Dim chkMainPump_isChecked As Boolean
    Dim chkHSMixer_isChanged As Boolean
    Dim chkSeparator_isChanged As Boolean
    Dim chkLSMixer_isChanged As Boolean
    Dim chkBackwashPump_isChanged As Boolean
    Dim chkRinse_isChanged As Boolean
    Dim numPumpFreq_isChanged As Boolean
    Dim numPumpFreq_Value As Double
    Dim numHSMixerFreq_isChanged As Boolean
    Dim numSeparatorFreq_isChanged As Boolean
    Dim useBackwashDOI As Boolean
    Dim chkPressure_isChecked As Boolean
    Dim chkFlow_isChecked As Boolean
    Dim numDOI_isChanged As Boolean
    Dim numDOI_Value As Double
    Dim DOI_High_Count As Integer

    'AF
    Dim chkPumpPrefilter_isChanged As Boolean
    Dim chkPumpUFMain_isChanged As Boolean
    Dim chkPumpUFMain_isChecked As Boolean
    Dim chkPumpUFBackwash_isChanged As Boolean
    Dim chkPumpNFMain_isChanged As Boolean
```

```
    Dim chkPumpNFMain_isChecked As Boolean
    Dim chkSelectorUFRecirculate_isChanged As Boolean
    Dim chkSelectorUFFiltrate_isChanged As Boolean
    Dim chkSelectorNFRecirculate_isChanged As Boolean
    Dim numUF_isChanged As Boolean
    Dim numUF_Value As Double
    Dim numNF_isChanged As Boolean
    Dim numNF_Value As Double
    Dim useBackwashAF As Boolean
    Dim useCleaning As Boolean
    Dim useNFCleaning As Boolean
    Dim headStart As Boolean
    Dim AutoRestartNF As Boolean
    Dim chkUFPressure_isChecked As Boolean
    Dim chkUFFlow_isChecked As Boolean
    Dim chkNFPressure_isChecked As Boolean
    Dim chkNFFlow_isChecked As Boolean
    Dim chkNFFeedFull_isChecked As Boolean
    Dim chkNFFeedEmpty_isChecked As Boolean
    Dim chkDepth_isChecked As Boolean

#End Region

#Region "Sensor Values"

    'DOI
    Dim Flowrate(3) As Double
    Dim Pressure(3) As Double
```

```
    'AF
    Dim UFFlowrate(3) As Double
    Dim UFPressure(3) As Double
    Dim NFFlowrate(3) As Double
    Dim NFPressure(3) As Double
    Dim Depth(3) As Double
    Dim isNFFull(3) As Boolean
    Dim isNFEmpty(3) As Boolean

#End Region

#Region "Pin Assignments"

    'DOI
    Dim pinFlowrate As Byte = 13 'digitalRead - Flow
    Dim pinPressure As Byte = 1 'analogRead - Pressure
    Dim pinSeparator As Byte = 2 'digitalWrite - VFD A
    Dim pinSeparatorAdjust As Byte = 3 'analogWrite - VFD A'
    Dim pinHSMix As Byte = 4 'digitalWrite - VFD B
    Dim pinHSMixerAdjust As Byte = 5 'analogWrite - VFD B'
    Dim pinBackwash As Byte = 12 'digitalWrite - Relay A
    Dim pinWash As Byte = 11 'digitalWrite - VFD D'
    Dim pinLSMix As Byte = 8 'digitalWrite - Relay B
    Dim pinPump As Byte = 7 'digitalWrite - VFD C
    Dim pinPumpAdjust As Byte = 6 'analogWrite - VFD C'
    Dim pinDOIPressureValve As Byte = 10 'analogWrite - Valve

    'AF
    Dim pinNFFull As Byte = 22 '(switch C) - digitalRead
```

```
    Dim pinNFEmpty As Byte = 24 '(switch D) - digitalRead
    Dim pinUFPressure As Byte = 9 'analog pin (pressure B) - analogRead
    Dim pinNFPressure As Byte = 15 'analog pin (pressure C) - analogRead
    Dim pinDepth As Byte = 4 'analog pin (pressure A) - analogRead
    Dim pinUFFlowrate As Byte = 14 '(flow A) - digitalRead
    Dim pinNFFlowrate As Byte = 15 '(flow B) - digitalRead
    Dim pinPrefilterPump As Byte = 52 '(relay A) - digitalWrite
    Dim pinUFMainPump As Byte = 53 '(relay B) - digitalWrite
    Dim pinUFBackwashPump As Byte = 49 '(relay C) - digitalWrite
    Dim pinNFMainPump As Byte = 41 '(relay D) - digitalWrite
    Dim pinUFRecirculateValve As Byte = 18 '(valve D) - digitalWrite
    Dim pinUFFiltrateValve As Byte = 19 '(valve C) - digitalWrite
    Dim pinNFRecirculateValve As Byte = 17 '(valve E) - digitalWrite
    Dim pinUFPressureValve As Byte = 7 '(valve A) - analogWrite
    Dim pinNFPressureValve As Byte = 4 '(valve B) - analogWrite

#End Region

#Region "Data Output"

    'creates unique output directory
    Private Sub CreateDirectory()

        'create header text
        Dim strID As String
        If isDOI Then
            If isSimulation Then
                strID = "DOI_Sim"
            Else
```

```
                strID = "DOI"
            End If
        Else
            If isSimulation Then
                strID = "AF_Sim"
            Else
                strID = "AF"
            End If
        End If

        'create unique output directory
        Dim folder As String = My.Computer.FileSystem.SpecialDirectories.Desktop & "\" &
strID & "_Data"
        If Not My.Computer.FileSystem.DirectoryExists(folder) Then
            My.Computer.FileSystem.CreateDirectory(folder)
        End If
        Dim filename As String = "\" & System.DateTime.Today.Year.ToString("0000") & "_"
& System.DateTime.Today.Month.ToString("00") & "_" & DateTime.Today.Day.ToString("00") &
"_00.csv"
        Dim count As Integer = 0
        While My.Computer.FileSystem.FileExists(folder & filename)
            count += 1
            filename = "\" & System.DateTime.Today.Year.ToString("0000") & "_" &
System.DateTime.Today.Month.ToString("00") & "_" & DateTime.Today.Day.ToString("00") &
"_" & count.ToString("00") & ".csv"
        End While
        filepath = folder & filename

        'create file with header
```

```
    RecordHeader()

End Sub

'record state and sensor headings
Private Sub RecordHeader()

    Dim dataline As String = "Time [3]" '0

    'add data
    If isDOI Then '15 headers
        dataline += "," & "Flowrate [LPM]" '1
        dataline += "," & "Pressure [psi]" '2
        dataline += "," & "is Main Pump On?" '3
        dataline += "," & "Main Pump Frequency" '4
        dataline += "," & "is HS Mixer On?" '5
        dataline += "," & "HS Mixer Frequency" '6
        dataline += "," & "is Separator On?" '7
        dataline += "," & "Separator Frequency" '8
        dataline += "," & "is Separator Wash On?" '9
        dataline += "," & "is LS Mixer On?" '10
        dataline += "," & "is Backwash Pump On?" '11
        dataline += "," & "Pressure Valve position [% open]" '12
    Else '19 headers
        dataline += "," & "UF Flowrate [LPM]" '1
        dataline += "," & "NF Flowrate [LPM]" '2
        dataline += "," & "UF Pressure [psi]" '3
        dataline += "," & "NF Pressure [psi]" '4
        dataline += "," & "UF Feed Depth [ft]" '5
```

```
            dataline += "," & "is NF Feed Full?" '6
            dataline += "," & "is NF Feed Empty?" '7
            dataline += "," & "is Prefilter Pump On?" '8
            dataline += "," & "is Main UF Pump On?" '9
            dataline += "," & "is UF Backwash Pump On" '10
            dataline += "," & "is Main NF Pump On?" '11
            dataline += "," & "is UF waste open?" '12
            dataline += "," & "is UF cleaner open?" '13
            dataline += "," & "is NF waste open?" '14
            dataline += "," & "UF Pressure Valve position [% open]" '15
            dataline += "," & "NF Pressure Valve position [% open]" '16
        End If

        'add version (annotation header)
        dataline += "," & "Annotations_v" & getVersion() '13/17

        'add total membrane area
        If isDOI Then
            dataline += "," & (FilterArea * MembraneCountDOI).ToString '14
        Else
            dataline += "," & (FilterArea * MembraneCountAF).ToString '18
        End If
        dataline += vbCrLf

        'create file
        My.Computer.FileSystem.WriteAllText(filepath, dataline, True)

    End Sub
```

```
'record state and sensor data to file
Private Sub RecordData()

    'time
    Dim dataline As String = getHours().ToString("0.000000")

    'record current sensor values
    If isDOI Then
        dataline += "," & Flowrate(0).ToString("0.000")
        dataline += "," & Pressure(0).ToString("0.00")
        dataline += "," & chkMainPump.Checked.ToString
        dataline += "," & numMainPump.Value.ToString("0.00")
        dataline += "," & chkHSMixer.Checked.ToString
        dataline += "," & numHSMixer.Value.ToString("0.00")
        dataline += "," & chkSeparator.Checked.ToString
        dataline += "," & numSeparator.Value.ToString("0.00")
        dataline += "," & chkRinse.Checked.ToString
        dataline += "," & chkLSMixer.Checked.ToString
        dataline += "," & chkBackwash.Checked.ToString
        dataline += "," & numDOI.Value.ToString("0.00")
    Else
        dataline += "," & UFFlowrate(0).ToString("0.000")
        dataline += "," & NFFlowrate(0).ToString("0.000")
        dataline += "," & UFPressure(0).ToString("0.00")
        dataline += "," & NFPressure(0).ToString("0.00")
        dataline += "," & Depth(0).ToString("0.00")
        dataline += "," & isNFFull(0).ToString
        dataline += "," & isNFEmpty(0).ToString
        dataline += "," & chkPumpPrefilter.Checked.ToString
```

```
            dataline += "," & chkPumpUFMain.Checked.ToString
            dataline += "," & chkPumpBackwash.Checked.ToString
            dataline += "," & chkPumpNFMain.Checked.ToString
            dataline += "," & chkSelectorUFRecirculate.Checked.ToString
            dataline += "," & chkSelectorUFFiltrate.Checked.ToString
            dataline += "," & chkSelectorNFRecirculate.Checked.ToString
            dataline += "," & numUF.Value.ToString("0.00")
            dataline += "," & numNF.Value.ToString("0.00")
        End If

        'record current annotation (if pending)
        If isAnnotationPending Then
            dataline += "," & AnnotationText
            isAnnotationPending = False
        Else
            dataline += ","
        End If

        dataline += vbCrLf

        'append data to file
        Try
            My.Computer.FileSystem.WriteAllText(filepath, dataline, True)
        Catch ex As Exception 'file not accessible
            CreateDirectory()
            My.Computer.FileSystem.WriteAllText(filepath, dataline, True)
        End Try

    End Sub
```

```
#End Region

#Region "Delegates (keeps processing threads separate)"

    Delegate Sub SetButtonTextCallback(ByVal btn As Button, ByVal txt As String)
    Private Sub SetButtonText(ByVal btn As Button, ByVal txt As String)
        If btn.InvokeRequired Then
            Dim d As New SetButtonTextCallback(AddressOf ButtonTextSet)
            Invoke(d, New Object() {btn, txt})
        Else
            btn.Text = txt
        End If
    End Sub
    Private Sub ButtonTextSet(ByVal btn As Button, txt As String)
        btn.Text = txt
    End Sub

    Delegate Sub SetButtonStateCallback(ByVal btn As Button, ByVal state As Boolean)
    Private Sub SetButtonState(ByVal btn As Button, ByVal state As Boolean)
        If btn.InvokeRequired Then
            Dim d As New SetButtonStateCallback(AddressOf ButtonStateSet)
            Invoke(d, New Object() {btn, state})
        Else
            btn.Enabled = state
        End If
    End Sub
    Private Sub ButtonStateSet(ByVal btn As Button, state As Boolean)
        btn.Enabled = state
```

```
End Sub

Delegate Sub SetTextBoxCallback(ByVal tb As TextBox, ByVal text As String)
Private Sub SetTextBox(ByVal tb As TextBox, ByVal text As String)
    If tb.InvokeRequired Then
        Dim d As New SetTextBoxCallback(AddressOf TextBoxSet)
        Try
            Invoke(d, New Object() {tb, text})
        Catch ex As Exception
            'do nothing (form is closing)
        End Try
    Else
        tb.Text = text
    End If
End Sub
Private Sub TextBoxSet(ByVal tb As TextBox, text As String)
    tb.Text = text
End Sub

Delegate Sub SetCheckBoxCallback(ByVal cb As CheckBox, ByVal state As Boolean)
Private Sub SetCheckBox(ByVal cb As CheckBox, ByVal state As Boolean)
    If cb.InvokeRequired Then
        Dim d As New SetCheckBoxCallback(AddressOf CheckBoxSet)
        Invoke(d, New Object() {cb, state})
    Else
        cb.Checked = state
    End If
End Sub
Private Sub CheckBoxSet(ByVal cb As CheckBox, ByVal state As Boolean)
```

```
    cb.Checked = state
End Sub

Delegate Sub SetGroupBoxCallback(ByVal gb As GroupBox, ByVal state As Boolean)
Private Sub SetGroupBox(ByVal gb As GroupBox, ByVal state As Boolean)
    If gb.InvokeRequired Then
        Dim d As New SetGroupBoxCallback(AddressOf GroupBoxSet)
        Invoke(d, New Object() {gb, state})
    Else
        gb.Enabled = state
    End If
End Sub
Private Sub GroupBoxSet(ByVal gb As GroupBox, ByVal state As Boolean)
    gb.Enabled = state
End Sub

Delegate Sub SetLabelCallBack(ByVal l As Label, ByVal text As String)
Private Sub SetLabel(ByVal l As Label, ByVal text As String)
    If l.InvokeRequired Then
        Dim d As New SetLabelCallBack(AddressOf LabelSet)
        Invoke(d, New Object() {l, text})
    Else
        l.Text = text
    End If
End Sub
Private Sub LabelSet(ByVal l As Label, ByVal text As String)
    l.Text = text
End Sub
```

```
    Delegate Sub SetProgressBarCallback(ByVal pb As ProgressBar, ByVal value As UInteger)
    Private Sub SetProgressBar(ByVal pb As ProgressBar, ByVal value As UInteger)
        If pb.InvokeRequired Then
            Try
                Dim d As New SetProgressBarCallback(AddressOf ProgressBarSet)
                Invoke(d, New Object() {pb, value})
            Catch ex As Exception
                'do nothing
            End Try
        Else
            pb.Value = value
        End If
    End Sub
    Private Sub ProgressBarSet(ByVal pb As ProgressBar, ByVal value As UInteger)
        pb.Value = value
    End Sub
    Private Sub SetProgressBarSafe(ByVal pb As ProgressBar, ByVal value As Double)
        If value >= 0 And value <= 100 Then
            SetProgressBar(pb, CUInt(value))
        End If
    End Sub

    Delegate Sub SetProgressBarVisibleCallback(ByVal pb As ProgressBar, ByVal state As
Boolean)
    Private Sub SetProgressBarVisible(ByVal pb As ProgressBar, ByVal state As Boolean)
        If pb.InvokeRequired Then
            Dim d As New SetProgressBarVisibleCallback(AddressOf ProgressBarVisibleSet)
            Invoke(d, New Object() {pb, state})
        Else
```

```
            pb.Visible = state
        End If
    End Sub
    Private Sub ProgressBarVisibleSet(ByVal pb As ProgressBar, ByVal state As Boolean)
        pb.Visible = state
    End Sub

    Delegate Sub SetPanel2CollapsedCallback(ByVal sc As SplitContainer, ByVal state As
Boolean)
    Private Sub SetPanel2Collapsed(ByVal sc As SplitContainer, ByVal state As Boolean)
        If sc.InvokeRequired Then
            Dim d As New SetPanel2CollapsedCallback(AddressOf Panel2CollapsedSet)
            Invoke(d, New Object() {sc, state})
        Else
            sc.Panel2Collapsed = state
        End If
    End Sub
    Private Sub Panel2CollapsedSet(ByVal sc As SplitContainer, ByVal state As Boolean)
        sc.Panel2Collapsed = state
    End Sub

    Delegate Sub UpdateChartCallback(ByVal [pinID] As Integer, ByVal [val] As Double)
    Private Sub UpdateChart(ByVal [pinID] As Integer, ByVal [val] As Double)
        Try
            If Chart1.InvokeRequired Then
                Dim d As New UpdateChartCallback(AddressOf UpdateChart)
                Invoke(d, New Object() {[pinID], [val]})
            Else
                Dim seriesID As Integer = PinToSeries(pinID)
```

```
                Chart1.Series(seriesID).Points.AddXY(getElapsedHours(), [val])
                Chart1.Update()
            End If
        Catch ex As Exception
            'do nothing
        End Try
    End Sub

    Delegate Sub SetNumericCallback(ByVal num As NumericUpDown, ByVal value As Decimal)
    Private Sub SetNumeric(ByVal num As NumericUpDown, ByVal value As Decimal)
        If num.InvokeRequired Then
            Dim d As New SetNumericCallback(AddressOf NumericSet)
            Invoke(d, New Object() {num, value})
        Else
            num.Value = value
        End If
    End Sub
    Private Sub NumericSet(ByVal num As NumericUpDown, ByVal value As Decimal)
        num.Value = value
    End Sub

    Delegate Sub SetToolStripLabelCallback(ByVal lbl As ToolStripLabel, ByVal txt As
String)
    Private Sub SetToolStripLabel(ByVal lbl As ToolStripLabel, ByVal txt As String)
        Try
            If StatusStrip1.InvokeRequired Then
                Dim d As New SetToolStripLabelCallback(AddressOf ToolStripLabelSet)
                Invoke(d, New Object() {lbl, txt})
            Else
```

```
            lbl.Text = txt
        End If
    Catch ex As Exception
        'TODO figure out why this fails during form shut down
    End Try

End Sub
Private Sub ToolStripLabelSet(ByVal lbl As ToolStripLabel, ByVal txt As String)
    lbl.Text = txt
End Sub

Delegate Sub TogglePictureBoxColorCallback(ByVal pic As PictureBox)
Private Sub TogglePictureBoxColor(ByVal pic As PictureBox)
    If pic.InvokeRequired Then
        Dim d As New TogglePictureBoxColorCallback(AddressOf PictureBoxColorToggle)
        Invoke(d, New Object() {pic})
    Else
        If pic.BackColor = StatusColor Then
            pic.BackColor = Color.White
        Else
            pic.BackColor = StatusColor
        End If
    End If
End Sub
Private Sub PictureBoxColorToggle(ByVal pic As PictureBox)
    If pic.BackColor = StatusColor Then
        pic.BackColor = Color.White
    Else
        pic.BackColor = StatusColor
```

```
        End If
    End Sub

#End Region

#Region "Arduino I/O"

    'add checksum and write data - return sensor output
    Private Function SerialWrite(command As Byte, pin As Byte, value As Byte) As UInt16

        If isSimulation Then 'simulate sensor data (for testing only)
            Dim temp As Double = 1 + (0.05 * (Rnd() - 0.5))
            If isDOI Then
                Select Case pin
                    Case pinPressure
                        If chkMainPump_isChecked Then
                            Dim d2 As Double = ((800 * (1 - (numDOI_Value / 100))) + 199)
* temp
                            Sleep(10)
                            Return CUInt(d2)
                        Else
                            Return CUInt(197 * temp)
                        End If
                    Case pinFlowrate
                        Sleep(1000)
                        Dim d3 As Double = (Pressure(0) / 20) * temp
                        If d3 < 0.1 Then
                            Return 0
                        Else
```

```
                            Return CUInt(22000 * (d3 / 60))
                        End If
                    Case Else
                        Return value
                End Select
            Else
                Select Case pin
                    Case pinUFPressure
                        If chkPumpUFMain_isChecked Then
                            Dim d2 As Double = ((800 * (1 - (numUF_Value / 100))) + 199)
* temp
                            Sleep(10)
                            Return CUInt(d2)
                        Else
                            Return CUInt(199 * temp)
                        End If
                    Case pinUFFlowrate
                        Sleep(1000)
                        Dim d3 As Double = (UFPressure(0) / 20) * temp
                        If d3 < 0.1 Then
                            Return 0
                        Else
                            Return CUInt(22000 * (d3 / 60))
                        End If
                    Case pinNFPressure 'identical to pinNFFlowrate
                        Select Case command
                            Case 3 'pressure - NFPressure
                                If chkPumpNFMain_isChecked Then
```

```
                                    Dim d2 As Double = ((800 * (1 - (numNF_Value / 100)))
+ 199) * temp

                                    Sleep(10)

                                    Return CUInt(d2)

                                Else

                                    Return CUInt(199 * temp)

                                End If

                            Case 6 'flow - NFFlowrate

                                Sleep(1000)

                                Dim d3 As Double = (NFPressure(0) / 200) * temp

                                If d3 < 0.1 Then

                                    Return 0

                                Else

                                    Return CUInt(22000 * (d3 / 60))

                                End If

                        End Select

                    Case pinDepth

                        Dim d2 As Double = ((1023 - 199) * 3 / 5) + 199 'TODO - simulate
depth

                        Sleep(10)

                        Return CUInt(d2)

                    Case pinNFEmpty

                        Return 0 'TODO - simulate NF empty

                    Case pinNFFull

                        Return 0 'TODO - simulate NF full

                    Case Else

                        Return value

                End Select

            End If
```

```
            Return value
        End If

        'create a data buffer
        Dim DataOut() As Byte = {command, pin, value, 0} 'command, pin, data, checksum
        DataOut(3) = getCheckSum(command, pin, value)

        'write the data
        Try
            myPort.Write(DataOut, 0, 4) 'write the four bytes of data
        Catch ex As Exception
            MessageBox.Show(ex.Message, "Error", MessageBoxButtons.OK,
MessageBoxIcon.Error)
            Return 0
        End Try

        'get the response
        While True
            If myPort.BytesToRead >= 5 Then
                'get data from buffer
                Dim DataIn() As Byte = {0, 0, 0, 0, 0}
                myPort.Read(DataIn, 0, 5)
                If isValid(DataIn) And DataIn(0) = DataOut(0) And DataIn(1) = DataOut(1)
Then
                    Return BitConverter.ToUInt16(DataIn, 2)
                Else
                    'try retransmitting (repeat until receipt confirmation)
                    myPort.DiscardOutBuffer()
                    While myPort.BytesToRead > 0
```

```
                    myPort.DiscardInBuffer()
                    Sleep(1)
                End While
                Try
                    myPort.Write(DataOut, 0, 4)
                    Sleep(1)
                Catch ex As Exception
                    'do nothing
                End Try
            End If
        Else
            Sleep(1)
        End If
    End While

    Return 0

End Function

'generate checksum for data set
Private Function getCheckSum(command As Byte, pin As Byte, value As Byte) As Byte
    Return CByte(Math.Ceiling((255.0 + command + pin + value) / 4.0))
End Function

'checks generated checksum against incoming data to ensure data fidelity
Private Function isValid(data() As Byte) As Boolean
    Dim checksum As Byte = CByte(Math.Ceiling((255.0 + data(0) + data(1) + data(2) +
data(3)) / 5))
    If data(4) = checksum Then
```

```
        Return True
    Else
        Return False
    End If
End Function

'convert analog sensor value to pressure or depth
Private Function SensorToValue(pin As Integer, val As Double) As Double
    'note: y = a0 * a1*x + a2*x^2
    Select Case pin
        Case pinPressure
            Return poly(0) + poly(1) * val + poly(2) * val * val
        Case pinUFPressure
            Return poly_UF(0) + poly_UF(1) * val + poly_UF(2) * val * val
        Case pinNFPressure
            Return poly_NF(0) + poly_NF(1) * val + poly_NF(2) * val * val
        Case pinDepth
            Return poly_Lev(0) + poly_Lev(1) * val + poly_Lev(2) * val * val
        Case Else
            Return -1
    End Select
End Function

'get the state of a pin (HIGH or LOW (closed or open))
Private Function GetSwitch(pin As Byte) As Boolean

    Dim data As UInt16 = SerialWrite(1, pin, 0)
    If data = 1 Then
        Return True
```

```
        Else
            Return False
        End If

    End Function

    'set the state of a pin to HIGH or LOW (on or off)
    Private Sub SetState(pin As Byte, state As Boolean)
        If state Then
            SerialWrite(2, pin, 1)
        Else
            SerialWrite(2, pin, 0)
        End If
    End Sub

    'set the PWM output of a pin to the specified value - Frequency (max 60)
    Private Sub SetSpeed(pin As Byte, value As Double)
        Dim val As Byte = CByte(value * (255.0 / 60))
        SerialWrite(4, pin, val)
    End Sub

    'set the PWM output of a pin to the specified value - Valve Position percentage
    Private Sub SetPosition(pin As Byte, value As Double)
        Dim val As Byte = CByte((100.0 - value) * 2.55)
        SerialWrite(4, pin, val)
    End Sub

#End Region
```

```
#Region "Time Management"

    'returns the number of hours since 12:00:00 AM Jan 1, 2001 (excluding leapseconds)
    Private Function getHours() As Decimal
        Return (Date.Now.Ticks - #1/1/2001#.Ticks) / 36000000000.0
    End Function

    'returns the number of hours since the start of the program (or start of run)
    Private Function getElapsedHours() As Decimal
        Return getHours() - starthour
    End Function

#End Region

#Region "Background Workers"

    'act of GUI states
    Private Sub ArduinoUpdater_DoWork(sender As Object, e As
System.ComponentModel.DoWorkEventArgs) Handles ArduinoUpdater.DoWork

        'run indefinitely
        While True

            'check for pending cancellation (during program close)
            If ArduinoUpdater.CancellationPending Then
                e.Cancel = True
                Exit While
            End If
```

```
            'update
            If Not isPaused Then
                UpdateForm()
            Else
                SerialWrite(1, 9, 0) 'send signal -- still connected
                SetToolStripLabel(tsslInfo, "System Paused")
                Sleep(1000) 'delay 1 second
            End If

        End While

    End Sub

    'record data to file
    Private Sub FileWriter_DoWork(sender As Object, e As
System.ComponentModel.DoWorkEventArgs) Handles FileWriter.DoWork
        Try
            RecordData()
        Catch ex As Exception
            'do nothing
        End Try
    End Sub

    'turn system off
    Private Sub ProcessWorker_RunWorkerCompleted(sender As Object, e As
System.ComponentModel.RunWorkerCompletedEventArgs) Handles
ProcessWorker.RunWorkerCompleted
        SetButtonState(btnDOIFluxHold, True)
        SetButtonState(btnDOIFluxHoldCancel, False)
```

```
        SetButtonState(btnDOIPressureHold, True)
        SetButtonState(btnDOIPressureHoldCancel, False)
        SetButtonState(btnAFFluxHold, True)
        SetButtonState(btnCancelUF, False)
        isPressureHold = False
    End Sub

    'run DOI/AF UF routine
    Private Sub ProcessWorker_DoWork(sender As Object, e As
System.ComponentModel.DoWorkEventArgs) Handles ProcessWorker.DoWork
        RunUF()
    End Sub

    'run AF prefilter routine
    Private Sub PrefilterWorker_DoWork(sender As Object, e As
System.ComponentModel.DoWorkEventArgs) Handles PrefilterWorker.DoWork
        RunPrefilter()
    End Sub

    'turn off AF prefiltration
    Private Sub PrefilterWorker_RunWorkerCompleted(sender As Object, e As
System.ComponentModel.RunWorkerCompletedEventArgs) Handles
PrefilterWorker.RunWorkerCompleted
        SetButtonState(btnAFPrefilter, True)
        SetButtonState(btnCancelPrefiltration, False)
    End Sub

    'run AF NF routine
```

```
    Private Sub NFWorker_DoWork(sender As Object, e As
System.ComponentModel.DoWorkEventArgs) Handles NFWorker.DoWork
        RunNF()
    End Sub

    'turn off AF NF routine
    Private Sub NFWorker_RunWorkerCompleted(sender As Object, e As
System.ComponentModel.RunWorkerCompletedEventArgs) Handles NFWorker.RunWorkerCompleted
        SetButtonState(btnAFPressureHold, True)
        SetButtonState(btnCancelNF, False)
    End Sub

#End Region

#Region "Form Updates"

    'checks the states of all form controls and sensors, and gets/sends updates to/from
Arduino
    Private Sub UpdateForm()

        Dim record As Boolean
        Dim info As String = ""
        Dim sw As New Stopwatch
        sw.Start()

        If isDOI Then

            'pressure control valve
            If numDOI_isChanged Then
```

```
        SetPosition(pinDOIPressureValve, numDOI_Value)
        numDOI_isChanged = False
        record = True
    End If

    'sensors
    If chkPressure_isChecked Then
        GetPressure(Pressure, pinPressure)
        info += "|  Pressure [psi]: " & Pressure(0).ToString("0.00") & "  |"
        record = True
    End If
    If chkFlow_isChecked Then
        GetFlowrate(Flowrate, pinFlowrate)
        info += "|  Flow rate [L/min]: " & Flowrate(0).ToString("0.000") & "  |"
        record = True
    End If

    'check for high pressure
    If Pressure(0) > DOIPressureMax Then
        If Not DOI_HighPTimer.IsRunning Then
            DOI_HighPTimer.Start()
        Else
            If DOI_HighPTimer.ElapsedMilliseconds > FaultTimeP Then
                SetCheckBox(chkMainPump, False)
                DOI_FaultCondition = True
            End If
        End If
    Else
        DOI_HighPTimer.Reset()
```

```
End If

'check for high flow
If Flowrate(0) > MaxFlow Then
    If Not DOI_HighFTimer.IsRunning Then
        DOI_HighFTimer.Start()
    Else
        If DOI_HighFTimer.ElapsedMilliseconds > FaultTimeF Then
            SetCheckBox(chkMainPump, False)
            DOI_FaultCondition = True
        End If
    End If
Else
    DOI_HighFTimer.Reset()
End If

'devices
If chkMainPump_isChanged Then
    SetState(pinPump, chkMainPump.Checked)
    chkMainPump_isChanged = False
    record = True
    numPumpFreq_isChanged = True
End If
If numPumpFreq_isChanged Then
    If chkMainPump.Checked Then
        SetSpeed(pinPumpAdjust, numMainPump.Value)
        numPumpFreq_isChanged = False
        record = True
    Else
```

```
            SetSpeed(pinPumpAdjust, 0)
            numPumpFreq_isChanged = False
            record = True
        End If
    End If
    If chkHSMixer_isChanged Then
        SetState(pinHSMix, chkHSMixer.Checked)
        chkHSMixer_isChanged = False
        record = True
    End If
    If numHSMixerFreq_isChanged Then
        SetSpeed(pinHSMixerAdjust, numHSMixer.Value)
        numHSMixerFreq_isChanged = False
        record = True
    End If
    If chkSeparator_isChanged Then
        SetState(pinSeparator, chkSeparator.Checked)
        chkSeparator_isChanged = False
        record = True
    End If
    If numSeparatorFreq_isChanged Then
        SetSpeed(pinSeparatorAdjust, numSeparator.Value)
        numSeparatorFreq_isChanged = False
        record = True
    End If
    If chkRinse_isChanged Then
        SetState(pinWash, chkRinse.Checked)
        chkRinse_isChanged = False
        record = True
```

```
    End If
    If chkLSMixer_isChanged Then
        SetState(pinLSMix, chkLSMixer.Checked)
        chkLSMixer_isChanged = False
        record = True
    End If
    If chkBackwashPump_isChanged Then
        SetState(pinBackwash, chkBackwash.Checked)
        chkBackwashPump_isChanged = False
        record = True
    End If

Else

    'pressure control valves
    If numUF_isChanged Then
        SetPosition(pinUFPressureValve, numUF_Value)
        numUF_isChanged = False
        record = True
    End If
    If numNF_isChanged Then
        SetPosition(pinNFPressureValve, numNF_Value)
        numNF_isChanged = False
        record = True
    End If

    'sensors
    If chkUFPressure_isChecked Then
        GetPressure(UFPressure, pinUFPressure)
```

```
        info += "|  UF P. [psi]: " & UFPressure(0).ToString("0.00") & "  |"
        record = True
    End If
    If chkUFFlow_isChecked Then
        GetFlowrate(UFFlowrate, pinUFFlowrate)
        info += "|  UF Flow [L/min]: " & UFFlowrate(0).ToString("0.000") & "  |"
        record = True
    End If
    If chkNFPressure_isChecked Then
        GetPressure(NFPressure, pinNFPressure)
        info += "|  NF P. [psi]: " & NFPressure(0).ToString("0.00") & "  |"
        record = True
    End If
    If chkNFFlow_isChecked Then
        GetFlowrate(NFFlowrate, pinNFFlowrate)
        info += "|  NF Flow [L/min]: " & NFFlowrate(0).ToString("0.000") & "  |"
        record = True
    End If
    If chkDepth_isChecked Then
        GetPressure(Depth, pinDepth)
        info += "|  UF Depth [ft]: " & Depth(0).ToString("0.000") & "  |"
        record = True
    End If
    If chkNFFeedFull_isChecked Then
        GetState(isNFFull, pinNFFull)
        If isNFFull(0) Then
            info += "|  NF is full  |"
        Else
            info += "|  NF is not full  |"
```

```
    End If
    record = True
End If
If chkNFFeedEmpty_isChecked Then
    GetState(isNFEmpty, pinNFEmpty)
    If isNFEmpty(0) Then
        info += "|  NF is empty  |"
    Else
        info += "|  NF is not empty  |"
    End If
    record = True
End If

'check for high pressure (UF)
If UFPressure(0) > AFPressureMaxUF Then
    If Not AF_UF_HighPTimer.IsRunning Then
        AF_UF_HighPTimer.Start()
    Else
        If AF_UF_HighPTimer.ElapsedMilliseconds > FaultTimeP Then
            SetCheckBox(chkPumpUFMain, False)
            AF_UF_FaultCondition = True
        End If
    End If
Else
    AF_UF_HighPTimer.Reset()
End If

'check for high flow (UF)
If UFFlowrate(0) > MaxFlow Then
```

```
    If Not AF_UF_HighFTimer.IsRunning Then
        AF_UF_HighFTimer.Start()
    Else
        If AF_UF_HighFTimer.ElapsedMilliseconds > FaultTimeF Then
            SetCheckBox(chkPumpUFMain, False)
            AF_UF_FaultCondition = True
        End If
    End If
Else
    AF_UF_HighFTimer.Reset()
End If

'check for high pressure (NF)
If NFPressure(0) > AFPressureMaxNF Then
    If Not AF_NF_HighPTimer.IsRunning Then
        AF_NF_HighPTimer.Start()
    Else
        If AF_NF_HighPTimer.ElapsedMilliseconds > FaultTimeP Then
            SetCheckBox(chkPumpNFMain, False)
            AF_NF_FaultCondition = True
        End If
    End If
Else
    AF_NF_HighPTimer.Reset()
End If

'check for high flow (NF)
If NFFlowrate(0) > MaxFlow Then
    If Not AF_NF_HighFTimer.IsRunning Then
```

```
            AF_NF_HighFTimer.Start()
        Else
            If AF_NF_HighFTimer.ElapsedMilliseconds > FaultTimeF Then
                SetCheckBox(chkPumpNFMain, False)
                AF_NF_FaultCondition = True
            End If
        End If
    Else
        AF_NF_HighFTimer.Reset()
    End If

    'pumps
    If chkPumpPrefilter_isChanged Then
        SetState(pinPrefilterPump, chkPumpPrefilter.Checked)
        chkPumpPrefilter_isChanged = False
        record = True
    End If
    If chkPumpUFMain_isChanged Then
        SetState(pinUFMainPump, chkPumpUFMain.Checked)
        chkPumpUFMain_isChanged = False
        record = True
    End If
    If chkPumpUFBackwash_isChanged Then
        SetState(pinUFBackwashPump, chkPumpBackwash.Checked)
        chkPumpUFBackwash_isChanged = False
        record = True
    End If
    If chkPumpNFMain_isChanged Then
        SetState(pinNFMainPump, chkPumpNFMain.Checked)
```

```
            chkPumpNFMain_isChanged = False
            record = True
        End If

        'selector valves
        If chkSelectorUFRecirculate_isChanged Then
            SetState(pinUFRecirculateValve, chkSelectorUFRecirculate.Checked)
            chkSelectorUFRecirculate_isChanged = False
            record = True
        End If
        If chkSelectorUFFiltrate_isChanged Then
            SetState(pinUFFiltrateValve, chkSelectorUFFiltrate.Checked)
            chkSelectorUFFiltrate_isChanged = False
            record = True
        End If
        If chkSelectorNFRecirculate_isChanged Then
            SetState(pinNFRecirculateValve, chkSelectorNFRecirculate.Checked)
            chkSelectorNFRecirculate_isChanged = False
            record = True
        End If

    End If

    'annotations
    If isAnnotationPending Then
        record = True
    End If

    sw.Stop() 'stop the timer
```

```
    'data output
    If record Then

        'save to file
        If Not FileWriter.IsBusy() Then
            FileWriter.RunWorkerAsync()
        End If

        'write sensor data to screen
        SetToolStripLabel(tsslInfo, info)

        'toggle box
        TogglePictureBoxColor(PictureBox1)

    Else
        SerialWrite(1, 9, 0) 'send signal -- still connected
        Sleep(1000)
    End If

    SensorRespond = sw.ElapsedMilliseconds() 'update the update time
    If SensorRespond < 100 Then
        Sleep(100) 'prevent runaway (10 Hz max update rate)
    End If

End Sub

'set up a new chart for data plotting
Private Sub ChartSetup()
```

```
        'reset everything
        With Chart1

            'reset everything
            .Annotations.Clear()
            .ChartAreas.Clear()
            .Legends.Clear()
            .Series.Clear()
            .Titles.Clear()

            'define label font
            Dim f As New System.Drawing.Font("Arial", 12, FontStyle.Bold)

            If isDOI Then

                'create and set up chart areas
                .ChartAreas.Add("Pressure/Flux")
                With .ChartAreas(0)
                    .AxisY.Title = "Pressure [psi]"
                    .AxisY.LabelStyle.Format = "{0:0.##}"
                    .AxisY.TitleFont = f
                    .AxisY.LineWidth = 2
                    .AxisY.LabelStyle.Font = f
                    .AxisY2.Title = "Flux [LMH]"
                    .AxisY2.LabelStyle.Format = "{0:0.##}"
                    .AxisY2.MajorGrid.LineDashStyle =
DataVisualization.Charting.ChartDashStyle.Dash
                    .AxisY2.TitleFont = f
```

```
        .AxisY2.LineWidth = 2
        .AxisY2.LabelStyle.Font = f
        .AxisX.Title = "Elapsed [3]"
        .AxisX.LabelStyle.Format = "{0:0.###}"
        .AxisX.TitleFont = f
        .AxisX.LineWidth = 2
        .AxisX.LabelStyle.Font = f
    End With

    'create and set up legends
    .Legends.Add("Pressure/Flux")
    With .Legends(0)
        .IsDockedInsideChartArea = True
        .DockedToChartArea = "Pressure/Flux"
        .Docking = DataVisualization.Charting.Docking.Left
        .BackColor = Color.Transparent
        .Font = f
    End With

    'create and set up series
    .Series.Add("Pressure") '0
    With .Series(0)
        .ChartType = DataVisualization.Charting.SeriesChartType.FastLine
        .ChartArea = "Pressure/Flux"
        .YAxisType = DataVisualization.Charting.AxisType.Primary
        .LegendText = .Name
        .Legend = "Pressure/Flux"
        .Color = Color.Blue
        .BorderWidth = 3
```

```
        End With
        .Series.Add("Flux") '1
        With .Series(1)
            .ChartType = DataVisualization.Charting.SeriesChartType.FastLine
            .ChartArea = "Pressure/Flux"
            .YAxisType = DataVisualization.Charting.AxisType.Secondary
            .LegendText = .Name
            .Legend = "Pressure/Flux"
            .Color = Color.Red
            .BorderWidth = 3
        End With

    Else

        'create and set up chart areas
        .ChartAreas.Add("Pressure/Flux")
        With .ChartAreas(0)
            .AxisY.Title = "Pressure [psi]"
            .AxisY.LabelStyle.Format = "{0:0.##}"
            .AxisY.TitleFont = f
            .AxisY.LineWidth = 2
            .AxisY.LabelStyle.Font = f
            .AxisY2.Title = "Flux [LMH]"
            .AxisY2.LabelStyle.Format = "{0:0.##}"
            .AxisY2.TitleFont = f
            .AxisY2.LineWidth = 2
            .AxisY2.LabelStyle.Font = f
            .AxisX.Title = "Elapsed [3]"
            .AxisX.LabelStyle.Format = "{0:0.###}"
```

```
        .AxisX.TitleFont = f
        .AxisX.LineWidth = 2
        .AxisX.LabelStyle.Font = f
    End With

    'create and set up legends
    .Legends.Add("Pressure/Flux")
    With .Legends(0)
        .IsDockedInsideChartArea = True
        .DockedToChartArea = "Pressure/Flux"
        .Docking = DataVisualization.Charting.Docking.Left
        .BackColor = Color.Transparent
        .Font = f
    End With

    'create and set up series
    .Series.Add("UF Pressure") '0
    With .Series(0)
        .ChartType = DataVisualization.Charting.SeriesChartType.FastLine
        .ChartArea = "Pressure/Flux"
        .YAxisType = DataVisualization.Charting.AxisType.Primary
        .LegendText = .Name
        .Legend = "Pressure/Flux"
        .Color = Color.Blue
        .BorderWidth = 3
    End With
    .Series.Add("NF Pressure") '1
    With .Series(1)
        .ChartType = DataVisualization.Charting.SeriesChartType.FastLine
```

```
            .ChartArea = "Pressure/Flux"
            .YAxisType = DataVisualization.Charting.AxisType.Primary
            .LegendText = .Name
            .Legend = "Pressure/Flux"
            .Color = Color.Red
            .BorderWidth = 3
        End With
        .Series.Add("UF Flux") '2
        With .Series(2)
            .ChartType = DataVisualization.Charting.SeriesChartType.FastLine
            .ChartArea = "Pressure/Flux"
            .YAxisType = DataVisualization.Charting.AxisType.Secondary
            .LegendText = .Name
            .Legend = "Pressure/Flux"
            .Color = Color.Lime
            .BorderWidth = 3
        End With
        .Series.Add("NF Flux") '3
        With .Series(3)
            .ChartType = DataVisualization.Charting.SeriesChartType.FastLine
            .ChartArea = "Pressure/Flux"
            .YAxisType = DataVisualization.Charting.AxisType.Secondary
            .LegendText = .Name
            .Legend = "Pressure/Flux"
            .Color = Color.Orange
            .BorderWidth = 3
        End With
    End If
```

```
        End With

    End Sub

#End Region

#Region "Form Control Events" 'events when the GUI changes (either from user input or
control routines)

    Private Sub btnDOI_Click(sender As Object, e As EventArgs) Handles btnDOI.Click
        TabControl1.SelectedTab =
TabControl1.TabPages(TabControl1.TabPages.IndexOf(tabDOI))
    End Sub

    Private Sub btnAF_Click(sender As Object, e As EventArgs) Handles btnAF.Click
        TabControl1.SelectedTab =
TabControl1.TabPages(TabControl1.TabPages.IndexOf(tabAF))
    End Sub

    Private Sub TabControl1_SelectedIndexChanged(sender As Object, e As EventArgs)
Handles TabControl1.SelectedIndexChanged
        TabChanged()
    End Sub

    Private Sub chkMainPump_CheckedChanged(sender As Object, e As EventArgs) Handles
chkMainPump.CheckedChanged
        If chkMainPump.Checked Then
            isPaused = False
        End If
```

```
        chkMainPump_isChanged = True
        chkMainPump_isChecked = chkMainPump.Checked
    End Sub

    Private Sub chkHSMixer_CheckedChanged(sender As Object, e As EventArgs) Handles
chkHSMixer.CheckedChanged
        If chkHSMixer.Checked Then
            isPaused = False
        End If
        chkHSMixer_isChanged = True
    End Sub

    Private Sub chkLSMixer_CheckedChanged(sender As Object, e As EventArgs) Handles
chkLSMixer.CheckedChanged
        If chkLSMixer.Checked Then
            isPaused = False
        End If
        chkLSMixer_isChanged = True
    End Sub

    Private Sub chkSeparator_CheckedChanged(sender As Object, e As EventArgs) Handles
chkSeparator.CheckedChanged
        If chkSeparator.Checked Then
            isPaused = False
        End If
        chkSeparator_isChanged = True
    End Sub
```

```
    Private Sub chkRinse_CheckedChanged(sender As Object, e As EventArgs) Handles
chkRinse.CheckedChanged
        If chkRinse.Checked Then
            isPaused = False
        End If
        chkRinse_isChanged = True
    End Sub

    Private Sub chkBackwash_CheckedChanged(sender As Object, e As EventArgs) Handles
chkBackwash.CheckedChanged
        If chkBackwash.Checked Then
            isPaused = False
        End If
        chkBackwashPump_isChanged = True
    End Sub

    Private Sub numMainPump_ValueChanged(sender As Object, e As EventArgs) Handles
numMainPump.ValueChanged
        numPumpFreq_isChanged = True
        numPumpFreq_Value = CDbl(numMainPump.Value)
    End Sub

    Private Sub numHSMixer_ValueChanged(sender As Object, e As EventArgs) Handles
numHSMixer.ValueChanged
        numHSMixerFreq_isChanged = True
    End Sub

    Private Sub numSeparator_ValueChanged(sender As Object, e As EventArgs) Handles
numSeparator.ValueChanged
```

```
        numSeparatorFreq_isChanged = True
    End Sub

    Private Sub chkPumpPrefilter_CheckedChanged(sender As Object, e As EventArgs) Handles
chkPumpPrefilter.CheckedChanged
        If chkPumpPrefilter.Checked Then
            isPaused = False
        End If
        chkPumpPrefilter_isChanged = True
    End Sub

    Private Sub chkPumpUFMain_CheckedChanged(sender As Object, e As EventArgs) Handles
chkPumpUFMain.CheckedChanged
        If chkPumpUFMain.Checked Then
            isPaused = False
        End If
        chkPumpUFMain_isChanged = True
        chkPumpUFMain_isChecked = chkPumpUFMain.Checked
    End Sub

    Private Sub chkPumpBackwash_CheckedChanged(sender As Object, e As EventArgs) Handles
chkPumpBackwash.CheckedChanged
        If chkPumpBackwash.Checked Then
            isPaused = False
        End If
        chkPumpUFBackwash_isChanged = True
    End Sub
```

```
    Private Sub chkPumpNFMain_CheckedChanged(sender As Object, e As EventArgs) Handles
chkPumpNFMain.CheckedChanged
        If chkPumpNFMain.Checked Then
            isPaused = False
        End If
        chkPumpNFMain_isChanged = True
        chkPumpNFMain_isChecked = chkPumpNFMain.Checked
    End Sub

    Private Sub chkSelectorUFRecirculate_CheckedChanged(sender As Object, e As EventArgs)
Handles chkSelectorUFRecirculate.CheckedChanged
        If chkSelectorUFRecirculate.Checked Then
            isPaused = False
        End If
        chkSelectorUFRecirculate_isChanged = True
    End Sub

    Private Sub chkSelectorUFFiltrate_CheckedChanged(sender As Object, e As EventArgs)
Handles chkSelectorUFFiltrate.CheckedChanged
        If chkSelectorUFFiltrate.Checked Then
            isPaused = False
        End If
        chkSelectorUFFiltrate_isChanged = True
    End Sub

    Private Sub chkSelectorNFRecirculate_CheckedChanged(sender As Object, e As EventArgs)
Handles chkSelectorNFRecirculate.CheckedChanged
        If chkSelectorNFRecirculate.Checked Then
            isPaused = False
```

```
        End If
        chkSelectorNFRecirculate_isChanged = True
    End Sub

    Private Sub numUF_ValueChanged(sender As Object, e As EventArgs) Handles
numUF.ValueChanged
        isPaused = False
        numUF_isChanged = True
        numUF_Value = CDbl(numUF.Value)
    End Sub

    Private Sub numNF_ValueChanged(sender As Object, e As EventArgs) Handles
numNF.ValueChanged
        isPaused = False
        numNF_isChanged = True
        numNF_Value = CDbl(numNF.Value)
    End Sub

    Private Sub chkPressure_CheckedChanged(sender As Object, e As EventArgs) Handles
chkPressure.CheckedChanged
        If chkPressure.Checked Then
            isPaused = False
        End If
        chkPressure_isChecked = chkPressure.Checked
    End Sub

    Private Sub chkFlow_CheckedChanged(sender As Object, e As EventArgs) Handles
chkFlow.CheckedChanged
        If chkFlow.Checked Then
```

```
            isPaused = False
        End If
        chkFlow_isChecked = chkFlow.Checked
    End Sub

    Private Sub chkUFPressure_CheckedChanged(sender As Object, e As EventArgs) Handles
chkUFPressure.CheckedChanged
        If chkUFPressure.Checked Then
            isPaused = False
        End If
        chkUFPressure_isChecked = chkUFPressure.Checked
    End Sub

    Private Sub chkUFFlow_CheckedChanged(sender As Object, e As EventArgs) Handles
chkUFFlow.CheckedChanged
        If chkUFFlow.Checked Then
            isPaused = False
        End If
        chkUFFlow_isChecked = chkUFFlow.Checked
    End Sub

    Private Sub chkNFPressure_CheckedChanged(sender As Object, e As EventArgs) Handles
chkNFPressure.CheckedChanged
        If chkNFPressure.Checked Then
            isPaused = False
        End If
        chkNFPressure_isChecked = chkNFPressure.Checked
    End Sub
```

```
    Private Sub chkNFFlow_CheckedChanged(sender As Object, e As EventArgs) Handles
chkNFFlow.CheckedChanged
        If chkNFFlow.Checked Then
            isPaused = False
        End If
        chkNFFlow_isChecked = chkNFFlow.Checked
    End Sub

    Private Sub chkDepth_CheckedChanged(sender As Object, e As EventArgs) Handles
chkDepth.CheckedChanged
        If chkDepth.Checked Then
            isPaused = False
        End If
        chkDepth_isChecked = chkDepth.Checked
    End Sub

    Private Sub chkNFFeedFull_CheckedChanged(sender As Object, e As EventArgs) Handles
chkNFFeedFull.CheckedChanged
        If chkNFFeedFull.Checked Then
            isPaused = False
        End If
        chkNFFeedFull_isChecked = chkNFFeedFull.Checked
    End Sub

    Private Sub chkNFFeedEmpty_CheckedChanged(sender As Object, e As EventArgs) Handles
chkNFFeedEmpty.CheckedChanged
        If chkNFFeedEmpty.Checked Then
            isPaused = False
        End If
```

```
        chkNFFeedEmpty_isChecked = chkNFFeedEmpty.Checked
    End Sub

    Private Sub DOICalculatorToolStripMenuItem_Click(sender As Object, e As EventArgs)
Handles DOICalculatorToolStripMenuItem.Click
        Dim temp As New DOI_Calculator.frmMain
        temp.Show()
    End Sub

    Private Sub ExitToolStripMenuItem_Click(sender As Object, e As EventArgs) Handles
ExitToolStripMenuItem.Click
        Me.Close()
    End Sub

    Private Sub btnCollapse_Click(sender As Object, e As EventArgs) Handles
btnCollapse.Click
        If SplitContainer1.Panel1Collapsed Then
            btnCollapse.Text = "<"
            SplitContainer1.Panel1Collapsed = False
            ToolTip1.SetToolTip(btnCollapse, "Hide Controls")
            MenuStrip1.Show()
        Else
            btnCollapse.Text = ">"
            SplitContainer1.Panel1Collapsed = True
            ToolTip1.SetToolTip(btnCollapse, "Show Controls")
            MenuStrip1.Hide()
        End If
    End Sub
```

```
    Private Sub btnStop_Click(sender As Object, e As EventArgs) Handles btnStop.Click
        AnnotationText = "All Stop Clicked"
        isAnnotationPending = True
        AllStop()
    End Sub

    Private Sub AboutUniveralFiltrationControllerToolStripMenuItem_Click(sender As
Object, e As EventArgs) Handles AboutUniveralFiltrationControllerToolStripMenuItem.Click
        AboutBox1.ShowDialog()
    End Sub

    Private Sub btnChartReset_Click(sender As Object, e As EventArgs) Handles
btnChartReset.Click
        ChartSetup()
    End Sub

    Private Sub numMaxUFPressure_ValueChanged(sender As Object, e As EventArgs) Handles
numMaxUFPressure.ValueChanged
        AFPressureMaxUF = CDbl(numMaxUFPressure.Value)
    End Sub

    Private Sub numMaxNFPressure_ValueChanged(sender As Object, e As EventArgs) Handles
numMaxNFPressure.ValueChanged
        AFPressureMaxNF = CDbl(numMaxNFPressure.Value)
    End Sub

    Private Sub numFullFlux_ValueChanged(sender As Object, e As EventArgs) Handles
numFullFlux.ValueChanged
        AFFluxTargetHigh = CDbl(numFullFlux.Value)
```

```
    End Sub

    Private Sub numEmptyFlux_ValueChanged(sender As Object, e As EventArgs) Handles
numEmptyFlux.ValueChanged
        AFFluxTargetLow = CDbl(numEmptyFlux.Value)
    End Sub

    Private Sub chkAFBackwash_CheckedChanged(sender As Object, e As EventArgs) Handles
chkAFBackwash.CheckedChanged
        useBackwashAF = chkAFBackwash.Checked
        If useBackwashAF Then
            SetLabel(lblAFBackwash, "Auto Backwashes On")
        Else
            SetLabel(lblAFBackwash, "Manual Backwash Only")
        End If
    End Sub

    Private Sub btnAFBackwash_Click(sender As Object, e As EventArgs) Handles
btnAFBackwash.Click
        AnnotationText = "AF Manual Backwash Clicked"
        isAnnotationPending = True
        isBackwashPending = True
        SetButtonState(btnAFBackwash, False)
    End Sub

    Private Sub btnAFBackwash_EnableChanged(sender As Object, e As EventArgs) Handles
btnAFBackwash.EnabledChanged
        If btnAFBackwash.Enabled Then
            btnAFBackwash.Text = "Schedule Backwash"
```

```
        Else
            btnAFBackwash.Text = "Backwash Scheduled"
        End If
    End Sub

    Private Sub btnAFFluxHold_Click(sender As Object, e As EventArgs) Handles
btnAFFluxHold.Click
        AnnotationText = "AF Flux Hold Clicked"
        isAnnotationPending = True
        SetButtonState(btnCancelUF, True)
        SetButtonState(btnAFFluxHold, False)
        'InitializeChart() 'TODO - check for functionality
        ProcessWorker.RunWorkerAsync()
    End Sub

    Private Sub btnAFFluxHold_EnabledChanged(sender As Object, e As EventArgs) Handles
btnAFFluxHold.EnabledChanged
        If btnAFFluxHold.Enabled Then
            btnAFFluxHold.Text = "Start Flux Hold"
        Else
            btnAFFluxHold.Text = "Running"
        End If
    End Sub

    Private Sub numMaxPressure_ValueChanged(sender As Object, e As EventArgs) Handles
numMaxPressure.ValueChanged
        DOIPressureMax = CDbl(numMaxPressure.Value)
    End Sub
```

```
    Private Sub numFluxSet_ValueChanged(sender As Object, e As EventArgs) Handles
numFluxSet.ValueChanged
        DOIFluxTarget = CDbl(numFluxSet.Value)
    End Sub

    Private Sub numBackwashDurDOI_ValueChanged(sender As Object, e As EventArgs) Handles
numBackwashDurDOI.ValueChanged
        BackwashDuration = CDbl(numBackwashDurDOI.Value * 1000)
    End Sub

    Private Sub numBackwashFreqDOI_ValueChanged(sender As Object, e As EventArgs) Handles
numBackwashFreqDOI.ValueChanged
        BackwashFrequency = CDbl(numBackwashFreqDOI.Value * 60 * 1000)
    End Sub

    Private Sub chkDOIBackwash_CheckedChanged(sender As Object, e As EventArgs) Handles
chkDOIBackwash.CheckedChanged
        useBackwashDOI = chkDOIBackwash.Checked
        If useBackwashDOI Then
            SetLabel(lblDOIBackwash, "Auto Backwashes On")
        Else
            SetLabel(lblDOIBackwash, "Manual Backwash Only")
        End If
    End Sub

    Private Sub btnDOIFluxHold_Click(sender As Object, e As EventArgs) Handles
btnDOIFluxHold.Click
        isPressureHold = False
        AnnotationText = "DOI Flux Hold Clicked"
```

```
        isAnnotationPending = True
        SetButtonState(btnDOIFluxHold, False)
        SetButtonState(btnDOIFluxHoldCancel, True)
        SetButtonState(btnDOIPressureHold, False)
        starthour = getHours()
        InitializeChart()
        ProcessWorker.RunWorkerAsync()
    End Sub

    Private Sub btnDOIFluxHoldCancel_Click(sender As Object, e As EventArgs) Handles
btnDOIFluxHoldCancel.Click
        AnnotationText = "UF Flux Hold Cancel Clicked"
        isAnnotationPending = True
        btnDOIFluxHoldCancel.Enabled = False
        ProcessWorker.CancelAsync()
    End Sub

    Private Sub btnDOIPressureHoldCancel_Click(sender As Object, e As EventArgs) Handles
btnDOIPressureHoldCancel.Click
        AnnotationText = "UF Pressure Hold Cancel Clicked"
        isAnnotationPending = True
        btnDOIPressureHoldCancel.Enabled = False
        isPressureHold = False
        ProcessWorker.CancelAsync()
    End Sub

    Private Sub btnDOIPressureHold_Click(sender As Object, e As EventArgs) Handles
btnDOIPressureHold.Click
        isPressureHold = True
```

```
        AnnotationText = "DOI Pressure Hold Clicked"
        isAnnotationPending = True
        SetButtonState(btnDOIPressureHold, False)
        SetButtonState(btnDOIPressureHoldCancel, True)
        SetButtonState(btnDOIFluxHold, False)
        starthour = getHours()
        InitializeChart()
        ProcessWorker.RunWorkerAsync()
    End Sub

    Private Sub btnDOIFluxHold_EnabledChanged(sender As Object, e As EventArgs) Handles
btnDOIFluxHold.EnabledChanged
        If btnDOIFluxHold.Enabled Then
            btnDOIFluxHold.Text = "Start Flux Hold"
        Else
            If Not isPressureHold Then
                btnDOIFluxHold.Text = "Running"
            End If
        End If
    End Sub

    Private Sub btnDOIPressureHold_EnabledChanged(sender As Object, e As EventArgs)
Handles btnDOIPressureHold.EnabledChanged
        If btnDOIPressureHold.Enabled Then
            btnDOIPressureHold.Text = "Start Pressure Hold"
        Else
            If isPressureHold Then
                btnDOIPressureHold.Text = "Running"
            End If
```

```
        End If
    End Sub

    Private Sub btnDOIBackwash_Click(sender As Object, e As EventArgs) Handles
btnDOIBackwash.Click
        AnnotationText = "DOI Manual Backwash Clicked"
        isAnnotationPending = True
        isBackwashPending = True
        SetButtonState(btnDOIBackwash, False)
    End Sub

    Private Sub btnDOIBackwash_EnabledChanged(sender As Object, e As EventArgs) Handles
btnDOIBackwash.EnabledChanged
        If btnDOIBackwash.Enabled Then
            btnDOIBackwash.Text = "Schedule Backwash"
        Else
            btnDOIBackwash.Text = "Backwash Scheduled"
        End If
    End Sub

    Private Sub numPressureSet_ValueChanged(sender As Object, e As EventArgs) Handles
numPressureSet.ValueChanged
        AFPressureTarget = CDbl(numPressureSet.Value)
    End Sub

    Private Sub numBackwashDurAF_ValueChanged(sender As Object, e As EventArgs) Handles
numBackwashDurAF.ValueChanged
        BackwashDurationAF = CDbl(numBackwashDurAF.Value * 1000)
    End Sub
```

```
    Private Sub numBackwashFreqAF_ValueChanged(sender As Object, e As EventArgs) Handles
numBackwashFreqAF.ValueChanged
        BackwashFrequencyAF = CDbl(numBackwashFreqAF.Value * 60 * 1000)
    End Sub

    Private Sub numCleaningDuration_ValueChanged(sender As Object, e As EventArgs)
Handles numCleaningDuration.ValueChanged
        CleanDuration = CDbl(numCleaningDuration.Value * 1000)
    End Sub

    Private Sub numCleaningPeriod_ValueChanged(sender As Object, e As EventArgs) Handles
numCleaningPeriod.ValueChanged
        CleanFrequency = CDbl(numCleaningPeriod.Value * 60 * 1000)
    End Sub

    Private Sub chkAFClean_CheckedChanged(sender As Object, e As EventArgs) Handles
chkAFClean.CheckedChanged
        useCleaning = chkAFClean.Checked
        If useCleaning Then
            SetLabel(lblAFClean, "Auto Cleaning On")
        Else
            SetLabel(lblAFClean, "Manual Cleaning Only")
        End If
    End Sub

    Private Sub btnAFClean_Click(sender As Object, e As EventArgs) Handles
btnAFClean.Click
        AnnotationText = "AF Manual Clean Clicked"
```

```
        isAnnotationPending = True
        isCleaningPending = True
        SetButtonState(btnAFClean, False)
    End Sub

    Private Sub btnAFClean_EnableChanged(sender As Object, e As EventArgs) Handles
btnAFClean.EnabledChanged
        If btnAFClean.Enabled Then
            btnAFClean.Text = "Clean"
        Else
            btnAFClean.Text = "Cleaning Scheduled"
        End If
    End Sub

    Private Sub btnAFPrefilter_Click(sender As Object, e As EventArgs) Handles
btnAFPrefilter.Click
        AnnotationText = "Prefilter Clicked"
        isAnnotationPending = True
        SetButtonState(btnCancelPrefiltration, True)
        SetButtonState(btnAFPrefilter, False)
        'InitializeChart() 'TODO - check for functionality
        PrefilterWorker.RunWorkerAsync()
    End Sub

    Private Sub btnAFPrefilter_EnabledChanged(sender As Object, e As EventArgs) Handles
btnAFPrefilter.EnabledChanged
        If btnAFPrefilter.Enabled Then
            btnAFPrefilter.Text = "Start Prefiltration"
        Else
```

```
            btnAFPrefilter.Text = "Running"
        End If
    End Sub

    Private Sub btnAFPressureHold_Click(sender As Object, e As EventArgs) Handles
btnAFPressureHold.Click
        AnnotationText = "AF Pressure Hold Clicked"
        isAnnotationPending = True
        SetButtonState(btnCancelNF, True)
        SetButtonState(btnAFPressureHold, False)
        'InitializeChart() 'TODO - check for functionality
        NFWorker.RunWorkerAsync()
    End Sub

    Private Sub btnAFPressureHold_EnabledChanged(sender As Object, e As EventArgs)
Handles btnAFPressureHold.EnabledChanged
        If btnAFPressureHold.Enabled Then
            btnAFPressureHold.Text = "Start Pressure Hold"
        Else
            btnAFPressureHold.Text = "Running"
        End If
    End Sub

    Private Sub btnCancelPrefiltration_Click(sender As Object, e As EventArgs) Handles
btnCancelPrefiltration.Click
        AnnotationText = "Prefilter Cancel Clicked"
        isAnnotationPending = True
        btnCancelPrefiltration.Enabled = False
        PrefilterWorker.CancelAsync()
```

```
    End Sub

    Private Sub btnCancelUF_Click(sender As Object, e As EventArgs) Handles
btnCancelUF.Click
        AnnotationText = "AF Flux Hold Cancel Clicked"
        isAnnotationPending = True
        btnCancelUF.Enabled = False
        ProcessWorker.CancelAsync()
    End Sub

    Private Sub btnCancelNF_Click(sender As Object, e As EventArgs) Handles
btnCancelNF.Click
        AnnotationText = "AF Pressure Hold Cancel Clicked"
        isAnnotationPending = True
        btnCancelNF.Enabled = False
        NFWorker.CancelAsync()
    End Sub

    Private Sub chkNFRestart_CheckedChanged(sender As Object, e As EventArgs) Handles
chkNFRestart.CheckedChanged
        AutoRestartNF = chkNFRestart.Checked
    End Sub

    Private Sub DataViewerToolStripMenuItem_Click(sender As Object, e As EventArgs)
Handles DataViewerToolStripMenuItem.Click
        Dim temp As New Universal_Data_Viewer.Form1
        temp.Show()
    End Sub
```

```
    Private Sub numDOI_ValueChanged(sender As Object, e As EventArgs) Handles
numDOI.ValueChanged
        numDOI_Value = CDbl(numDOI.Value)
        numDOI_isChanged = True
        isPaused = False
    End Sub

    Private Sub CalibrationToolStripMenuItem_Click(sender As Object, e As EventArgs)
Handles CalibrationToolStripMenuItem.Click
        Dim temp As New frmCalibration
        temp.Show()
    End Sub

    Private Sub btnAnnotate_Click(sender As Object, e As EventArgs) Handles
btnAnnotate.Click
        frmAnnotation.Show()
    End Sub

    Private Sub chkPressure_Click(sender As Object, e As EventArgs) Handles
chkPressure.Click
        AnnotationText = "DOI Pressure Sensor Clicked"
        isAnnotationPending = True
    End Sub

    Private Sub chkFlow_Click(sender As Object, e As EventArgs) Handles chkFlow.Click
        AnnotationText = "DOI Flow Sensor Clicked"
        isAnnotationPending = True
    End Sub
```

```
    Private Sub chkMainPump_Click(sender As Object, e As EventArgs) Handles
chkMainPump.Click
        AnnotationText = "DOI Main Pump Clicked"
        isAnnotationPending = True
    End Sub

    Private Sub chkHSMixer_Click(sender As Object, e As EventArgs) Handles
chkHSMixer.Click
        AnnotationText = "DOI HS Mixer Clicked"
        isAnnotationPending = True
    End Sub

    Private Sub chkLSMixer_Click(sender As Object, e As EventArgs) Handles
chkLSMixer.Click
        AnnotationText = "DOI LS Mixer Clicked"
        isAnnotationPending = True
    End Sub

    Private Sub chkSeparator_Click(sender As Object, e As EventArgs) Handles
chkSeparator.Click
        AnnotationText = "DOI Separator Clicked"
        isAnnotationPending = True
    End Sub

    Private Sub chkRinse_Click(sender As Object, e As EventArgs) Handles chkRinse.Click
        AnnotationText = "DOI Rinse Clicked"
        isAnnotationPending = True
    End Sub
```

```
    Private Sub chkBackwash_Click(sender As Object, e As EventArgs) Handles
chkBackwash.Click
        AnnotationText = "DOI Backwash Pump Clicked"
        isAnnotationPending = True
    End Sub

    Private Sub chkDOIBackwash_Click(sender As Object, e As EventArgs) Handles
chkDOIBackwash.Click
        AnnotationText = "DOI Include Backwash Clicked"
        isAnnotationPending = True
    End Sub

    Private Sub chkUFPressure_Click(sender As Object, e As EventArgs) Handles
chkUFPressure.Click
        AnnotationText = "AF UF Pressure Sensor Clicked"
        isAnnotationPending = True
    End Sub

    Private Sub chkUFFlow_Click(sender As Object, e As EventArgs) Handles chkUFFlow.Click
        AnnotationText = "AF UF Flow Sensor Clicked"
        isAnnotationPending = True
    End Sub

    Private Sub chkNFPressure_Click(sender As Object, e As EventArgs) Handles
chkNFPressure.Click
        AnnotationText = "AF NF Pressure Sensor Clicked"
        isAnnotationPending = True
    End Sub
```

```
    Private Sub chkNFFlow_Click(sender As Object, e As EventArgs) Handles chkNFFlow.Click
        AnnotationText = "AF NF Flow Sensor Clicked"
        isAnnotationPending = True
    End Sub

    Private Sub chkDepth_Click(sender As Object, e As EventArgs) Handles chkDepth.Click
        AnnotationText = "AF Depth Sensor Clicked"
        isAnnotationPending = True
    End Sub

    Private Sub chkNFFeedFull_Click(sender As Object, e As EventArgs) Handles
chkNFFeedFull.Click
        AnnotationText = "NF Feed Full Clicked"
        isAnnotationPending = True
    End Sub

    Private Sub chkNFFeedEmpty_Click(sender As Object, e As EventArgs) Handles
chkNFFeedEmpty.Click
        AnnotationText = "NF Feed Empty Clicked"
        isAnnotationPending = True
    End Sub

    Private Sub chkPumpPrefilter_Click(sender As Object, e As EventArgs) Handles
chkPumpPrefilter.Click
        AnnotationText = "Prefilter Pump Clicked"
        isAnnotationPending = True
    End Sub
```

```
    Private Sub chkPumpUFMain_Click(sender As Object, e As EventArgs) Handles
chkPumpUFMain.Click
        AnnotationText = "AF UF Main Pump Clicked"
        isAnnotationPending = True
    End Sub

    Private Sub chkPumpBackwash_Click(sender As Object, e As EventArgs) Handles
chkPumpBackwash.Click
        AnnotationText = "AF Backwash Pump Clicked"
        isAnnotationPending = True
    End Sub

    Private Sub chkPumpNFMain_Click(sender As Object, e As EventArgs) Handles
chkPumpNFMain.Click
        AnnotationText = "AF NF Main Pump Clicked"
        isAnnotationPending = True
    End Sub

    Private Sub chkSelectorUFRecirculate_Click(sender As Object, e As EventArgs) Handles
chkSelectorUFRecirculate.Click
        AnnotationText = "AF UF Recirculate Clicked"
        isAnnotationPending = True
    End Sub

    Private Sub chkSelectorUFFiltrate_Click(sender As Object, e As EventArgs) Handles
chkSelectorUFFiltrate.Click
        AnnotationText = "AF UF Filtrate Clicked"
        isAnnotationPending = True
    End Sub
```

```
    Private Sub chkSelectorNFRecirculate_Click(sender As Object, e As EventArgs) Handles
chkSelectorNFRecirculate.Click
        AnnotationText = "AF NF Recirculate Clicked"
        isAnnotationPending = True
    End Sub

    Private Sub chkNFRestart_Click(sender As Object, e As EventArgs) Handles
chkNFRestart.Click
        AnnotationText = "AF Auto Start Clicked"
        isAnnotationPending = True
    End Sub

    Private Sub chkAFBackwash_Click(sender As Object, e As EventArgs) Handles
chkAFBackwash.Click
        AnnotationText = "AF UF Include Backwash Clicked"
        isAnnotationPending = True
    End Sub

    Private Sub chkAFClean_Click(sender As Object, e As EventArgs) Handles
chkAFClean.Click
        AnnotationText = "AF UF Include Cleaning Clicked"
        isAnnotationPending = True
    End Sub

    Private Sub btnNFClean_Click(sender As Object, e As EventArgs) Handles
btnNFClean.Click
        AnnotationText = "AF Manual NF Clean Clicked"
        isAnnotationPending = True
```

```
        isNFCleaningPending = True
        SetButtonState(btnNFClean, False)
    End Sub

    Private Sub numNFCleanDuration_ValueChanged(sender As Object, e As EventArgs) Handles
numNFCleanDuration.ValueChanged
        NFCleanDuration = CDbl(numNFCleanDuration.Value * 60 * 1000)
    End Sub

    Private Sub numNFCleaningPeriod_ValueChanged(sender As Object, e As EventArgs)
Handles numNFCleaningPeriod.ValueChanged
        NFCleanFrequency = CDbl(numNFCleaningPeriod.Value * 60 * 1000)
    End Sub

    Private Sub chkNFCleaning_CheckedChanged(sender As Object, e As EventArgs) Handles
chkNFCleaning.CheckedChanged
        useNFCleaning = chkNFCleaning.Checked
        If useNFCleaning Then
            SetLabel(lblNFClean, "Auto Cleaning On")
        Else
            SetLabel(lblNFClean, "Manual Cleaning Only")
        End If
    End Sub

    Private Sub numDOIPressureSet_ValueChanged(sender As Object, e As EventArgs) Handles
numDOIPressureSet.ValueChanged
        DOIPressureTarget = CDbl(numDOIPressureSet.Value)
    End Sub
```

```
    Private Sub chkDOIIncludeBackwashPressure_CheckedChanged(sender As Object, e As
EventArgs) Handles chkDOIIncludeBackwashPressure.CheckedChanged
        useBackwashDOI = chkDOIIncludeBackwashPressure.Checked
        If useBackwashDOI Then
            SetLabel(lblDOIBackwash, "Auto Backwashes On")
        Else
            SetLabel(lblDOIBackwash, "Manual Backwash Only")
        End If
    End Sub

    Private Sub ParametersToolStripMenuItem_Click(sender As Object, e As EventArgs)
Handles ParametersToolStripMenuItem.Click
        frmSettings.ShowDialog()
    End Sub

    Private Sub DOIToolStripMenuItem1_Click(sender As Object, e As EventArgs) Handles
DOIToolStripMenuItem1.Click
        isPaused = True
        TabControl1.SelectedIndex = 1 'set to DOI tab
    End Sub

    Private Sub AFToolStripMenuItem1_Click(sender As Object, e As EventArgs) Handles
AFToolStripMenuItem1.Click
        isPaused = True
        TabControl1.SelectedIndex = 2 'set to AF tab
    End Sub

    Private Sub chkHeadStart_CheckedChanged(sender As Object, e As EventArgs) Handles
chkHeadStart.CheckedChanged
```

```
        headStart = chkHeadStart.Checked
    End Sub

#End Region

#Region "Get Sensor Data"

    Private Sub GetFlowrate(ByRef holder() As Double, pin As Byte)

        Dim index As Integer
        Select Case pin
            Case pinFlowrate
                index = 0
            Case pinUFFlowrate
                index = 1
            Case pinNFFlowrate
                index = 2
        End Select

        pps = SerialWrite(6, pin, 0) 'pulses

        Dim ans As Double
        If pps = 0 Then
            ans = 0
        Else
            ans = Flowmeters * ((60 * pps) - FlowmeterOffset(index)) /
PulsePerLiter(index) 'LPM
        End If
```

```
    'store values and get average
    holder(3) = holder(2)
    holder(2) = holder(1)
    holder(1) = ans

    'average
    holder(0) = (holder(1) + holder(2) + holder(3)) / 3

    UpdateChart(pin * 100, FlowToFlux(holder(0)))

End Sub

Private Sub GetPressure(ByRef holder() As Double, pin As Byte)

    'get value
    Dim val As Double = SensorToValue(pin, SerialWrite(3, pin, PressureSamples))

    'store values and get average
    holder(3) = holder(2)
    holder(2) = holder(1)
    holder(1) = val

    holder(0) = (holder(1) + holder(2) + holder(3)) / 3

    UpdateChart(pin, holder(0))
    If isDOI Then
        UpdateCrossFlow()
    End If
```

```
    End Sub

    Private Sub GetState(ByRef holder() As Boolean, pin As Byte)

        'get value
        Dim state As Boolean = GetSwitch(pin)

        'store values and get average
        holder(3) = holder(2)
        holder(2) = holder(1)
        holder(1) = state

        'get average
        Dim sum As Integer = 0
        For i As Integer = 1 To 3
            If holder(i) Then
                sum += 1
            End If
        Next
        If sum > 1 Then
            holder(0) = True
        Else
            holder(0) = False
        End If

    End Sub

#End Region
```

```
#Region "Supporting Functions"

    'get start hour and reset chart if no processes are active
    Private Sub InitializeChart()
        If ProcessWorker.IsBusy() Then
            Exit Sub
        End If
        If PrefilterWorker.IsBusy() Then
            Exit Sub
        End If
        If NFWorker.IsBusy() Then
            Exit Sub
        End If
        starthour = getHours()
        ChartSetup()
    End Sub

    'takes the pin and returns the chart series ID it belongs to
    Function PinToSeries(pinID As Integer) As Integer
        If isDOI Then
            Select Case pinID
                Case pinPressure
                    Return 0
                Case pinFlowrate * 100
                    Return 1
            End Select
        Else
            Select Case pinID
                Case pinUFPressure
```

```
                Return 0
            Case pinNFPressure
                Return 1
            Case pinUFFlowrate * 100
                Return 2
            Case pinNFFlowrate * 100
                Return 3
        End Select
    End If
    Return -1
End Function

'set the pin functions of the Arduino
Private Sub SetPins()

    If isDOI Then
        'digitalread
        SerialWrite(5, pinFlowrate, 1)

        'digitalwrite
        SerialWrite(5, pinSeparator, 2)
        SerialWrite(5, pinHSMix, 2)
        SerialWrite(5, pinBackwash, 2)
        SerialWrite(5, pinLSMix, 2)
        SerialWrite(5, pinPump, 2)
        SerialWrite(5, pinWash, 2)
    Else
        'digitalread
        SerialWrite(5, pinUFFlowrate, 1)
```

```
            SerialWrite(5, pinNFFlowrate, 1)
            SerialWrite(5, pinNFFull, 1)
            SerialWrite(5, pinNFEmpty, 1)

            'digitalwrite
            SerialWrite(5, pinPrefilterPump, 2)
            SerialWrite(5, pinUFMainPump, 2)
            SerialWrite(5, pinUFBackwashPump, 2)
            SerialWrite(5, pinNFMainPump, 2)
            SerialWrite(5, pinUFRecirculateValve, 2)
            SerialWrite(5, pinUFFiltrateValve, 2)
            SerialWrite(5, pinNFRecirculateValve, 2)
        End If

    End Sub

    'get polynomial coefficients to calibration data
    Private Sub GetFits()

        Dim pUF_analog(2) As Double
        Dim pNF_analog(2) As Double
        Dim Lev_analog(2) As Double
        Dim p_analog(2) As Double
        Dim factor As Double = 1.023 / 5.0

        For i As Integer = 0 To 2
            pUF_analog(i) = pUF_resistor * pUF_sig(i) * factor
            pNF_analog(i) = pNF_resistor * pNF_sig(i) * factor
            Lev_analog(i) = Lev_resistor * Lev_sig(i) * factor
```

```
            p_analog(i) = p_resistor * p_sig(i) * factor
        Next

        'regression analysis
        polyfit(poly_UF, pUF_analog, pUF_psi)
        polyfit(poly_NF, pNF_analog, pNF_psi)
        polyfit(poly_Lev, Lev_analog, Lev_depth)
        polyfit(poly, p_analog, p_psi)

    End Sub

    'return 2nd-order polynomial coefficients for least-squares fit data
    Private Sub polyfit(ByRef ans() As Double, x() As Double, y() As Double)

        Dim sumx As Double = 0
        Dim sumx2 As Double = 0
        Dim sumx3 As Double = 0
        Dim sumx4 As Double = 0
        Dim sumy As Double = 0
        Dim sumyx As Double = 0
        Dim sumyx2 As Double = 0

        'get sums and products
        For i As Integer = 0 To 2
            Dim x2 As Double = x(i) * x(i)
            sumx += x(i)
            sumx2 += x2
            sumx3 += x(i) * x2
            sumx4 += x2 * x2
```

```
    sumy += y(i)
    sumyx += y(i) * x(i)
    sumyx2 += y(i) * x2
Next

'create augmented matrix
Dim Ab(3, 4) As Double
Ab(0, 0) = 3 'n
Ab(0, 1) = sumx
Ab(0, 2) = sumx2
Ab(0, 3) = sumy
Ab(1, 0) = sumx
Ab(1, 1) = sumx2
Ab(1, 2) = sumx3
Ab(1, 3) = sumyx
Ab(2, 0) = sumx2
Ab(2, 1) = sumx3
Ab(2, 2) = sumx4
Ab(2, 3) = sumyx2

'make first column ones
For row As Integer = 0 To 2
    For col As Integer = 1 To 3
        Ab(row, col) = Ab(row, col) / Ab(row, 0)
    Next
Next
Ab(0, 0) = 1
Ab(1, 0) = 1
Ab(2, 0) = 1
```

```
'subtract row 1 from rows 2 and 3
For row As Integer = 1 To 2
    For col As Integer = 0 To 3
        Ab(row, col) = Ab(row, col) - Ab(0, col)
    Next
Next

'make the second column (2nd + 3rd row) ones
For row As Integer = 1 To 2
    For col As Integer = 2 To 3
        Ab(row, col) = Ab(row, col) / Ab(row, 1)
    Next
Next
Ab(1, 1) = 1
Ab(2, 1) = 1

'subtract row 2 from row 3
For col As Integer = 1 To 3
    Ab(2, col) = Ab(2, col) - Ab(1, col)
Next

'make the third column (3rd row) ones
Ab(2, 3) = Ab(2, 3) / Ab(2, 2)
Ab(2, 2) = 1

'back substitution
ans(2) = Ab(2, 3)
ans(1) = Ab(1, 3) - ans(2) * Ab(1, 2)
```

```
    ans(0) = Ab(0, 3) - ans(2) * Ab(0, 2) - ans(1) * Ab(0, 1)

End Sub

'set initial states and values
Public Sub SetupEnvironment()

    isPaused = True
    SetPins() 'set the digital pins in the Arduino to input or output
    CreateDirectory() 'create or check for a data directory
    GetFits() 'get the polynomial curve fits for sensor calibrations
    ChartSetup() 'set up the chart for data tracking
    starthour = getHours() 'record the starting hour
    If Not isUpdaterRunning Then
        ArduinoUpdater.RunWorkerAsync() 'start the form updater
        isUpdaterRunning = True
    End If

    'flowmeter
    PulsePerLiter(0) = My.Settings.DOI_PPL
    PulsePerLiter(1) = My.Settings.AF_UF_PPL
    PulsePerLiter(2) = My.Settings.AF_NF_PPL
    FlowmeterOffset(0) = My.Settings.DOI_Offset
    FlowmeterOffset(1) = My.Settings.AF_UF_Offset
    FlowmeterOffset(2) = My.Settings.AF_NF_Offset

    'set valves wide open
    If isDOI Then
        SetNumeric(numDOI, 100)
```

```
        Else
            SetNumeric(numUF, 100)
            SetNumeric(numNF, 100)
        End If
        isPaused = False

    End Sub

    'returns the software version
    Private Function getVersion() As String
        Try
            Dim v As System.Version =
System.Deployment.Application.ApplicationDeployment.CurrentDeployment.CurrentVersion
            Return v.Major.ToString & "." & v.Minor.ToString & "." & v.Build.ToString
        Catch ex As Exception
            Return "1.0.0"
        End Try
    End Function

    'starts USB communication with the Arduino
    Private Function OpenPort() As Boolean
        'get available ports
        Dim portnames As String() = SerialPort.GetPortNames()

        If portnames.Length < 1 Then
            MessageBox.Show("No ports available. Connect Control Unit to USB and restart
program. The program will operate in simulation mode.", _
                            "Connection Missing", MessageBoxButtons.OK,
MessageBoxIcon.Exclamation)
```

```
            isSimulation = True
            Return False
        End If

        If portnames.Length <> 1 Then
            MessageBox.Show("Too many ports connected. Fix the problem and restart the
program. The program will operate in simulation mode.", _
                            "Communication Error", MessageBoxButtons.OK,
MessageBoxIcon.Error)
            isSimulation = True
            Return False
        End If

        'start communication
        With myPort
            .PortName = portnames(0)
            .BaudRate = 500000
            .Parity = Parity.None
            .DataBits = 8
            .StopBits = StopBits.One
            .DtrEnable = True
            .RtsEnable = True
            .ReceivedBytesThreshold = 10
            .Open()
        End With

        'wait for opened port
        Dim count As Integer = 0
        While Not myPort.IsOpen()
```

```
            Sleep(1)
            count += 1
            If count >= 10000 Then
                MessageBox.Show("Communication unsuccessful. Fix the problem and restart
the program. The program will operate in simulation mode.", _
                                "Port Failure", MessageBoxButtons.OK,
MessageBoxIcon.Error)
                Return False
            End If
        End While

        Return True

    End Function

    'calculates cross-flow velocities based on area, pump speed, and pressure
    Private Sub UpdateCrossFlow()

        'data holders
        Dim Flat, Hollow, Spiral As Double

        'get flowrate
        Dim flowrate As Double = getPumpFlow(Pressure(0), numPumpFreq_Value) '[m^3/s]

        'calcaulte flow velocities [cm/s]
        Flat = 100 * flowrate / A_Flat
        Hollow = 100 * flowrate / (A_Hollow * MembraneCountDOI)
        Spiral = 100 * flowrate / (A_Spiral * MembraneCountDOI)
```

```
    'update labels
    SetLabel(lblFlat, Flat.ToString("0.0"))
    SetLabel(lblHollow, Hollow.ToString("0.0"))
    SetLabel(lblSpiral, Spiral.ToString("0.0"))

End Sub

'calculates the flowrate given the pump speed and pressure
Private Function getPumpFlow(p As Double, hz As Double)

    Dim m_0 As Double = 0.1537
    Dim m_25 As Double = 0.1141
    Dim m_50 As Double = 0.0711

    Dim m As Double
    Dim f As Double
    If p = 0 Then
        m = m_0
    ElseIf p > 0 And p < 25 Then
        f = (p - 0) / (25 - 0)
        m = m_0 + f * (m_25 - m_0)
    ElseIf p = 25 Then
        m = m_25
    ElseIf p > 25 And p < 50 Then
        f = (p - 25) / (50 - 25)
        m = m_25 + f * (m_50 - m_25)
    ElseIf p = 50 Then
        m = m_50
    Else
```

```
            Return 0
        End If

        Dim galpermin As Double = hz * m '[gal/min]
        Return galpermin * 0.0000630901964 '[m3/s]

        Return 0
    End Function

#End Region

#Region "Form Events"

    'confirm exit and close program
    Private Sub frmMain_FormClosing(sender As Object, e As FormClosingEventArgs) Handles
Me.FormClosing

        AnnotationText = "Program is Closing"
        isAnnotationPending = True

        If MsgBox("Confirm Exit?", MsgBoxStyle.YesNo) = MsgBoxResult.No Then
            e.Cancel = True
        Else

            AnnotationText = "User Confirmed Close"
            isAnnotationPending = True

            'save exit time
```

```
            Dim filepath As String = My.Computer.FileSystem.SpecialDirectories.Desktop &
"\Exit Hour.txt"
            My.Computer.FileSystem.WriteAllText(filepath, getHours().ToString("0.000000")
& vbCrLf, True)

            'stop processes
            AllStop()

        End If
    End Sub

    'initialize serial communications
    Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load

        Me.Text = "Universal Filtration Controller - v" & getVersion() 'set title +
program version info
        If Not OpenPort() Then 'start communication with the arduino
            Exit Sub 'goto simulation mode
        End If
        Sleep(3000) 'wait for communication to be established

    End Sub

    'display the appropriate GUI page
    Private Sub frmMain_Shown(sender As Object, e As EventArgs) Handles Me.Shown
        TabChanged()
    End Sub

#End Region
```

```
#Region "Processes"

    'stops all processes/sensors immediately
    Private Sub AllStop()

        'stop any running processes
        ProcessWorker.CancelAsync()
        NFWorker.CancelAsync()
        PrefilterWorker.CancelAsync()

        If isDOI Then
            'pumps off
            SetCheckBox(chkMainPump, False)
            SetCheckBox(chkBackwash, False)

            'devices off
            SetCheckBox(chkHSMixer, False)
            SetCheckBox(chkLSMixer, False)
            SetCheckBox(chkSeparator, False)
            SetCheckBox(chkRinse, False)

            'sensors off
            SetCheckBox(chkPressure, False)
            SetCheckBox(chkFlow, False)

            'pressure control valve
            SetNumeric(numDOI, 100)
```

```
    Else
        'sensors
        SetCheckBox(chkUFPressure, False)
        SetCheckBox(chkUFFlow, False)
        SetCheckBox(chkNFPressure, False)
        SetCheckBox(chkNFFlow, False)
        SetCheckBox(chkDepth, False)
        SetCheckBox(chkNFFeedFull, False)
        SetCheckBox(chkNFFeedEmpty, False)

        'pumps
        SetCheckBox(chkPumpPrefilter, False)
        SetCheckBox(chkPumpUFMain, False)
        SetCheckBox(chkPumpBackwash, False)
        SetCheckBox(chkPumpNFMain, False)

        'selector valves
        SetCheckBox(chkSelectorUFRecirculate, False)
        SetCheckBox(chkSelectorUFFiltrate, False)
        SetCheckBox(chkSelectorNFRecirculate, False)

        'pressure control valves
        SetNumeric(numUF, 100)
        SetNumeric(numNF, 100)

    End If

End Sub
```

```
'run AF prefiltration
Private Sub RunPrefilter()

    SetCheckBox(chkDepth, True) 'turn on depth monitoring
    SetCheckBox(chkPumpPrefilter, True) 'turn on prefilter pump

    While True

        Sleep(100) 'wait a bit

        'check for cancellation
        If PrefilterWorker.CancellationPending Then
            AnnotationText = "Prefilter Cancel Exit"
            isAnnotationPending = True
            Exit While
        End If

        'check for full
        If Depth(0) > maxdepth Then
            AnnotationText = "Prefiter Full Exit"
            isAnnotationPending = True
            Exit While
        End If

    End While

    SetCheckBox(chkPumpPrefilter, False) 'turn off prefilter pump
    If Not ProcessWorker.IsBusy Then
```

```
            SetCheckBox(chkDepth, False) 'turn off depth monitoring if it's not being
used
        End If

    End Sub

    'run AF NF filtration
    Private Sub RunNF()

        'wait for head start
        If headStart Then
            SetCheckBox(chkDepth, True) 'ensure depth sensor is on
            Sleep(2000)
            While Depth(0) > 2.5

                Sleep(1000) 'TODO this will prevent simulation from working until
simulation is changed to change depth

                'check for cancellation
                If NFWorker.CancellationPending Then
                    AnnotationText = "NF Cancel Exit"
                    isAnnotationPending = True
                    GoTo ShutDown
                    Exit While 'TODO this line does nothing - change other "Exit While"s
to GoTo
                End If

            End While
        End If
```

```
        'stopwatch
        Dim swc As New Stopwatch
        swc.Start()

        'checks
        Dim state As Boolean
        Dim switchcount As Integer

        'reset checks
        AF_NF_FaultCondition = False
        AF_NF_HighFTimer.Reset()
        AF_NF_HighPTimer.Reset()

        'initialize
        isNFEmpty = {False, False, False, False}

        Dim wasEmpty As Boolean = False
NFStart:
        'loop while waiting for feed
        While wasEmpty
            If isNFEmpty(0) Then

                Sleep(1000)

                'check for cancellation
                If NFWorker.CancellationPending Then
                    wasEmpty = False
                    GoTo ShutDown
```

```
        End If
    Else
        wasEmpty = False
    End If
End While

'turn on sensors/pumps
SetCheckBox(chkNFFeedEmpty, True)
SetCheckBox(chkNFFlow, True)
SetCheckBox(chkNFPressure, True)
SetCheckBox(chkPumpNFMain, True)

Sleep(SensorRespond + 2000) 'wait for sensors to get going

'run indefinitely
While True

    'check to see if feed tank is empty
    If isNFEmpty(0) Then
        If AutoRestartNF And ProcessWorker.IsBusy Then
            wasEmpty = True
        End If
        AnnotationText = "NF NFEmpty Exit"
        isAnnotationPending = True
        Exit While
    End If

    'check for overpressure or overflow
    If AF_NF_FaultCondition Then
```

```
    AF_NF_FaultCondition = False
    AnnotationText = "NF Fault Exit"
    isAnnotationPending = True
    Exit While
End If

'check if pressure needs adjustment
If NFPressure(0) > AFPressureTarget * (1 + SettingRange) Then
    isPressureAdjusting = True
    switchcount = 0
ElseIf NFPressure(0) < AFPressureTarget * (1 - SettingRange) Then
    isPressureAdjusting = True
    switchcount = 0
End If

'Pressure Adjustment
If isPressureAdjusting Then

    'pressure match
    Dim old As Boolean = state
    Dim newval As Decimal
    If NFPressure(0) > AFPressureTarget Then
        newval = numNF_Value + (100.0 / 255.0)
        state = True
    Else
        newval = numNF_Value - (100.0 / 255.0)
        state = False
    End If
```

```
        'Check Bounds
        If newval > 0 Then
            If newval < 100 Then
                SetNumeric(numNF, newval) 'okay
                StatusColor = Color.Lime
            Else
                SetNumeric(numNF, 100) 'too high
                StatusColor = Color.Red
            End If
        Else
            SetNumeric(numNF, 0) 'too low
            StatusColor = Color.Red
        End If

        'check for switch
        If old <> state Then
            switchcount += 1
        End If

        'stop adjustment
        If Not state Then
            If switchcount >= 2 Then
                isPressureAdjusting = False
            End If
        End If

    End If

    'check cleaning timer
```

```
            If useNFCleaning Then

                'schedule cleaning
                If swc.ElapsedMilliseconds > NFCleanFrequency Then
                    isNFCleaningPending = True
                Else
                    SetLabel(lblNFClean, "Auto in " & ((NFCleanFrequency -
swc.ElapsedMilliseconds) / 60000).ToString("0.#") & " min") 'update timer
                End If
            End If

            'run clean if necessary
            If isNFCleaningPending Then
                RunNFCleaning()
                swc.Restart()
            End If

            Sleep(SensorRespond) 'wait a bit

            'check for cancellation
            If NFWorker.CancellationPending Then
                AnnotationText = "NF Cancel Exit"
                isAnnotationPending = True
                Exit While
            End If

        End While

ShutDown:
```

```
    'turn off sensors/pumps
    SetCheckBox(chkNFFlow, False)
    SetCheckBox(chkNFPressure, False)
    SetCheckBox(chkPumpNFMain, False)
    If Not ProcessWorker.IsBusy Then
        SetCheckBox(chkNFFeedFull, False)
    End If
    If wasEmpty Then
        GoTo NFStart
    Else
        SetCheckBox(chkNFFeedEmpty, False)
    End If

End Sub

'run UF filtration
Private Sub RunUF()

    'start a timer
    Dim sw As New Stopwatch
    sw.Start()

    'reset checks
    DOI_FaultCondition = False
    AF_UF_FaultCondition = False
    AF_NF_FaultCondition = False

    DOI_HighFTimer.Reset()
```

```
DOI_HighPTimer.Reset()
AF_UF_HighFTimer.Reset()
AF_UF_HighPTimer.Reset()

If isDOI Then

    'ensure backwash pump is off
    SetCheckBox(chkBackwash, False)

    'turn on sensors
    SetCheckBox(chkPressure, True) 'pressure transducer
    SetCheckBox(chkFlow, True) 'flow meter

    'turn on devices
    SetCheckBox(chkSeparator, True) 'magnetic separator
    Sleep(1000)
    SetCheckBox(chkRinse, True) 'separator rinse
    Sleep(1000)
    SetCheckBox(chkLSMixer, True) 'Low shear mixer
    Sleep(1000)
    SetCheckBox(chkHSMixer, True) 'High shear mixer
    Sleep(1000)

    'turn on pump
    SetCheckBox(chkMainPump, True) 'main pump

    Sleep(SensorRespond + 2000) 'wait for sensors to get going

    'reset counter
```

```
DOI_High_Count = 0


'run indefinitely
While True


    'check for overpressure or overflow
    If DOI_FaultCondition Then
        DOI_FaultCondition = False
        AnnotationText = "UF Fault Exit"
        isAnnotationPending = True
        Exit While
    End If


    'match flux (or pressure)
    If DOI_High_Count < 10 Then
        If DOI_High_Count = 0 Then
            SetNumeric(numMainPump, maxPumpFreq) 'set pump to maximum speed
            Sleep(SensorRespond) 'wait for new measurements
        End If
        MatchFluxDOIValve() 'match the flux by adjusting the valve
    Else
        MatchFluxDOIPump() 'match the flux by adjusting the pump speed
    End If



    'check backwash timer
    If useBackwashDOI Then


        'schedule backwash
```

```
                    If sw.ElapsedMilliseconds > BackwashFrequency Then
                        isBackwashPending = True
                    Else
                        SetLabel(lblDOIBackwash, "Auto in " & ((BackwashFrequency -
sw.ElapsedMilliseconds) / 60000).ToString("0.#") & " min") 'update timer
                    End If

                End If

                'run backwash if necessary
                If isBackwashPending Then
                    RunBackwash()
                    sw.Restart()
                End If

                'check for cancellation
                If ProcessWorker.CancellationPending Then
                    AnnotationText = "UF Cancel Exit"
                    isAnnotationPending = True
                    Exit While
                End If

                'wait for new data
                Sleep(SensorRespond)

            End While

            'turn off pump
            SetCheckBox(chkMainPump, False)
```

```
        'turn off sensors
        SetCheckBox(chkPressure, False) 'pressure transducer
        SetCheckBox(chkFlow, False) 'flow meter

        'turn off devices
        SetCheckBox(chkSeparator, False) 'magnetic separator
        SetCheckBox(chkRinse, False) 'separator rinse
        SetCheckBox(chkLSMixer, False) 'Low shear mixer
        SetCheckBox(chkHSMixer, False) 'High shear mixer

    Else

        Dim state As Boolean
        Dim switchcount As Integer

        'ensure backwash pump is off, and valves are in correct locations
        SetCheckBox(chkPumpBackwash, False)
        SetCheckBox(chkSelectorUFFiltrate, False)
        SetCheckBox(chkSelectorUFRecirculate, False)

        'start a cleaning timer
        Dim swc As New Stopwatch
        swc.Start()

        'turn on devices
        SetCheckBox(chkUFPressure, True) 'UF Pressure
        SetCheckBox(chkUFFlow, True) 'UF Flow
        SetCheckBox(chkDepth, True) 'UF Feed Depth
```

```
SetCheckBox(chkNFFeedFull, True) 'NF Feed (UF Filtrate) Full
SetCheckBox(chkPumpUFMain, True) 'UF Main Pump

'initialize depth and feed
Depth = {maxdepth, maxdepth, maxdepth, maxdepth}
isNFFull = {False, False, False, False}
isNFEmpty = {False, False, False, False}

Sleep(SensorRespond + 3000) 'wait for sensors to get going

'run indefinitely
While True

    'check for overpressure or overflow
    If AF_UF_FaultCondition Then
        AF_UF_FaultCondition = False
        AnnotationText = "UF Fault Exit"
        isAnnotationPending = True
        Exit While
    End If

    'check for empty tank
    If Depth(0) <= mindepth Then
        AnnotationText = "UF Depth Exit"
        isAnnotationPending = True
        Exit While
    End If

    'check for overfull tank
```

```
If isNFFull(0) Then
    AnnotationText = "UF NFFull Exit"
    isAnnotationPending = True
    Exit While
End If

'check if flux needs adjustment
If UFFlowrate(0) > GetFlowrateSet() * (1 + SettingRange) Then
    isFluxAdjusting = True
    switchcount = 0
ElseIf UFFlowrate(0) < GetFlowrateSet() * (1 - SettingRange) Then
    isFluxAdjusting = True
    switchcount = 0
End If

'Flux Adjustment
If isFluxAdjusting Then

    'flux match
    Dim old As Boolean = state
    Dim newval As Decimal
    If UFFlowrate(0) > GetFlowrateSet() Then
        newval = CDec(numUF_Value + (100.0 / 255.0))
        state = True
    Else
        newval = CDec(numUF_Value - (100.0 / 255.0))
        state = False
    End If
```

```
        'Check Bounds
        If newval > 0 Then
            If newval < 100 Then
                SetNumeric(numUF, newval) 'okay
                StatusColor = Color.Lime
            Else
                SetNumeric(numUF, 100) 'too high
                StatusColor = Color.Red
            End If
        Else
            SetNumeric(numUF, 0) 'too low
            StatusColor = Color.Red
        End If

        'check for switch
        If old <> state Then
            switchcount += 1
        End If

        'stop adjustment
        If Not state Then
            If switchcount >= 2 Then
                isFluxAdjusting = False
            End If
        End If

    End If

    'check backwash timer
```

```
                If useBackwashAF Then

                    'schedule backwash
                    If sw.ElapsedMilliseconds > BackwashFrequencyAF Then
                        isBackwashPending = True
                    Else
                        SetLabel(lblAFBackwash, "Auto in " & ((BackwashFrequencyAF -
sw.ElapsedMilliseconds) / 60000).ToString("0.#") & " min") 'update timer
                    End If

                End If

                'run backwash if necessary
                If isBackwashPending Then
                    RunBackwash()
                    sw.Restart()
                End If

                'check cleaning timer
                If useCleaning Then

                    'schedule cleaning
                    If swc.ElapsedMilliseconds > CleanFrequency Then
                        isCleaningPending = True
                    Else
                        SetLabel(lblAFClean, "Auto in " & ((CleanFrequency -
swc.ElapsedMilliseconds) / 60000).ToString("0.#") & " min") 'update timer
                    End If
                End If
```

```
    'run clean if necessary
    If isCleaningPending Then
        RunCleaning()
        swc.Restart()
    End If

    'check for cancellation
    If ProcessWorker.CancellationPending Then
        AnnotationText = "UF Cancel Exit"
        isAnnotationPending = True
        Exit While
    End If

    'wait for new data
    Sleep(SensorRespond)

End While

'turn off devices
SetCheckBox(chkUFPressure, False) 'UF Pressure
SetCheckBox(chkUFFlow, False) 'UF Flow
SetCheckBox(chkPumpUFMain, False) 'UF Main Pump
If Not PrefilterWorker.IsBusy() Then
    SetCheckBox(chkDepth, False) 'UF Feed Depth
End If
If Not NFWorker.IsBusy Then
    SetCheckBox(chkNFFeedFull, False) 'NF Feed (UF Filtrate) Full
End If
```

```
    End If

End Sub

'match the flux with the pump speed
Private Sub MatchFluxDOIPump()

    'flux matching I (for pump adjustment)
    Dim newval As Decimal
    If isPressureHold Then
        If Pressure(1) > DOIPressureTarget Then
            newval = CDec(numPumpFreq_Value - (60.0 / 255))
        Else
            newval = CDec(numPumpFreq_Value + (60.0 / 255))
        End If
    Else
        If Flowrate(1) > GetFlowrateSet() Then
            newval = CDec(numPumpFreq_Value - (60.0 / 255))
        Else
            newval = CDec(numPumpFreq_Value + (60.0 / 255))
        End If
    End If

    'Check Bounds
    If newval > minPumpFreq Then
        If newval < maxPumpFreq Then
            SetNumeric(numMainPump, newval) 'okay
            StatusColor = Color.Lime
```

```
            Else
                SetNumeric(numMainPump, maxPumpFreq) 'pump speed is too high and filtrate
flow is too low (close valve if possible)
                StatusColor = Color.Red
                DOI_High_Count = 0 'give control to valve
            End If
        Else
            SetNumeric(numMainPump, minPumpFreq) 'pump speed too low and filtrate flow is
too high (open valve if possible)
            StatusColor = Color.Red
            DOI_High_Count = 0 'give control to valve
        End If

    End Sub

    'match the flux with the valve
    Private Sub MatchFluxDOIValve()

        'Flux Matching II (original valve adjustment)
        Dim newval As Decimal
        If isPressureHold Then
            If Pressure(1) > DOIPressureTarget Then
                newval = CDec(numDOI_Value + (100.0 / 255.0))
                DOI_High_Count += 1 'count as one against valve control (switches to pump
after 10)
            Else
                newval = CDec(numDOI_Value - (100.0 / 255.0))
            End If
        Else
```

```
            If Flowrate(1) > GetFlowrateSet() Then
                newval = CDec(numDOI_Value + (100.0 / 255.0))
                DOI_High_Count += 1 'count as one against valve control (switches to pump
after 10)
            Else
                newval = CDec(numDOI_Value - (100.0 / 255.0))
            End If
        End If



        'Check Bounds
        If newval > 0 Then
            If newval < 100 Then
                SetNumeric(numDOI, newval) 'okay
                StatusColor = Color.Lime
            Else
                SetNumeric(numDOI, 100) 'valve opening is too high and filtrate flow is
too high (slow down pump if possible)
                StatusColor = Color.Red
                DOI_High_Count = 10 'give control to pump
            End If
        Else
            SetNumeric(numDOI, 0) 'valve opening is too small and filtrate flow is too
low (speed up pump if possible)
            StatusColor = Color.Red
            DOI_High_Count = 10 'give control to pump
        End If
```

```
    End Sub

    'gets the flowrate to match given the current depth
    Private Function GetFlowrateSet() As Double
        If isDOI Then
            Return FluxToFlow(DOIFluxTarget)
        Else
            Return FluxToFlow(GetRequiredFlux())
        End If

    End Function

    'gets the required flux for the current depth
    Private Function GetRequiredFlux() As Double
        Dim percentage As Double = 100 * Depth(0) / maxdepth
        Dim slope As Double = (AFFluxTargetHigh - AFFluxTargetLow) / (100 - 10)
        Dim ans As Double = AFFluxTargetLow + slope * (percentage - 10)
        Return ans
    End Function

    'convert given flux to flowrate
    Private Function FluxToFlow(flux As Double) As Double
        If isDOI Then
            Return flux * FilterArea * MembraneCountDOI / 60.0 '[LPM] - target flowrate
for UF
        Else
            Return flux * FilterArea * MembraneCountAF / 60.0 '[LPM] - target flowrate
for UF
        End If
```

```
End Function

'convert given flowrate to flux
Private Function FlowToFlux(flow As Double) As Double
    If isDOI Then
        Return flow * 60 / (MembraneCountDOI * FilterArea)
    Else
        Return flow * 60 / (MembraneCountAF * FilterArea)
    End If
End Function

'run backwash on UF membrane
Private Sub RunBackwash()
    If isDOI Then

        SetButtonState(btnDOIBackwash, False) 'disable manual backwashes
        SetButtonText(btnDOIBackwash, "Backwashing")

        Dim prevState As Boolean = chkMainPump_isChecked
        SetCheckBox(chkMainPump, False) 'turn off main pump
        SetCheckBox(chkBackwash, True) 'turn on backwash pump

        'start timer
        Dim temp As New Stopwatch
        temp.Start()

        While True
```

```
                SetLabel(lblDOIBackwash, "Backwashing - " & ((BackwashDuration -
temp.ElapsedMilliseconds) / 1000).ToString("0") & " s") 'update label

                Sleep(100) 'wait a bit

                'check for cancellation
                If ProcessWorker.CancellationPending Then
                    AnnotationText = "Backwash Cancel Exit"
                    isAnnotationPending = True
                    Exit While
                End If

                'check for completion
                If temp.ElapsedMilliseconds > BackwashDuration Then
                    AnnotationText = "DOI Backwash Elapsed Exit"
                    isAnnotationPending = True
                    Exit While
                End If

            End While

            temp.Stop() 'stop the timer
            SetCheckBox(chkBackwash, False) 'turn off the backwash pump
            SetCheckBox(chkMainPump, prevState) 'return main pump to previous state

            SetButtonState(btnDOIBackwash, True) 'enable manual backwashes
            SetLabel(lblDOIBackwash, "Manual Backwash Only")
            SetButtonText(btnDOIBackwash, "Schedule Backwash")
```

```
        Else

            SetButtonState(btnAFBackwash, False) 'disable manual backwashes
            SetButtonText(btnAFBackwash, "Backwashing")

            Dim prevState As Boolean = chkPumpUFMain_isChecked 'get pump state
            SetCheckBox(chkPumpUFMain, False) 'turn off main pump
            SetNumeric(numUF, 100) 'open pressure control valve 'TODO: eliminate this
line to expedite system recovery
            SetCheckBox(chkPumpBackwash, True) 'turn on backwash pump

            'start timer
            Dim temp As New Stopwatch
            temp.Start()

            While True

                SetLabel(lblAFBackwash, "Backwashing - " & ((BackwashDurationAF -
temp.ElapsedMilliseconds) / 1000).ToString("0") & " s") 'update label
                Sleep(100) 'wait a bit

                'check for cancellation
                If ProcessWorker.CancellationPending Then
                    AnnotationText = "Backwash Cancel Exit"
                    isAnnotationPending = True
                    Exit While
                End If

                'check for completion
```

```
            If temp.ElapsedMilliseconds > BackwashDurationAF Then
                AnnotationText = "AF Backwash Elapsed Exit"
                isAnnotationPending = True
                Exit While
            End If

        End While

        temp.Stop() 'stop the timer
        SetCheckBox(chkPumpBackwash, False) 'turn off the backwash pump
        SetCheckBox(chkPumpUFMain, prevState) 'return main pump to previous state

        SetButtonState(btnAFBackwash, True) 'enable manual backwashes
        SetLabel(lblAFBackwash, "Manual Backwash Only")
        SetButtonText(btnAFBackwash, "Schedule Backwash")

    End If

    isBackwashPending = False 'backwash completed

End Sub

'run cleaning on UF membrane (AF only)
Private Sub RunCleaning()

    SetButtonState(btnAFClean, False) 'disable manual cleaning
    SetButtonText(btnAFClean, "Cleaning")

    Dim pumpstate As Boolean = chkPumpUFMain_isChecked 'record current pump state
```

```
        SetCheckBox(chkPumpUFMain, False) 'turn off main pump
        SetCheckBox(chkSelectorUFFiltrate, True) 'switch to cleaner
        SetCheckBox(chkSelectorUFRecirculate, True) 'switch to waste
        Dim UF_old As Decimal = numUF_Value 'record old valve position
        SetNumeric(numUF, 0) 'close UF valve

        ValveDelay() 'wait for valves to move

        SetCheckBox(chkPumpBackwash, True) 'turn on backwash pump
        Dim temp As New Stopwatch
        temp.Start()

        While True

            SetLabel(lblAFClean, "Cleaning - " & ((CleanDuration -
temp.ElapsedMilliseconds) / 1000).ToString("0") & " s") 'update label
            Sleep(100) 'wait a bit

            'check for completion
            If temp.ElapsedMilliseconds > CleanDuration Then
                AnnotationText = "UF Cleaning Elapsed Exit"
                isAnnotationPending = True
                Exit While
            End If

            'check for cancellation
            If ProcessWorker.CancellationPending Then
                AnnotationText = "UF Cleaning Cancel Exit"
                isAnnotationPending = True
```

```
            Exit While
        End If

    End While

    temp.Stop() 'stop the timer
    SetCheckBox(chkPumpBackwash, False) 'turn off backwash pump
    SetCheckBox(chkSelectorUFRecirculate, False) 'switch to recirculation
    SetCheckBox(chkSelectorUFFiltrate, False) 'switch to filtrate
    SetNumeric(numUF, UF_old) 'open path to feed

    ValveDelay() 'wait for valves to move

    SetCheckBox(chkPumpUFMain, pumpstate) 'return pump to previous state

    SetButtonState(btnAFClean, True) 'enable manual cleaning
    SetLabel(lblAFClean, "Manual Clean Only")
    isCleaningPending = False
    SetButtonText(btnAFClean, "Schedule Cleaning")

End Sub

'run cleaning on NF membrane (AF only)
Private Sub RunNFCleaning()

    AnnotationText = "NF Cleaning Started"
    isAnnotationPending = True

    SetButtonState(btnNFClean, False) 'disable manual cleaning
```

```
        SetButtonText(btnNFClean, "Cleaning")

        SetNumeric(numNF, 100) 'open valve wide open

        'start a timer
        Dim temp As New Stopwatch
        temp.Start()

        While True

            SetLabel(lblNFClean, "Cleaning - " & ((NFCleanDuration -
temp.ElapsedMilliseconds) / 1000).ToString("0") & " s") 'update label
            Sleep(100) 'wait a bit

            'check for cancellation
            If NFWorker.CancellationPending Then
                AnnotationText = "NF Cleaning Cancel Exit"
                isAnnotationPending = True
                Exit While
            End If

            If temp.ElapsedMilliseconds > NFCleanDuration Then
                AnnotationText = "NF Cleaning Elapsed Exit"
                isAnnotationPending = True
                Exit While
            End If

        End While
```

```
        isNFCleaningPending = False
        temp.Stop() 'stop the timer
        SetButtonState(btnNFClean, True) 'enable manual cleaning
        SetLabel(lblNFClean, "Manual Clean Only")
        SetButtonText(btnNFClean, "Schedule Cleaning")

    End Sub

    'delays for valve movement, while keeping user updated
    Private Sub ValveDelay()

        Dim sw As New Stopwatch
        sw.Start()
        While True

            SetLabel(lblAFClean, "Valves Moving - " & ((ValveTime -
sw.ElapsedMilliseconds) / 1000).ToString("0") & " s") 'update label
            Sleep(100)

            'check for completion
            If sw.ElapsedMilliseconds > ValveTime Then
                AnnotationText = "Valve Elapsed Exit"
                isAnnotationPending = True
                Exit While
            End If

            'check for cancellation
            If ProcessWorker.CancellationPending Then
                AnnotationText = "Valve Cancel Exit"
```

```
            isAnnotationPending = True
            Exit While
        End If

    End While
    sw.Stop()

    SetLabel(lblAFClean, "") 'update label

End Sub

'setup routines for new filtration process
Private Sub TabChanged()

    'stop background workers (if busy)
    ProcessWorker.CancelAsync()
    PrefilterWorker.CancelAsync()
    NFWorker.CancelAsync()
    AllStop()

    'change filtration mode and start Arduino
    If TabControl1.SelectedIndex = TabControl1.TabPages.IndexOf(tabDOI) Then
        isDOI = True
        SetupEnvironment()
        SplitContainer1.Panel2.Show()
        SplitContainer1.SplitterDistance = 310
    ElseIf TabControl1.SelectedIndex = TabControl1.TabPages.IndexOf(tabAF) Then
        isDOI = False
        SetupEnvironment()
```

```
            SplitContainer1.Panel2.Show()
            SplitContainer1.SplitterDistance = 390
        ElseIf TabControl1.SelectedIndex = TabControl1.TabPages.IndexOf(tabStart) Then
            SplitContainer1.Panel2.Hide()
            SplitContainer1.SplitterDistance = 303
        End If

    End Sub

#End Region

End Class
```