Vindum Pump DLL Documentation RevJ 21July2021

Note 1: Older deprecated syntax is not shown in this document. Request RevG of this document if you need the deprecated syntax.

Note 2: New content and changes from prior version are highlighted in yellow.

CONTENTS

Section 1: Overview & Requirements 2
Section 2: Important Note Regarding DLL Changes from Prior Versions 2
Section 3: Interfacing to the DLL - C# Code Examples 3
Code Example - Declarations 3
Code Example - Initialization 4
Code Example - Enable/Disable Pump Communications 5
Code Example - Pump Event Handler 6
Code Example – Accessing Pump Data 6
Code Example - Sending Operational Commands to a Pump (new, preferred syntax) 7
Section 4: Pump DLL Function Calls – Full List 7
Function Calls 9
Section 5: Pump Events 16
Events that signal availability of new data 17
Events that signal comState change 17
Events related to Command Status 17
Section 6: Pump Data Items 18
State Variables 18
Section 7: Variable Updates 22

SECTION 1: OVERVIEW & REQUIREMENTS

The VPS_COM.DLL file is a .NET assembly DLL that manages the interface between Vindum pumps and a PC-based control program. VPS_COM handles the low-level communication to the pump hardware providing the software developer with a simple API (Application Programming Interface) for pump control and monitoring.

Requirements

- Microsoft Windows 7 or later.
- Microsoft .NET Version 4.5 or later.
- VPS_COM.DLL version 1.3.6 or later.
    - To check the VPS_COM_DLL version you are using, right-click on the file in Windows Explorer and selecting 'Properties', then selecting the 'Details' tab and noting 'File Version').

****

SECTION 2: IMPORTANT NOTE REGARDING DLL CHANGES FROM PRIOR VERSIONS

The DLL has been updated and the structure of some functions has changed from earlier versions of the DLL. Now, all cylinder-specific functions sent to the pump have an interface that requires the cylinder index be passed in. This is a change from the older DDL, which used separate functions for each cylinder. In the new syntax, the cylinder index is:

0 for Cylinder A

1 for Cylinder B

Pump IDs remain 1-based, so pumpID for pump 1 = 1.

Functions that have changed are still available to be called using the original syntax, so either the new syntax or the older structure will still work. If you would like documentation showing all the changes affected by this update, please contact Vindum Engineering (support@vindum.com).

****

SECTION 3: INTERFACING TO THE DLL - C# CODE EXAMPLES

Code Example - Declarations

```
// 'Pumps' is a public class of the DLL that provides all
//  the functionality required for full control of Vindum pumps.
//  PumpDef is a public list of defined pumps

Pumps pumps;          // Declare a Pumps object.
PumpDef[] pumpList;   // Declare a list of pump definitions

// PumpDef is defined in the DLL - here is a listing for reference
// along with enum PumpType and SerialPortParameters
// PumpDef, PumpType, and SerialPortParameters should not be
// re-declared.
 public class PumpDef
 {
     public bool defined;         // Not used by DLL
     public bool enabled;         // Must be true if enabling pump
     public PumpType pumpType;    // pumpType.VindumSerial currently
                                  // supported
     public uint pumpID;          // Must specify unique ID for pump
     public uint uiPosition;      // Not used by DLL
     public SerialPortParameters comParams;   // See below
     public string pumpName;      // Not used by DLL

    // Saved information used only for VPware features. The arrays are of size 2.
    // These can be ignored by non-VPware users.
```

```
        public bool[] startInVolumeMode;
        public int lastCylinderStarted;
        public double[] volumesAtLastStart;
        public int lastAutoReturnRateTime;
}

 public enum PumpType
 {
        VindumSerial,  // RS-232 Serial, or USB-Serial adapter
        VindumUSB      // Native USB currently not supported
 }

 public class SerialPortParameters
 {
        public string port;         // eg. "COM2"
        public int baudRate;        // 38400 standard
        public Parity parity;       // Parity.None
        public int dataBits;        // databits = 8
        public char[] termChars;    // Not used by DLL
 }
```

Code Example - Initialization

```
// The following code initializes the interface between the control
// program and the Vindum Pump DLL. Each step is required.

// Instantiate the Pumps object
pumps = new Pumps();

// Subscribe to events sent from the DLL, passing the Event Handler
// function as an argument
```

```
pumps.pumpEvent += new Pumps.PumpEventDelegate(PumpEventHandler);

// Call the InitPumps function of the DLL with a list of defined
// pumps as an argument.  The list can optionally be null, with pumps
// added for communication using the EnablePumpCom function described
// below.  If one or more pump definitions are specified in the list,
// then the DLL will attempt to enable communications with the pumps
// using the supplied pump definition parameters. Note that the
// DLL only uses the following fields in the PumpDef structure:
//     Enabled
//     comParams
//     pumpType
//     pumpID
// The rest of the fields in PumpDef (defined, uiPosition, pumpName,
// etc.) are there for the convenience of applications - VPware uses
// them, but they're not required by the DLL.
// The InitPumps function call needs to be made even if pumpList is
// null. (ddeNameSuffix) is a string that is appended to "VPware" to
// create the name of the DDE server, and can safely be set to null
// or an empty string
pumps.InitPumps(pumpList, "");
```

Code Example - Enable/Disable Pump Communications

```
// To enable communications with a pump not already enabled during
// InitPumps function call, use the EnablePumpCom function which
// takes an argument of type PumpDef. See the comments for the
// InitPumps function for how to populate the PumpDef structure
pumps.EnablePumpCom(pumpDef);
```

```
    // Communications to a pump are terminated with DisablePumpCom
    // Only pumpDef.pumpType and pumpDef.pumpID are required for
    // the DisablePumpCom call
    pumps.DisablePumpCom(pumpDef);
```

Code Example - Pump Event Handler

```
    // An event handler of the following form must be included to
    // handle the pump events sent from the DLL
    // See Section 5, 'Pump Events', below for details on pump events
    void PumpEventHandler(PumpEvent pumpEvent, uint index, CommandType    command,
double value, string message)
        {
            switch (pumpEvent)
            {
                case PumpEvent.PumpFound:
                    // Handle the communication established event
                    break;

                case PumpEvent.StatusDataUpdate:
                    // Handle the new status data available event
                    break;

                //… etc.
            }
        }
```

Code Example – Accessing Pump Data

```
    // The Pump Event Handler should access pump data, as needed, following Events
```

```
// that signal availability of new pump data.  These are StatusDataUpdate,
// InitDataUpdate, ErrorUpdate, LowLevelUpdate.  See Section 5, 'Pump Events'
// for specifics.
// For example, the pressure in Cylinder A of the pump with ID = 1
// can be accessed as shown below
```

New, Preferred Access Syntax

```
double pressureValue = pumps[1].cylinder[0].pressure
```

Code Example - Sending Operational Commands to a Pump (new, preferred syntax)

```
pumps.SetMode(1, 0, PfRateDel); // Set Pump ID=1, Cylinder A to Paired Rate Deliver
pumps.SetRate(1, 0, 20.0);  // Set Pump ID=1 Cylinder A Rate to 20.0 ml/min
pumps.StartCylinder(1, 0);  // Start Pump ID=1 with Cylinder A active
```

****

SECTION 4: PUMP DLL FUNCTION CALLS – FULL LIST

Note on units – internal units are PSI (pressure), ml (Volume) and ml/min (rate). C# UI provides UnitsManager for alternate pressure and rate units.

- **bar = psi * 0.0689476**
- **kpa = psi * 6.89475728**
- **Mpa = psi * 0.006894757**
- ml/hr = ml/min * 60

```
public bool IdExists(uint key) // Returns existence of configured pump with given key (ID)
```

```
public void InitPumps(PumpDef[] pumpList)  // Initialization call to DLL, argument is a list of Pump Definitions. Success or failure of pump initialization communicated via events, PumpFound and PumpNotFound

public string SetUpdateInterval(int milliseconds) // Sets the interval, in milliseconds, between “Read Status” commands to the pump. This value must be between 50 and 10,000.

public int GetUpdateInterval() // Returns the update interval, in milliseconds

public void InitPumps(PumpDef[] pumpList, string ddeNameSuffix)   // Initialization call to DLL, arguments are a list of Pump Definitions and a suffix for the DDE server name. The name suffix is used to support multiple instances of the application using the DLL. Success or failure of pump initialization communicated via events, PumpFound and PumpNotFound


public void EnablePumpCom(PumpDef pumpDef)  // Enable single pump,

public void DisablePumpCom(PumpDef pumpDef) // Disable single pump

public void Shutdown() // Stops the “read status” timer and stops sending events to the event handler

public void SimulatePumpEnable(uint pumpID, int msInterval)




public class PumpDef

{

   public bool defined;  // True – show on screen; False – hide from view

   public bool enabled;   // True – communicating or attempting communication; False – Placeholder, not communication or attempting communication.


   public PumpType pumpType;  // See PumpType definition below

   public uint pumpID;    // Intended to be unique pump ID provided by the pump itself, as of Nov15, 2015 using on-screen position for pumpID

   public uint uiPosition;  // On-screen position, 1-4 for four pumps

   public SerialPortParameters comParams;

   public string pumpName; // Name defined for pump and displayed in UI


   // Saved information used only for VPware features. The arrays are of size 2. These can be ignored by other users.
```

```
    public bool[] startInVolumeMode;
    public int lastCylinderStarted;
    public double[] volumesAtLastStart;
    public int lastAutoReturnRateTime;
}

public class SerialPortParameters
{
    public string port;
    public int baudRate;            // currently 38400
    public Parity parity;
    public int dataBits;
    public char[] termChars;
}

public enum PumpType
{
    VindumSerial,  // As of Nov 15, 2015 all pumps are VindumSerial    including pumps with direct USB connection since Virtual Serial ports are used for USB.
    VindumUSB, // Possible future raw-USB interface
}
```

Function Calls

```
public string StartCylinder(uint pumpID, int cylinderIndex)  // In individual modes starts specified cylinder (A = 0, B = 1); in paired modes starts pump with specified cylinder delivering.

public string AutoStartCylinder(uint pumpId, int cylIndex)    // Pump will try and start selected piston.  If the piston is out of position (eg, too far extended), pump will try and start the other piston.    If both pistons are out of position (such that continuous flow would not be possible if pumping started with pistons in current position), then one
```

piston will be repositioned before starting the pump. We call this “SMART-START” or “Auto-Start”).

```
public string StartCylinderVolumeMode(uint pumpId, int cylIndex , double volume)    // Pump will start cylinder selected.    If cylinder is out of position, an error message will be displayed.  Pump will then stop when volume amount is reached.

public string AutoStartCylinderVolumeMode(uint pumpId, int cylIndex, double volume)

// Same as AutoStartCylinder above, but Pump will then stop when volume amount is reached.


public string StopCylinder(uint pumpID, int cylinderIndex) // In individual modes stops specified cylinder; in paired modes stops pump.


// Valve Operation Commands

public string ValveOpenFill(uint pumpID, int cylinderIndex) // Opens Cylinder Fill Valve

public string ValveCloseFill(uint pumpID, cylinderIndex)    // Closes Cylinder Fill Valve

public string ValveOpenDeliver(uint pumpID, cylinderIndex)  // Opens Cylinder Deliver Valve

public string ValveCloseDeliver(uint pumpID, cylinderIndex) // Closes Cylinder Deliver Valve


// User Warnings and Restrictions Required for Valve Operations

// 1. Warning on Attempting to Open both Fill and Deliver valve on a Cylinder

// 2. Warning on Attempting to Open Fill or Deliver Valve when Cylinder Pressure // is above 1000 psi

// 3. Valve condition cannot be changed if cylinder is running.



public string SetExtend(uint pumpID, int cylinderIndex)  // Set Cylinder direction to Extend

public string SetRetract(uint pumpID, int cylinderIndex)  // Set Cylinder direction to Retract


// Notes on Set Direction Commands

// - Only active on a stopped pump, the pump will return an error if direction //command sent while running.

// - Only active when pump is in an Individual mode; ignored if in a Paired mode
```

```
public string SetMode(uint pumpID, int cylinderIndex, PumpMode modeA)  // Sets pump mode
for specified cylinder, or for pump if in paired mode.


public enum PumpMode
    {
        Manual = 0,                 // Currently not used
        IndRate,                    // Independent Rate (EventLog mode=1; VPware=IR)
        IndPressure,                // Independent Pressure Deliver (mode=2; IPD)
        IndRateCycled,              // Independent Rate Cycled (mode=3; IRDC)
        IndPressureCycled,          // Independent Pressure Cycled (mode=4; IPDC)
        IndRateRecCycled,           // Independent Rate Receive Cycled (mode=5; IRRC)
        RecirculationCompensation,  // Recirculation-Compensation (mode=6; RIPD)
        IndPressureRecCycled,       // Independent Pressure Receive Cycled (mode=7; IPRC)
        Mode8,                      // not used
        Mode9,                      // not used
        ModeA,                      // not used
        ModeB,                      // not used
        ModeC,                      // not used
        ModeD,                      // not used
        ModeE,                      // not used
        ModeF,                      // not used
        PfGear,                     // Paired Rate Deliver-Geared (mode=16; PRDG)
        PfRateDel,                  // Pulse-free Rate Delivery (mode=17; PRD)
        PfPressureDel,              // Pulse-free Pressure Delivery (mode=18; PPD)
        PfRateRec,                  // Pulse-free Rate Receive (mode=19; PRR)
        PfPressureRec,              // Pulse-free Pressure Receive (mode=20; PPR)
        PfPressureBiDir,            // Pulse-free Pressure Bi-Directional (mode=21; PPBD)
        PfDeltaPressureDel,         // Paired Delta Pressure Deliver (mode=22; PDPD)
        RecirculationFlow,          // Recirculation-Flow (mode=23; RPRD)
```

```
        NotSet
    }

public string SetRate(uint pumpID, int cylinderIndex, double rate)  // Sets rate (ml/min) for specified cylinder, or for pump in Paired Modes

public string SetMaxRate(uint pumpID, double rate) // Sets the Maximum User Settable Rate (ml/min).  This allows the user to specify a maximum settable rate that is less than the intrinsic rate capability of the pump.

public string SetPressure(uint pumpID, int cylinderIndex, double pressure) // Sets pressure (psi) for specified cylinder, or for pump in Paired Modes

public string SetSafetyPressure(uint pumpID, int cylinderIndex, double pressure) // Sets safety pressure (psi) for specified cylinder, or for pump in Paired Modes.  Pump will automatically stop if safety pressure is exceeded.  Maximum settable safety pressure is the lesser of the intrinsic maximum pump pressure and the user settable max pressure. Default is zero, so safety pressure needs to be set before pump can be operated.

public string SetLowPressureOpen(uint pumpID, double pressure)

// Sets the pressure (psi) at which depressurization will stop and the fill valve will open for refill of cylinder.  Default and minimum is 250 psi.

public string SetMaxPressure(uint pumpID, double pressure)

// Sets the Maximum User Settable Pressure (psi).  This allows the user to specify a maximum settable pressure that is less than the intrinsic pressure capability of the pump.

public string ResetVolume(uint pumpID, int which)

// Resets pump volume to zero:

// 1 = Cylinder A, 2 = Cylinder B, 4 = Cumulative Volume, 8 = Cylinder A Cumulative Volume, 16 = Cylinder B Cumulative Volume, (cylinder cumulative volumes valid only in independent cycled modes), 31 = All Volumes

public string ResetZeroOffset(uint pumpID, int cylinderIndex)

// Resets specified cylinder Pressure Transducer Offset to Zero.  The Fill Valve for the
```

cylinder must be open and connected to atmospheric pressure before invoking this command.

```
public string SetReturnRateMult(uint pumpID, double multiplier)
```

// Sets the Cylinder Return Rate **Multiplier,** which is the factor by which the rate of a filling cylinder exceeds that of the cylinder during fluid delivery. For example, a setting of 1.5 results in a fill rate of 15 ml/min if the set rate is 10 ml/min. The return rate must be higher than the deliver rate to allow for repressurization time once the cylinder is filled.

```
public string SetMinReturnRate(uint pumpID, double minReturnRate)
```

// minReturnRate units are ml/min

```
public string SetTAU(uint pumpID, int rampStepSize)
```

// Currently unused, do not send any values down

```
public string SetClosedPropGain(uint pumpID, int cylinderIndex, int value)
```

// Servo Pressure Gains: Specified cylinder Closed-Valve Proportional Gain

```
public string SetClosedDiffGain(uint pumpID, int cylinderIndex, int value)
```

// Servo Pressure Gains: Specified cylinder Closed-Valve Differential Gain

```
public string SetOpenPropGain(uint pumpID, int cylinderIndex, int value)
```

// Servo Pressure Gains: Specified cylinder Open-Valve Proportional Gain

```
public string SetOpenDiffGain(uint pumpID, int cylinderIndex, int value)
```

// Servo Pressure Gains: Specified cylinder Open-Valve Differential Gain

```
public string SetPressureGain(uint pumpID, int cylinderIndex, double value)
```

// Set specified cylinder Pressure Transducer Gain Value

```
public string ResetErrors(uint pumpID, double value)
```

```
// Reset Pump latched errors

// Set “value” to zero to clear the errors.


public string SetPumpType(uint pumpID, int value)      // Pump top assembly must be changed to match value sent.  3.5K = 35, 6.5K = 65, 12K = 120, 20K = 200, 25K = 250


public string SetTransducerType(uint pumpID, int value)// Transducer must be changed to match value sent.  3 = 300, 1.5K = 15, 3.5K = 35, 6.5K = 65, 12K = 120, 15K = 150, 20K = 200, 25K = 250


public string SetUserOptionMode(uint pumpID, int value)// 0 = not installed, 1 = Dual pump control with start/stop and pressure/rate control based on pump mode (See VP-User Guide for details about Option Module configuration=1).  2 = Lock out control.  3 = Delta pressure transducer installed… See VP-User Guide for details on each mode.


public string SetDeltaPressure(uint pumpID, double value)     // Sets target pressure (psi) for Delta pressure mode.


public string SetDeltaZeroOffset(uint pumpID)   // Zero the transducer pressure (psi).


public string SetDeltaPressureGain(uint pumpID, double value) //    Gain    value    for calibrating delta pressure transducer.  Use a calibrated pressure source to set the gain value.


public string SetDeltaPressurePropGain(uint pumpID, double value)    // Servo Pressure Gain setting for delta-pressure transducer: Open-Valve Proportional Gain.


public string SetDeltaPressureDiffGain(uint pumpID, double value)    // Servo Pressure Gain setting for delta-pressure transducer: Open-Valve Differential Gain.




public string SetLockConfigureBits(uint pumpID, int value)


public enum LockConfigureBits

{
```

```
    OpenValveGains = 0x01,
    ClosedValveGains = 0x02,
    PressureTransducerGain = 0x04,
    DeltaPressureMode = 0x08,
    RampStepSize = 0x10,
    MinReturnRate = 0x20,
    LowPressureOpen = 0x40,
    ReturnRateMultiplier = 0x80,
    MaxRate = 0x100,
    MaxPressure = 0x200,
    TransducerType = 0x400,
    PumpType = 0x800,
    OptionModule = 0x1000,

    PasswordCorrect = 0x4000,
    PasswordBitValid = 0x8000
};
```

```
public string SetPassWord(uint pumpID, int value)

public string CheckPassWord(uint pumpID, int value)

public string ResetPassWordValidBit(uint pumpID

public string SetAutoReturnRate(uint pumpID, int value)       //  Value  is  in  seconds.
Cylinder will retract at a rate that will allow the cylinder this much time to pressurize
after filling.  Value cannot be less than 10 (seconds).

public void SetPumpStrokeLimit(uint pumpID, int limit)  // Value is the percentage of the
stroke to use, with a minimum of 30%. Sending a value greater than 100 will exit pump stroke
limit mode.
```

```
public string SetSaveSafetyPressureMode(uint pumpID, bool saveSafetyPressureOnPump)
public static string PausePumpCommunication()
public static string RestartPumpCommunication()

*****************************************************************************
```

SECTION 5: PUMP EVENTS

```
public enum PumpEvent
    {
        DefinedPumpNotFound,   // Failed to establish com with a defined pump
        ReInitPumpComFailed,     // Failed to re-initialize communications following a timeout.
        PumpComDisabled,    // Confirmation that Pump com disabled
        StatusDataUpdate,   // New pump status data available
        InitDataUpdate,     // New pump setpoint data available
        ErrorUpdate,        // New pump error information available
        LowLevelUpdate,     // New pump configuration data available
        CommandSuccess,     // A pump command was processed without error
        CommandError,        // A pump command was not processed – Error given in message parameter.
        CommandRejected,    // Pump command not sent to pump from DLL
        CommandPurged,      // A queued pump command was purge, likely due to communication loss.
        LogicError,         // Unexpected programmatic logic error
        PumpFound,          // Pump Communication Established
        PumpNotFound,
        PumpComTimeout,
        PumpComLost,        // Timeout on previously established pump comm
        PumpCom_ReEstablished,  // Pump Communication Re-established.
        PumpResponseError,  // Error reading pump message
```

```
        DdeError,           // DDE protocol error
        DdeCommandSuccess,  // DDE command was successful
        DdeCommandError,    // DDE response to application command
        ApplicationWarning, // Transaction Timer required restart
        FillErrorAtStart,    // Pistons in positions that don't allow start in current mode
        OpcCommandSuccess,
        OpcCommandError,
        GearedModeStartError,
        PumpCommStopped,
        PumpCommRestarted,
    }
```

Events that signal availability of new data

```
        StatusDataUpdate,
        InitDataUpdate,
        ErrorUpdate,
        LowLevelUpdate,
```

Events that signal comState change

```
        PumpFound,
        PumpNotFound,
        PumpComLost,
        PumpCom_ReEstablished,
        UnexpectedComPortClosed,
        PumpComTimeout
```

Events related to Command Status

```
        CommandSuccess,
```

```
        CommandError,
        CommandRejected,
        CommandPurged,
```

****

SECTION 6: PUMP DATA ITEMS

State Variables

```
        public PumpComState comState; // Defined in PumpComState enum


          public enum PumpComState
           {
               Not_Defined,
               Initializing,
               ComActive,
               ComActive_Retrying,
               ComLost_Retrying,
               HardwareFault
           }

        public PumpCommand activeCommand  // Information about the outstanding command to
the pump


        public class PumpCommand
        {
            public CommandType commandType;
            public double commandValue;
```

```
            public long syncCode;      // 0 if no sychronization with other pumps required,
otherwise use unique identifier (eg. DateTime,UtcNow.Ticks)
        }


        public enum CommandType
        {
                ReadStatus,
                ReadErrors,
                ReadInit,
                ReadLowLevel,
                StartCylA,
                StartCylB,
                StopCylA,
                StopCylB,
                SetModeCylA,
                SetModeCylB,
                SetRateA,
                SetRateB,
                SetMaxRate,
                SetPressure,
                SetPressureA,
                SetPressureB,
                SetLowPressureOpen,
                SetSafetyPressureA,
                SetSafetyPressureB,
                SetMaxPressure,
                ResetVolume,
                ResetVolumeA,
                ResetVolumeB,
                SetExtendA,
                SetRetractA,
```

```
SetExtendB,
SetRetractB,
ValveOpenFillA,
ValveCloseFillA,
ValveOpenDeliverA,
ValveCloseDeliverA,
ValveOpenFillB,
ValveCloseFillB,
ValveOpenDeliverB,
ValveCloseDeliverB,
ZeroPressureOffsetA,
ZeroPressureOffsetB,
SetReturnRateMultiplier, // Was SetReturnRate
SetMinReturnRate,
SetTAU,
SetClosedPropGainA,
SetClosedPropGainB,
SetClosedDiffGainA,
SetClosedDiffGainB,
SetOpenPropGainA,
SetOpenPropGainB,
SetOpenDiffGainA,
SetOpenDiffGainB,
SetPressureGainA
SetPressureGainB
ResetErrors,
PumpType,
TransducerType,
StartCylAVolume,
StartCylBVolume,
```

AutoStartCylA,

AutoStartCylB,

AutoStartCylAVolume,

AutoStartCylBVolume,

SetUserOptionModule,

ResetPump,

SetDeltaPressure,

SetDeltaZeroOffset,

SetDeltaPressureGain,

SetDeltaPressurePropGain,

SetDeltaPressureDiffGain,

SetLockConfigureBits,

SetPassWord,

CheckPassWord,

ResetPassWordValidBit,

Reserved,

SetAutoReturnRate,

Reserved,

Reserved,

Reserved,

Reserved,

Reserved,

Reserved,

Reserved,

Reserved,

Reserved,

Reserved,

Reserved,

Reserved,

SetPumpStrokeLimit,

```
SetPressurizationTime,
SetCompressionVolume,
Reserved,
Reserved,
Reserved,
SetSaveSafetyPressureMode,
}
```

****

SECTION 7: VARIABLE UPDATES

Important Note: All cylinder-specific data items received from the pump are stored in the data structure shown in this document. The data structure used in the prior DLL versions is still functional but is deprecated and may not be supported in the future.

```
public class CylinderInfo
    {
        // Status Items

        public PumpMode pumpMode;
        public PumpDirection direction;
        public int pumpCondition;
        public bool isRunning;
        public bool isActive;
        public bool isVolumePumping;
        public bool isFilling;
        public int measuredTurns;
        public double pressure;   // Internal units (psi)
```

```
        public double rate;       // Internal units (ml/min)
        public double volume;     // Internal units are milliliters
        public double cumulativeVolume;  // Internal units are milliliters

        public ValveState fillValve;
        public ValveState deliverValve;

        public double setPressure;    // Internal units (psi)
        public double safetyPressure; // Internal units (psi)

        public double setRate;        // Internal units (ml/min)
        public double setVolume;      // Internal units are milliliters

        public int closedPropGain;
        public int closedDiffGain;
        public int openPropGain;
        public int openDiffGain;

        public int TAU;

        public int error;
        public int lError;

        public double pressureGain;
    }
```

This structure is stored in the Pump class:

```
  public CylinderInfo[] cylinder;
```

Cylinder-specific errors also use the new structure shown below:

```
public class CylinderErrors
{
    public bool safetyPressure;
    public bool fetCurrent;
    public bool motorStall;
    public bool getToPressureVol;
    public bool servoLimitReached;
    public bool transducerLowVoltage;
    public bool positionSensorStuckOn;
    public bool missingPositionSensor;


    // Cylinder latched errors
    public bool lSafetyPressure;
    public bool lFetCurrent;
    public bool lMotorStall;
    public bool lGetToPressureVol;
    public bool lServoLimitReached;
    public bool lTransducerLowVoltage;
    public bool lPositionSensorStuckOn;
    public bool lMissingPositionSensor;
}
```

This data structure is stored in the PumpError class:

```
   public CylinderErrors[] cylinderErrors;
```

Variables Updated prior to StatusDataUpdate Event

```
public double rateOutput;          // Valid in Paired Modes
public double volumeCumulative;    // Cumulative Volume - Valid in Paired Modes
public int errorsPresent;          // Low-level error indicator – should be private
public double pressureOutput;      // Valid in Paired Modes
public int codeVersion;            // Pump firmware
public double deltaPressureOutput; // Valid in Paired Delta Pressure mode only
public int PCBtemperature;
public int FETtemperature;
public int PWSupplytemperature;
public enum PumpDirection
   {
      Extending,
      Retracting,
      Unknown
   }

public enum ValveState
   {
       Open,
       Closed,
       Unknown
   }
```

In cylinder[cylinderIndex]:

```
public double rate;          // Internal units (ml/min)
public int pumpCondition;    // Low level variable – To be removed in future
```

```
public PumpDirection direction;
public bool isRunning;       // Cylinder running status
public bool isActive;        //
public bool isVolumePumping; // Cylinder is pumping in volume mode
public bool isFilling;       // Cylinder is filling
public int measuredTurns;      // Cylinder position: 0-fully retracted, 635-fully extended
public double volume;        // Internal units are milliliters
public double volumeCumulative
public ValveState fillValve;
public ValveState deliverValve;
public double pressure;      // Internal units (psi)
```

Variables Updated prior to InitDataUpdate Event – Setpoint Values

```
public double maxPressure; // See SetMaxPressure command
public double maxRate;     // See SetMaxRate command
public int optionModuleType;
public byte autoReturnRateTime;  // Seconds
public int hardwareVersion;
public int setDeltaPressure;  // Delta pressure set point
public double lowPressureFillOpen;
public UInt16 lockConfigureBits;
public int cylinderPercentage;  // Pump stroke limit percentage
public bool SaveSafetyPressureOnPump;
```

In cylinder[cylinderIndex]:

```
public PumpMode pumpMode;
```

```
public double setPressure;    // Internal units (psi)
public double safetyPressure; // Internal units (psi)
public double setRate;        // Internal units (ml/min)
public double setVolume;      // Not yet active – future use
```

Variables Updated prior to LowLevelUpdate Event

```
public double returnRateMultiplier; // was returnRate, See SetReturnRateMult cmd
public double returnRateMin;        // See SetMinReturnRate command

public double deltaPressureGain;
public double deltaPressurePropGain;
public double deltaPressureDiffGain;

public int pumpType;          // Gives MaxPressure divided by 100
public int transducerType;
public int pumpSerialNumber;
public int TAU;
```

In cylinder[cylinderIndex]:

```
public double pressureGain;  // Pressure transducer gains
public int closedPropGain;
public int closedDiffGain;
public int openPropGain;
public int openDiffGain;
public int TAU;
```

Variables Updated prior to ErrorUpdate Event

```
public PumpError pumpError;


public class PumpError
{
    // Common errors – Apply to both Cylinders A and B
    public bool comError;         // Pump received a bad message ignore
    public bool emergencyStop;    // Not currently used - ignore
    public bool powerSupplyTemp;  // Power supply temp exceeded – pump stopped
    public bool fetTemp;   // Motor drive temp exceeded – pump stopped
    public bool powerSupplyVoltage;  // Power supply voltage fault
    public bool pressureCable;   // Pressure cable fault – pump stopped
    public bool valveCable;  // Valve cable fault – pump stopped
    public bool transducerABOffset; //Maximum offset between CylA and CylB pressure transducers exceeded – warning error, Transducer calibration needed.
    public bool flasherror; // call Vindum for tech support
    public bool dualCylExtendError;  // Both cylinders are fully extended.  Raise return rate multiplier setting, so as to give piston more time to retract in order to avoid this error.
    public bool dualCylRetractError; // Both cylinders are fully retracted.  Raise return rate multiplier setting, so as to allow the piston more time to extend, in order to avoid this error.
    public bool pumpTransducerMismatch;     // Check transducer configuration
    public bool openCloseGainsMismatch;     // Warning, gain values of cylinder are not the same.  Please check configuration screen for desired values.
    public bool userOptionModuleInUse;      // VPware is trying to override option module control.  Disable option module for VPware to control pump.
    public bool userOptionModuleAnalogInputTooHigh;       // Detected input voltage greater than 3.3V, which is the max allowable input voltage on the Option Module port.
    public bool autoReturnRateTooHigh;      // Cylinder will retract at maximum speed but pressurize time will be less then time desired (ie, <10seconds).
    public bool sensorCableError;
```

```
    public bool rateLimitedError;
    public bool pressureLimitedError;
    public bool pressureLimitedError;

    // Common latched errors – active until cleared by ResetErrors command
    public bool lComError;
    public bool lEmergencyStop;
    public bool lPowerSupplyTemp;
    public bool lFetTemp;
    public bool lPowerSupplyVoltage;
    public bool lPressureCable;
    public bool lValveCable;
    public bool lTransducerABOffset;
    public bool lflasherror;
    public bool lDualCylExtendError;
    public bool lDualCylRetractError;
    public bool lCommShortBytes;
    public bool lPumpTransducerMismatch;
    public bool lOpenCloseGainsMismatch;
    public bool lUserOptionModuleInUse;
    public bool lUserOptionModuleAnalogInputTooHigh;
    public bool lAutoReturnRateTooHigh;
    public bool lSensorCableError;
    public bool lRateLimitedError;
    public bool lPressureLimitedError;
```

In PumpError class:

```
    CylinderErrors cylinderErrors;
```

Code Example to access Cylinder A safety pressure errors status:

```
cylinderErrors[0].safetyPressure


public class CylinderErrors
{
    public bool safetyPressure; // Safety Pressure exceeded – cylinder stopped
    public bool fetCurrent; // Motor Drive current for cylinder exceeded
    public bool motorStall; // Cylinder A Motor Stall
    public bool getToPressureVol; // Cylinder failed to get to pressure in the
allow amount of volume.
    public bool servoLimitReached; // Cylinder tried to servo pass end point
    public bool transducerLowVoltage;
    // Cylinder latched errors
    public bool lSafetyPressure;
    public bool lFetCurrent;
    public bool lMotorStall;
    public bool lGetToPressureVol;
    public bool lServoLimitReached;
    public bool lTransducerLowVoltage;
    public bool positionSensorStuckOn;
    public bool missingPositionSensor;
}
```