

# **Biodiesel Grade Oil Extraction Automation and Control**

James N. Long, M.S.

*Associate Professor, Computer Systems Engineering Technology, Oregon Institute of Technology*

Marc A. Timmerman, Ph.D.

*Associate Professor, Electrical Engineering and Renewable Energy, Oregon Institute of Technology*

Drew Loika, B.S.

*Research Associate, Oregon Institute of Technology*

*August 24, 2009*

## **Abstract**

This report describes the design, development, and implementation of a closed-loop control system for regulating the moisture of grain feedstock to a biodiesel auger press system in a medium-scale commercial setting. The mechanical-fluidic dynamics of the auger process requires a minimum water content to avoid seizing and processing down time. Excess water in the feedstock causes production problems with other aspects of the process. Moisture content of the feedstock is measured using an electromagnetic sensor. A computer based control algorithm activates a water valve that adds additional moisture to the feedstock using real-time feedback calculation. The control algorithm features high-level internet-capable user interface features for remote process and quality control monitoring. The key scientific contribution of the work is the development of a calibration process for the on-line electromagnetic sensor that relates its output to calibration data from a pre-existing off-line quality control system. This study essentially converted an off-line manual quality control process to an on-line real-time automated process without affecting the validity or robustness of the process.

## **Introduction**

The production of Biodiesel requires a source of vegetable based oil. Green Fuels of Oregon, a company based in Klamath Falls, has set up a mid-size biodiesel plant at Liskey Farms where rapeseed is being grown, harvested, stored, and processed into biodiesel [1]. In Fall, 2007, Rick Walsh of Green Fuels of Oregon approached the researchers with a problem he was having at his seed oil presses. Inconsistent moisture content of canola seed feed stock into the seed oil presses was causing presses to clog bringing biodiesel production to a halt and requiring costly manual intervention to clean out the presses. This problem was an excellent candidate for automated computer control.

The biodiesel plant is considered a mid size production facility, producing in the range of 150,000 gallons biodiesel a year. The plant was set up on a small budget to create biodiesel from locally grown oil seed in the Klamath Basin. In light of the small budget, lower cost seed presses were purchased for oil extraction. The cost savings was approximately \$30,000 for purchase of oil presses manufactured in China compared to more expensive presses of German manufacture. The lower cost presses have much less tolerance for variance in oil seed moisture content than the higher cost presses. If the moisture content in the seed gets

too low, the presses heat up and clog. When a press clogs, the entire pressing operation must be shut down so the presses can be taken apart and cleaned out. The operation takes one to two hours of manual labor. The accumulated cost of this operation would eventually offset the savings realized in purchase of less expensive presses. This problem is due to the fundamental fabrication of the lower-cost presses and cannot be addressed by modifying the presses themselves.

The objective of this Technology Development Proposal is to design, implement and test a low cost control system for on-line moisture control in the processing of biodiesel grade vegetable oil. The end goal is enhanced mid-size biodiesel production through a standardized and adaptable computer based moisture control system. This will increase efficiency of farm site biodiesel production of locally distributed biodiesel fuels. The control process developed is quite general and can easily be ported to other factories. The process is not equipment or grain specific.

Biodiesel plant operation at Liskey Farms involves pressing canola and camolina seed in two presses; a six and a ten ton press. Initial work in this process indicates seed moisture content is highly important in press operation and oil yield from the seed [2]. Current moisture control is done by a manual process where seed sample is periodically taken from the processing cycle, moisture content is checked in an off-line moisture meter, water injection is next adjusted to either increase or decrease seed moisture content. This manual process is problematic resulting in lower oil yield from the pressing of the seed and periodic processing line shut-down due to clogged equipmen. Both of these conditions are attributed to inadequate moisture control in the seed.

The control problem is simple: given a desired moisture percentage, spray water on the seed to increase moisture before it is run through the press. The problem of oil seed moisture control has been solved many times; however, there are no low cost control units available based on current “off-the-shelf” components specifically targeted at pressing of oil seed. The unique scientific contribution of this work is relating the off-line process to the on-line process in a robust and reliable way.

## **Seed Characteristic Control Function Analysis**

This section of the report contains a detailed technical description of the design of the control system for the project. Figure 1 depicts a schema of the overall system.

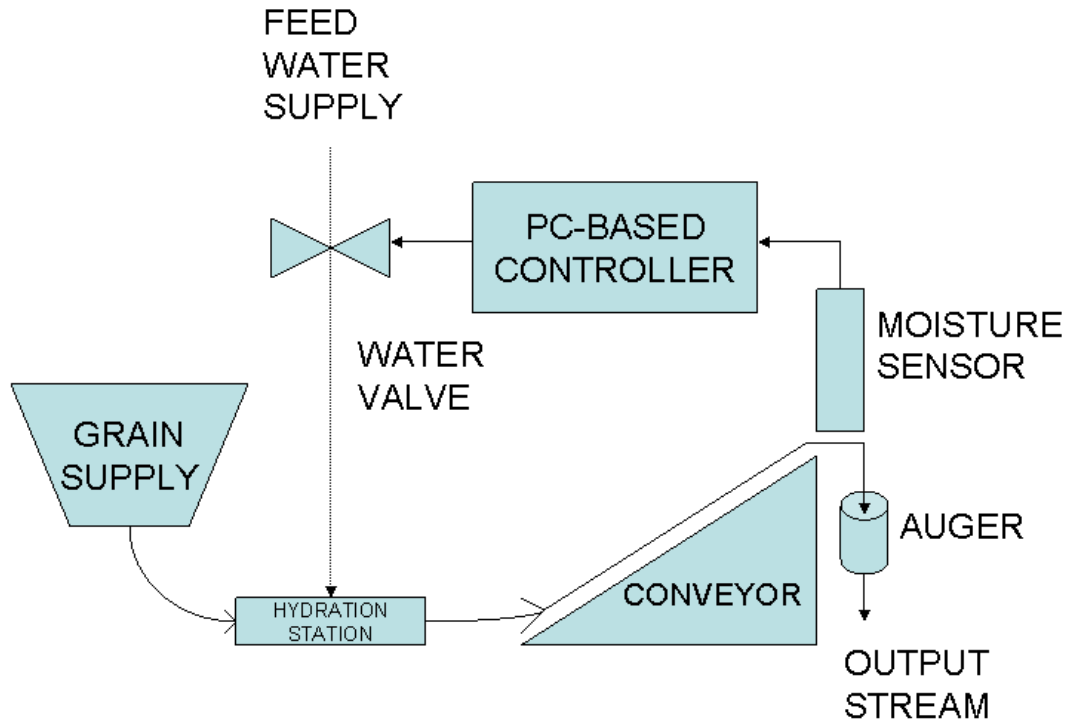


Figure 1 – Schema of Oil Seed Press System

Grain feedstock is held in a large hopper and is gravity fed into a hydration station. At the hydration station water is metered through a control valve into the grain feedstock. Once hydrated the grain mixture is placed on an inclined conveyor. At the top of the conveyor an electromagnetic sensor measures the percentage moisture of the grain mixture. The grain mixture then gravity feeds into the auger system for further processing. Figure 2 depicts the block diagram of the control system.

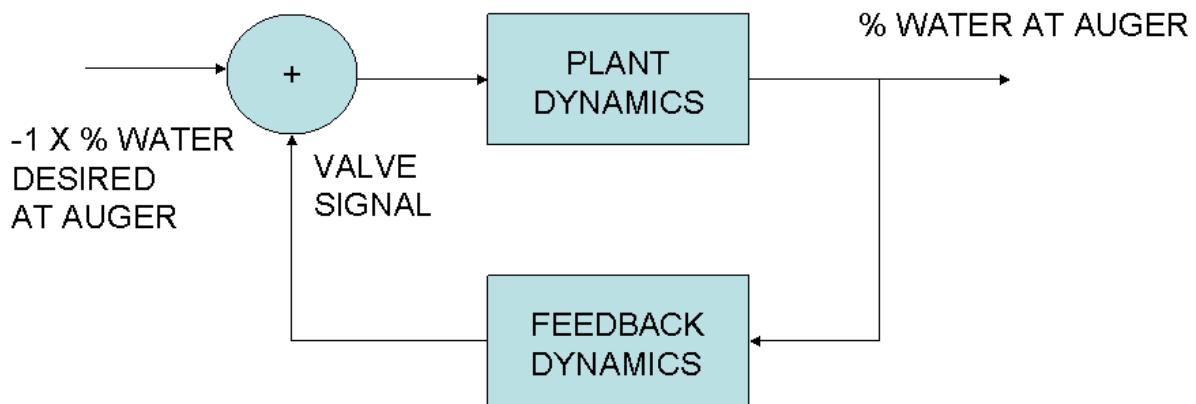


Figure 2 – Control System Block Diagram

The control system is a simple one-loop regulator. The controlled variable is the percentage moisture in the grain mixture delivered into the auger assembly. The desired percentage moisture is determined by the dynamics of the auger. The grain mixture is essentially a bi-phase solid-liquid interface with fluid and thermal transfer characteristics determined by the percentage of moisture. Insufficient moisture causes

seizure of the auger through a friction process requiring complete shutdown of the system for clean-out. Excess moisture causes problems in the further processing of the biodiesel.

The control equations are derived as follows:

- (1) The characteristics of the moisture sensor were determined experimentally by direct mass titration and mass measurement. A standard grain moisture sensor operating on a capacitance principle was used for calibration. It is important to note that the calibration is dependent on the exact feedstock used. Through linear regression analysis of the data set a slope intercept calibration curve was derived as follows:

$$\text{Voltage Output} = 0.965 \times \% \text{ grain moisture} - 7.68 \text{ VDC-offset}$$

- (2) Using the dynamic mass balance method the dynamics of the hydration process are as follows:

$$\dot{m}_{\text{water at auger}} = \dot{m}_{\text{grain feedstock}} + \dot{m}_{\text{water from valve}}$$

As the control variable of interest is water mass as a percentage basis this equation can be re-written as follows:

$$\dot{m}_{\text{dry grain}} \bullet \% \text{ water}_{\text{grain at auger}} = \dot{m}_{\text{dry grain}} \bullet \% \text{ water}_{\text{grain at input}} + \dot{m}_{\text{water from valve}}$$

The dry grain mass is conserved due to minimal grain leakage and the water evaporation loss is assumed to be minimal. This result can be manipulated to yield the needed water flow through the valve:

$$\dot{m}_{\text{water from valve}} = \dot{m}_{\text{dry grain}} \bullet \% \text{ water}_{\text{grain at auger}} - \dot{m}_{\text{dry grain}} \bullet \% \text{ water}_{\text{grain at input}} = K_{\text{valve}} \bullet V_{\text{valve}}$$

The valve constant is dependent on water pressure. Using the vendor's literature at an average water pressure of 5 psi yields this result:

$$K_{\text{valve}} = \frac{\Delta \dot{m}_{\text{water}}}{\Delta V_{\text{dc}}} \approx \frac{.6 \text{ gpm}}{24 \text{ vdc}} \bullet \frac{8 \text{ lbf} / \text{gal}}{1 \text{ gpm}}$$

- (3) The dynamics of the conveyor system are simply that of a delay of about 1 minute:

$$H(s)_{\text{conveyor}} \approx 1e^{-60s}$$

Again, this assumes minimum grain leakage and minimum water evaporation.

- (4) A complication may arise at this point in the control equation synthesis as the feed-forward dynamics of the conveyor have a non-minimum-phase zero. Technically this is a non-collocated control problem. However, the slow speed of the conveyor is essentially matched by the slow speed of the data acquisition process and for all practical purposes the plant behaves like a unity gain system except of course for a dimensional scaling factor. Classically these types of problems are usually solved using PID (proportional-integral-derivative) controllers. Experimentally reasonable good results have been obtained with straightforward proportional

control. This works because the grain in the hopper is very homogeneous and once a successful constant is found the system requires little tuning.

To derive an initial control equation, measurements of the change in detected seed moisture given addition of a measured weight of water were taken in the laboratory. The results of these measurements are shown in Figure 3 through 6. These results were obtained through experimentally processing numerous samples by both the on-line and off-line techniques, obtaining a set of numerical data from the samples, and using a linear-curve-fit-regression analysis to relate the data. As can be seen in Figure 6 the linear curve fit gives excellent agreement to the original data.

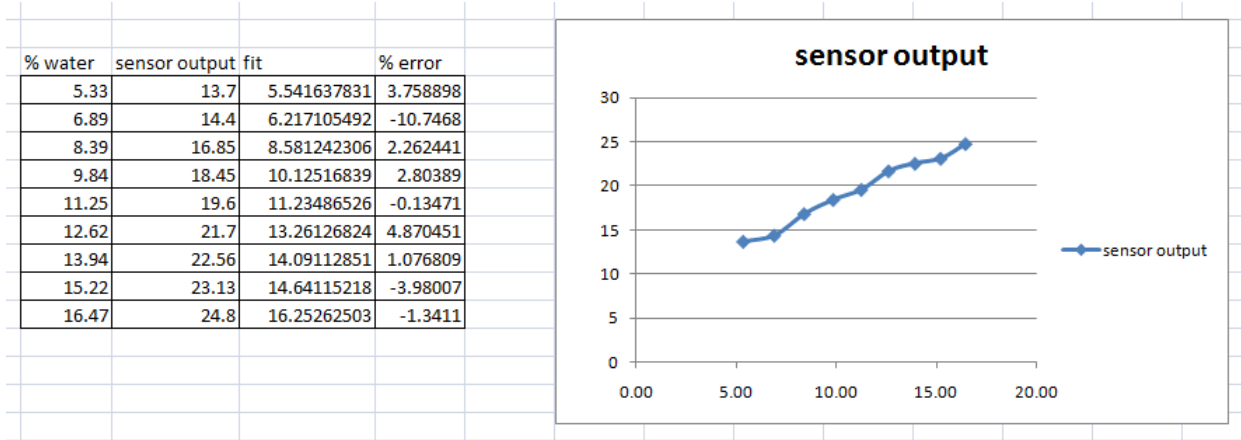


Figure 3 – Measured Sensor Output

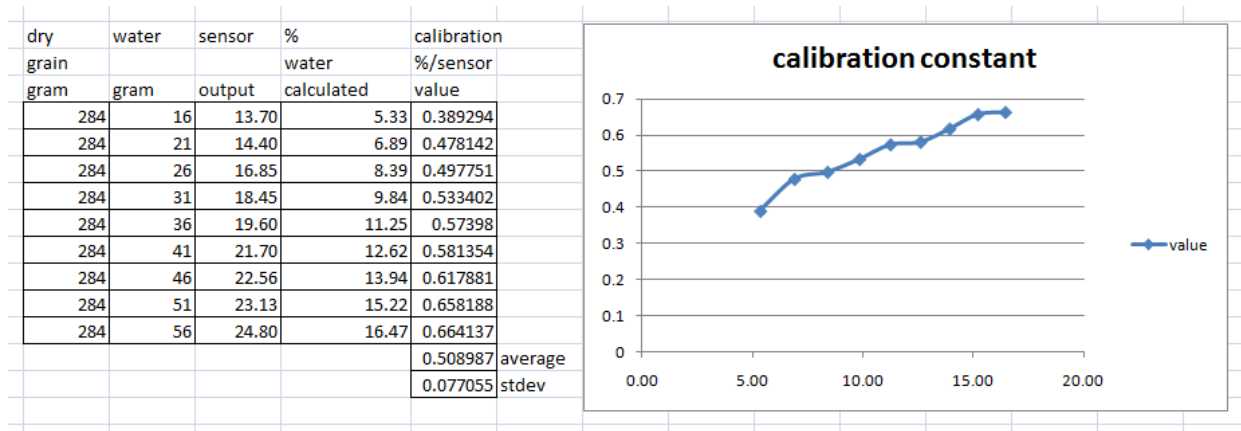


Figure 4 – Calibration Constant Derivation

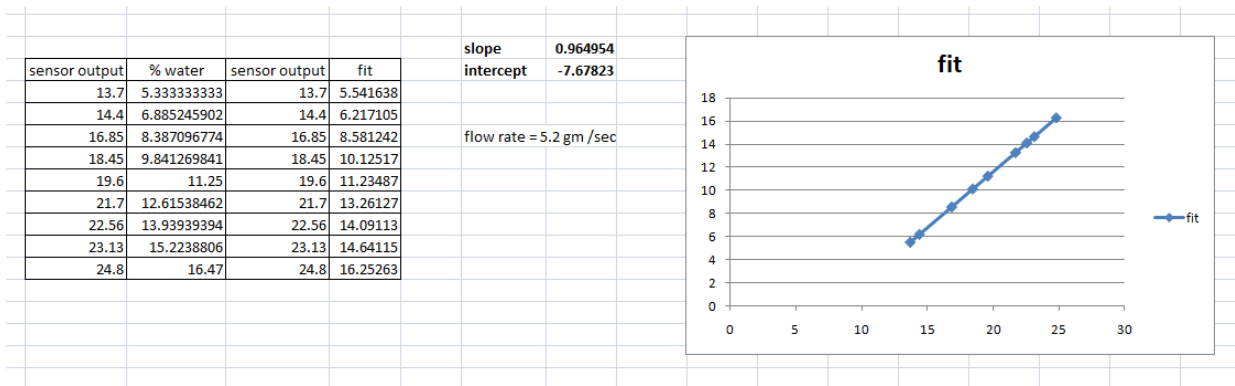


Figure 5 – Calibration Equation

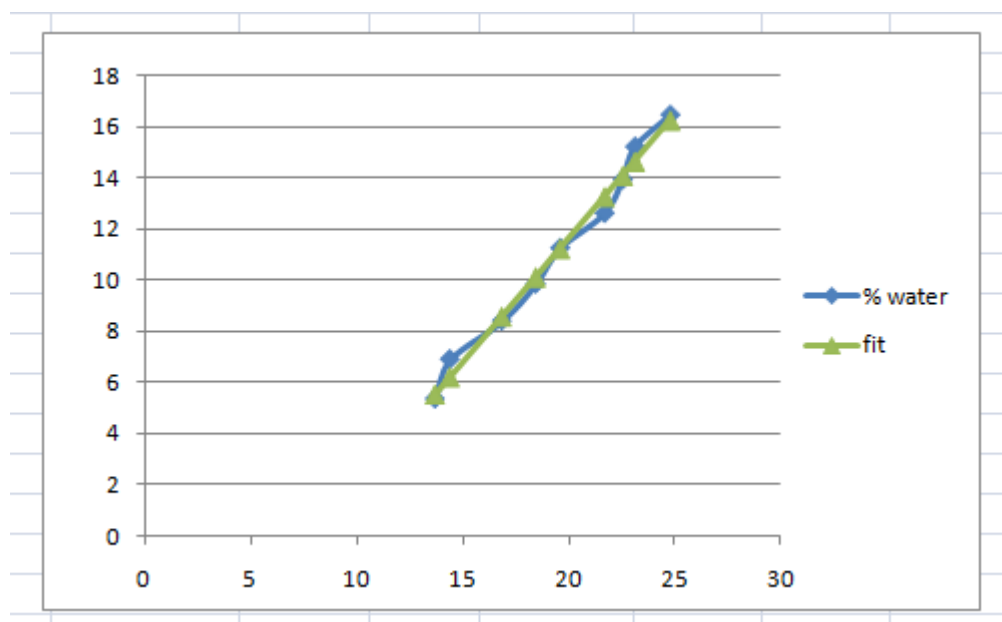


Figure 6 – Actual vs. Fitted Data

## Sensor Characterization

The moisture sensor was selected based on cost and performance. The ideal sensor was low cost, but would be able to obtain internal seed moisture readings. The sensors also had to be readily available and have software interface support libraries provided by the manufacturer.

Commercial moisture content sensors are based on two physical principles; optical material properties, typically in the infrared frequency range, and electromagnetic material properties at various frequencies depending on the type of sensor (capacitive or microwave). Each type of sensor has physical issues such as sample penetration depth, accuracy, reliability, and cost. An extensive survey of available sensor products was undertaken.

The first class of sensors studied were capacitance sensors used for moisture control in irrigation applications. These sensors proved to be inexpensive and able to detect external moisture of the canola

seed; however, they could not see “into” the seed to get overall moisture content for the purpose of pressing and oil extraction. The margin of error for these sensors also made them unfit as a solution.

The second class of sensor studied were those of a class using near infrared sensing (NIR). The NIR sensors were capable of internal grain moisture sensing. All had an acceptable margin of error ranging to  $\pm 0.01\%$ . However, the NIR sensors started at a price of \$10,000 making them an expensive solution.

The third sensor technology evaluated was microwave sensing. The microwave sensor also had a good margin of error and was capable of sensing overall seed moisture content. Unfortunately, it too was relatively expensive, also coming in around the \$10,000 mark.

The fourth type of sensor considered was a large plate capacitance sensor. These sensors had the desired characteristics of:

- Software development library support.
- RS232 Serial communication support.
- A margin of error  $\pm 0.1\%$
- Ability to sense overall seed moisture.
- Relatively low cost of ~\$3500.00

The sensor specified for the project was the Hydronix Hydro-Probe II shown in Figure 7.

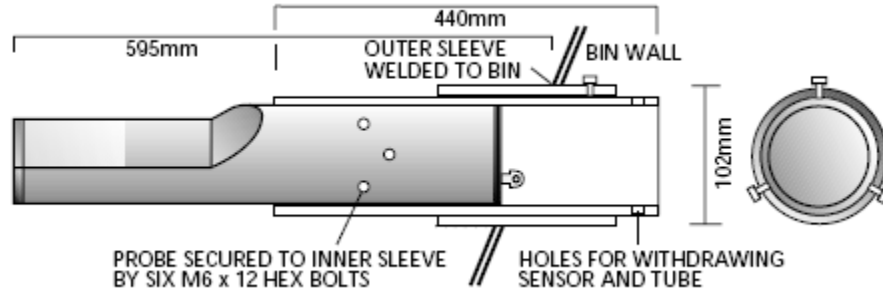


Figure 7 – Hydroprobe II Flat Plate Capacitance Sensor - (<http://www.hydronix.com/products/hydroprobe.php>) .

## Water Injection Device Characterization

Upon sensing of the moisture content of the seed, if the moisture is too low, water needs to be injected into the grain stream before it travels up the conveyor into the hopper where the sensor is located. The valve needs to be electronically controlled allowing the computer to cycle the valve on and off for analog adjustment of the water flow into the grain stream. By experimentation, the press operators were able to manually adjust water flow for successful control of seed moisture content. The manual process utilized by the press operators was to check the seed moisture content every ten to fifteen minutes, then either turn up or turn down the valve on a hose spraying into the seed stream. The valve needed to allow for the system to emulate this process.

To effect automation of the manual process of loosening or tightening a hose valve, a characterized control valve (CCV) was chosen allowing the computer to send signals to the valve to open or close. The specific valve chosen was a Belimo B2 Series, 2 Way Characterized Control Valve. This valve is shown in Figure 8.



Figure 8 – Belimo B2 Series 2 Way control valve.

( [https://www.belimo.us/bellib/Characterized\\_Control\\_Valve/B2B\\_LRB24\\_3\\_T\\_S.pdf](https://www.belimo.us/bellib/Characterized_Control_Valve/B2B_LRB24_3_T_S.pdf) )

The control valve cost is ~\$230.00.

In addition to the control valve, an interface is introduced to allow the control computer to send a signal to the valve causing it to either open or close. The valve operates on a bang-bang principle so the control strategy was equivalent to pulse width modulation or PWM. The details of relating the desired output to the duty cycle of the valve are taken care of in the software. The valve operation requires 24 V to be applied to one of two lines. 24V on line A, 0V on line B will cause the valve to open. 24V on line B, 0V on line A will cause the valve to close. To keep in line with the I/O requirements of the moisture sensor, a digital output device is required that will interface to the control computer via RS232. The device chosen

was a Phidget Interface Kit 0/0/8. This device provides Small Signal Relays with eight relay outputs, rated at 250VAC. The computer interfaces through this device over USB thus fitting into the interface mode of the Hydronix sensor. The Phidget 0/0/8 is shown in Figure 9.



Figure 9 – Phidget 0/0/8 USB interface kit.

( [http://www.phidgets.com/products.php?product\\_id=1017](http://www.phidgets.com/products.php?product_id=1017) )

The cost of the Phidget 0/0/8 is \$90.00.

## Control Computer Characterization

The control computer was to be a single purpose device tasked with running the moisture control system. The Hydronix Hydroprobe II uses a standard RS232 interface and the Phidget uses a USB connection. These interface requirements force the computer to have a USB connectors where the Hydroprobe will be interfaced through a USB to RS232 adapter. The software interface libraries provided for the Hydroprobe and the Phidget were based on a software suite called ".Net 3.5."<sup>1</sup> This requires the operating system installed on the control computer to support the .Net platform which leads to the necessity of a Microsoft based platform. Since the device is single purpose, the computing power of the system does not need to be great allowing selection from lower CPU power devices. Following is a list summarizing the control computer requirements:

- Moderate processing power
- Support of Microsoft technologies
- Support of .Net libraries
- Support of a graphical user interface
- USB support

---

<sup>1</sup> The term .Net 3.5 (dot-net-three-point-five) refers to a large package of computer interface software tools. This package facilitates the development of interface and communications software applications.

- Small footprint

To fill these requirements, one of the low cost Net Books was purchased. These machines are relatively inexpensive and have full support of Microsoft development tools and libraries. The specific Netbook chosen is an Eee PC from Asus. This computer is shown in Figure 10.



Figure 10 – Asus Eee PC

The cost of the Eee PC is \$378.00.

### **Software Development Platform**

Software development is done on the native Eee PC platform. The operating system run on the Eee PC is Windows XP. Interfaces provided with the Hydro-Probe II and the Phidget 8/8/8 are .Net 3.5 compliant and were distributed with C++ and C# examples showing programmatic use of the interface libraries that build and execute under the Visual Studio software development suite.<sup>2</sup> Because C# provides a rich programming environment for creating graphical user interfaces, the C# programming language was chosen.

The software development environment needs to support:

- C# programming language
- .Net 3.5 framework
- Windows XP, Vista, and 7

---

<sup>2</sup> C++ (C-plus-plus) and C# (C-sharp) are specific versions of the C computer programming language. C# was chosen for compatibility reasons to the .Net framework.

- Windows Graphical User Interface API

Given these factors, Visual Studio 2008 was chosen as the Integrated Development environment, hosted on Windows XP. Although the Eee PC is not an extremely powerful machine, code could be developed on more powerful development environments, then moved to the Eee PC for debugging on the actual hardware on which the system would be deployed. The software developed for the system is detailed in Appendix A.

## Results

The system has been developed and deployed in an Alpha release mode. To integrate the control system into the seed oil presses, the press system had to be physically modified to insert the sensor and locate the water injection station. The control computer could be set any place around the presses within range of the sensor cabling, valve control cabling, and the press operator. As can be seen in the figures the factory environment is unfriendly due to atmospheric particulate contamination and heat. For this reason and also due to electrical and occupational safety code requirements the final system will be enclosed in a NEMA-standard industrial enclosure apparatus. The current installation is shown without the enclosure for illustration purposes only. Alpha system installation is shown in Figures 11 through 16.



Figure 11 – An overall system view.



Figure 12 – The Presses



Figure 13 – Installation of the sensor in the seed stream – outside seed hopper.



Figure 14 – Installation of the sensor in the seed stream – sensor head inside the hopper.



Figure 15 – Water injection location.

A test run was performed with the system configured to sample every 15 seconds. The results of this run are shown in Figure 16. This graph is from the comma separated value file. The window of data was taken after the system had stabilized. The Y-Axis shows the calculated seed moisture percentage. The X-Axis is the time series, every tick representing 15 seconds wall clock time.

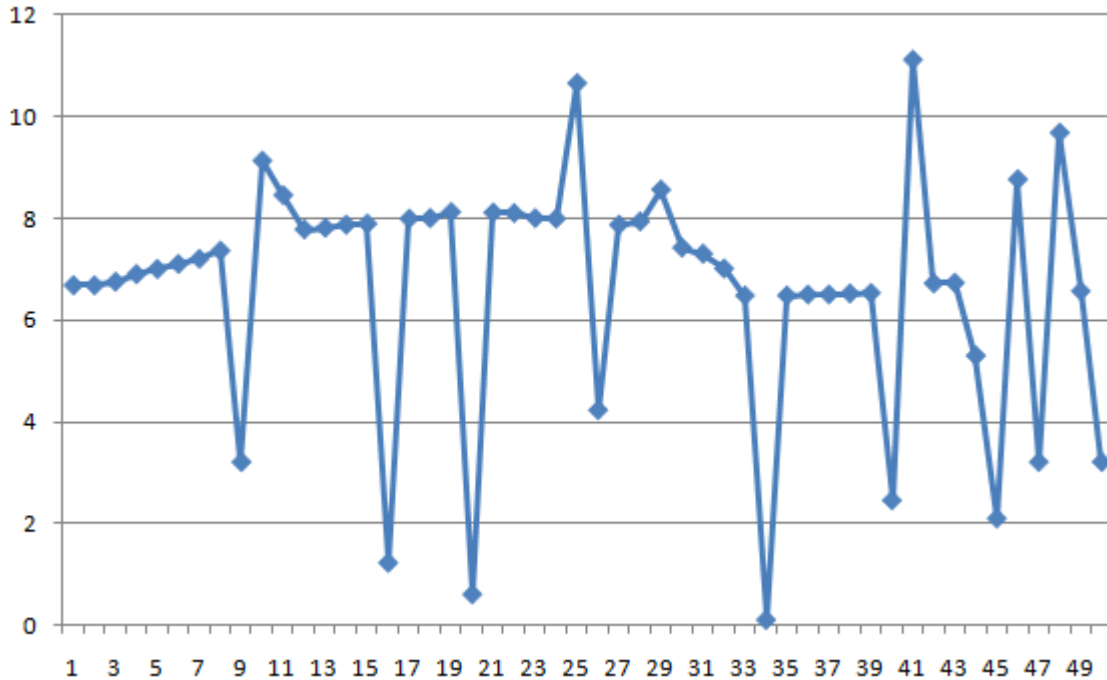


Figure 16 – Test run time series.

The test run results were good for first cut at control of the seed moisture. Some points to notice in the data snapshot of Figure 16:

1. The biodiesel plant where the system is deployed has a fair amount of electrical noise. The noise in the signal received from the moisture probe increased as the distance from the probe to the control computer increased. The sensor signal noise impacted the control algorithm causing water to be injected into the seed stream at unnecessary times. Eventually, the signal noise rendered the control system ineffective. Due to the random nature of this noise it can easily be taken care of by adding an averaging or integral term to the control algorithm. Future work will involve the relatively simple and straight forward task of migrating from a proportional controller (P) to a proportional-integral-derivative (PID) controller. This does not involve any hardware issues--just additional experimentation and software development. Based on well-known principles of control theory this does not represent a major technical or scientific issue.
2. Over time, the control system tended to cause the moisture content of the seed to oscillate beyond acceptable error limits of  $\pm 0.5\%$ . This oscillation is due to an errant noise input and not to a system right-hand-plane pole. Again, it can be resolved easily by developing a PID control algorithm.

## Conclusions

As a first release of the control system, the results are very promising. The alpha release control tests; although not good enough to use as a production unit, did show that further work in sensor noise

reduction and tweaking of the control function would provide a low cost seed moisture control system for use in lower grade seed oil presses. The overall cost of deploying this system is summarized in Table 1.

<b>Component</b>	<b>Cost</b>
Hydro-Probe II and Accessories	\$3500.00
ASUS Eee PC Control Computer	\$400.00
Phidget 0/0/8 I/O Module	\$90.00
Belimo B2 Valve	\$230.00
<b>Total:</b>	<b>\$4220.00</b>

Table 1 – Overall cost of seed oil moisture control system.

With an equipment and software cost of \$4220.00, the cost of deploying the control system to enhance performance of lower cost seed oil presses is far less than the expenditure of high end manufacture seed oil press.

The researchers are pleased that the outcome of the project has met expectations of characterizing, producing and developing an initial prototype of a control system capable of offsetting the lower performance specifications of low cost seed oil presses. This work can be expanded to further develop and deploy a fully functional moisture control system for the purpose of biodiesel grade oil production.

The next step for this project will be development and testing of a PID control software algorithm.

## **Future Work**

The initial release of the Alpha version of the seed moisture control system has laid a good foundation for further development and refinement of a control system. Following are recommendations for further work to be done.

### **Refine Control Algorithm**

The system as derived works reasonably well. The key assumptions are (1) a low sampling rate relative to the time dynamics of the conveyor delay and (2) a very homogeneous grain feedstock. These assumptions may not work well if the system is scaled to a much larger size. For example, if the grain supply came from different geographical locations, different cultivars of one species or different species of grain, then the assumption of homogeneity may not hold. Also, if it were desired to process grain at a much faster rate the time dynamics of the conveyor belt could lead to non-collated-control issues. Some possible areas for further work might include:

- (1) Developing a set of tables of control constants for different species of grains and different cultivars of species.
- (2) Developing a look-up-table of different calculated or interpolated control constants based on measured moisture characteristics of feed stocks.
- (3) Implementing a true PID controller inclusive of non-proportional terms to improve the time dynamics of the system. This would be crucial to scaling this result to faster through-put systems.
- (4) Using artificial intelligence techniques to develop learning / self-tuning controllers. This problem would lend itself well to analysis by fuzzy systems techniques.

## Refine Control Equipment

The Hydro-Probe , Phidget, and Belimo valve are good solutions for this problem space. The control computer choice could be refined through spending more on an industrial grade, touch screen Windows PC. The operating environment in the processing plant is high in dust and oil. Such a combination can render a keyboard useless relatively quickly. A touch screen with protective cover would better fit the harsh, oil and dust rich environment. Another drawback of the Eee PC is difficulty in securing the unit without development of a specialized mounting platform. A panel mount touch screen control computer running Windows would increase the cost of the installed system by \$500.00 to \$1000.00; however, it would prove to be simpler to mount, operate, and maintain. This would require rewriting the UI code to include touch screen elements for operation.

The noise issue needs to be addressed. A noise canceling circuit can be developed to include in line with the Hydro-Probe. This would filter out the undesirable spikes in the moisture content signal. An effort like this would require prototype fabrication and testing of the sensor characteristics in line with the noise canceling circuit. The system would then need to be calibrated to compensate for any differences.

## Final Budget

Budget Area	Budgeted Amount	Expended	Remaining Amount
Researcher Pay (Total)	\$12,800.00	\$12,800.00	\$0.00
Unclassified	\$8,004.00	\$8,004.00	\$0.00
Other Unclassified	\$1,840.00	\$1,840.00	\$0.00
Research Associate	\$2,835.00	\$2,835.00	\$0.00
OPE	\$4041.00	\$4041.00	\$0.00
Laboratory Supplies	\$0.00	\$116.03	-\$116.03
Equipment	\$5,459.00	\$5,073.00	\$385.82
Seed Stock	\$3,200.00	\$0.00	\$3,200.00
Management Consulting Services	\$1000.00	\$1000.00	\$0.00
S&S Expenses	\$0.00	\$106.90	-\$106.90
Totals:	\$26,500.00	\$23,522.93	\$2977.07

## References

[1] Andrew Chiasson, March 2007, Greenfuels of Oregon: Geothermal Energy Utilization in Biodiesel Production, Geo-Heat Center, Geo-Heat Center Quarterly Bulletin, Oregon Institute of Technology, Klamath Falls, Oregon.

[2] Erik Ferchau, November 2000, Equipment for Decentralized Cold Pressing of Oil Seed, Folkecenter [sic] for Renewable Energy, Hurup Thy, Denmark.

# Seed Core Processor

---

*Experimental Software Application Final Report*

*Drew Loika  
2009.08.02*

## Table of Contents

\_Toc239052917

Table of Figures .....	21
Overview .....	22
Details .....	23
Business Objects .....	23
Persistence .....	25
Business Logic .....	27
Presentation.....	28
Services .....	31
Usage.....	34
Troubleshooting.....	35

## Table of Figures

Figure 1 - N-Tier Design of Control Module .....	22
Figure 2 - Class Diagram of Control Module Business Objects .....	23
Figure 3 - Control Module Persistence Tier Diagram .....	25
Figure 4 - Control Module Business Logic Class Diagram .....	27
Figure 5 - Main UI: Chart and Moisture Control .....	28
Figure 6 - Main UI: Select Active Sensor .....	29
Figure 7 - Main UI: Logging and Live Data.....	29
Figure 8 - Control Module Presentation Project Elements .....	30
Figure 9 - Service Host and Valve Interaction .....	31
Figure 10 - Moisture Service Interaction .....	33

## Overview

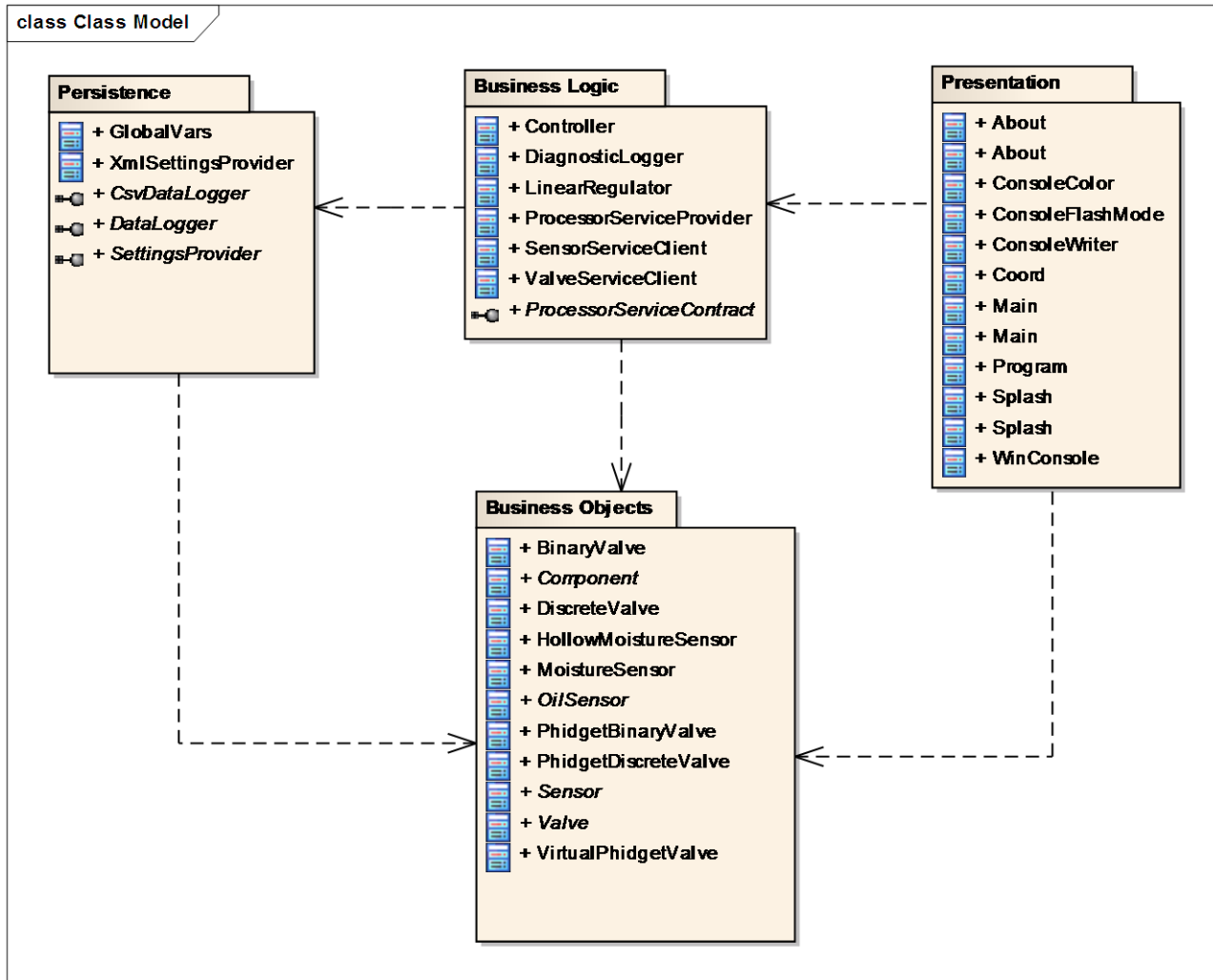


Figure 1 - N-Tier Design of Control Module

This application was created for the purpose of monitoring the moisture content of seed being pressed for its oil and to adjust that moisture content for optimal oil output from the pressing process by adding varying amounts of water. The application consists of two major components, Control and Interface. Control is a traditional n-tier implementation using business objects, persistence, business logic, and presentation. It was intended for this module to be hosted on a central, more traditional PC workstation. It communicates via web service with the second major component of the application, Interface, which is responsible for communicating with and controlling one or more sensors and water valves in their proprietary formats and exposing them in a universal format via web service to the Control module. One or more Interface instances are intended to be hosted on lightweight hardware such as a .NET capable imbedded device or small PC such as a netbook. It was anticipated that each site could have different seed pressing configurations

requiring flexibility in number and placement of sensors and valves, while control and monitoring would still be best from a single, more capable location. Alternatively, the entire application is capable of being hosted in one location on lightweight hardware for minimalist scenarios. It was this last configuration under which real-world testing of the application has taken place to date.

## Details

### Business Objects

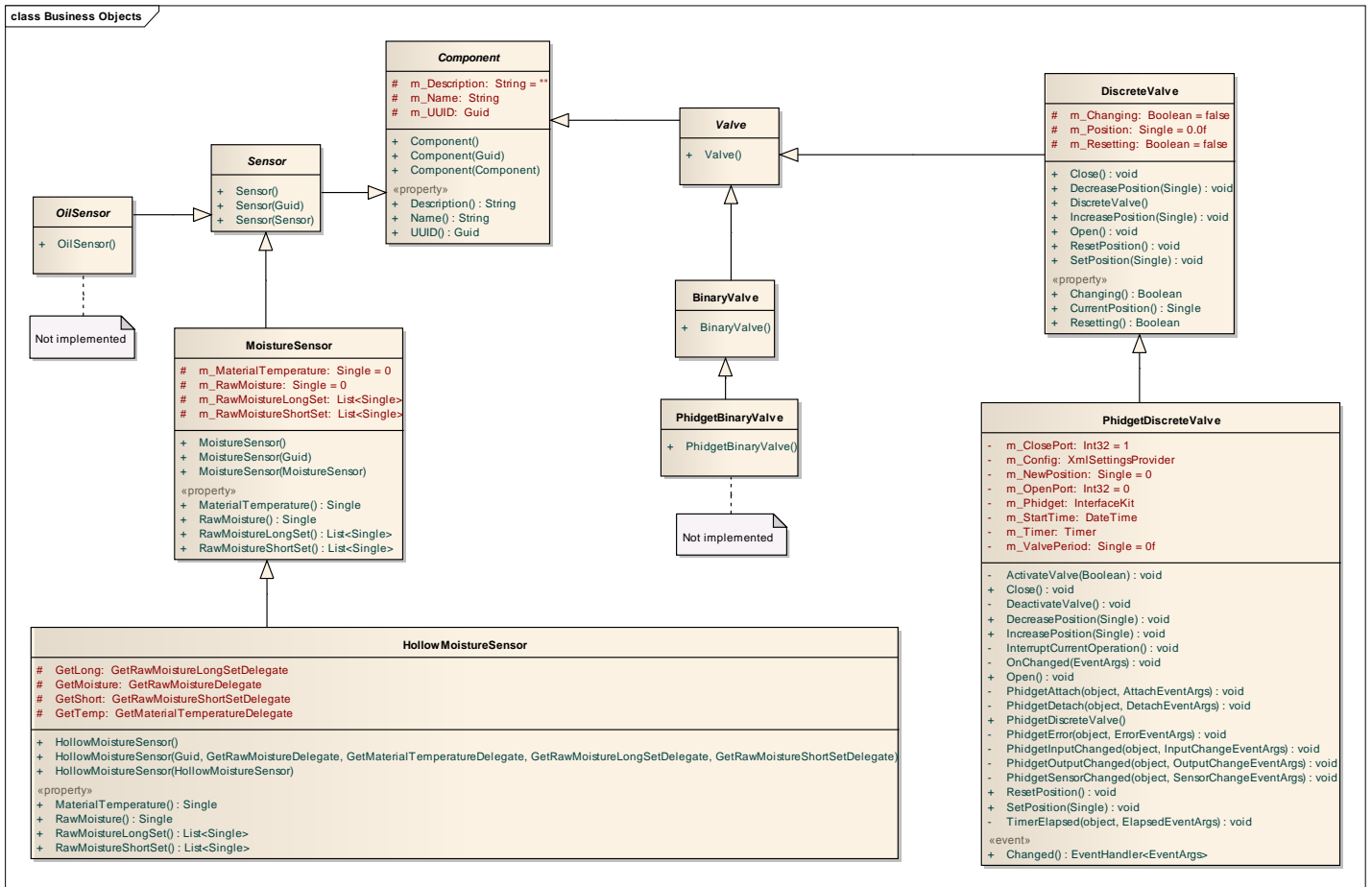


Figure 2 - Class Diagram of Control Module Business Objects

The business objects used by the Control module start out very generic to allow for future expansion and specialization. Currently there are only two hardware "components" implemented, a **MoistureSensor** and a **DiscreteValve**. These in turn are used to represent a Hydronix moisture sensor and a Belimo characterized control valve connected to a Phidget USB interface device. Note that the **HollowMoistureSensor** is a specialization of the **MoistureSensor** that utilizes callbacks (due to the lag in retrieving values from a physical sensor connected via a serial interface to a usb-to-serial

converter connected to a embedded device possibly at the other end of a slow network connection) and that the Get methods **MUST** be overridden by the concrete implementation.<sup>3</sup>

---

<sup>3</sup> The terms in non-standard capitalization and spelling are software objects referred to in the actual computer code.

# Persistence

class Persistence

```

class GlobalVars
+ BusinessLogicBasePath: String = "System/" {readOnly}
+ CsvAutoLogFilename: String = "autolog.csv" {readOnly}
+ CsvManualLogFilename: String = "manuallog.csv" {readOnly}
+ CsvSystemLogFilename: String = "systemlog.csv" {readOnly}
+ DataLoggerStringBuilderSize: Int32 = 150 {readOnly}
+ DefaultSettingsFileName: String = "settings.xml" {readOnly}
+ DesiredMoisturePercentage: Single = 10.0f {readOnly}
+ DiagnosticLoggerInterval: Int32 = 15000 {readOnly}
+ DiagnosticLoggerStartupMessage: String = "Logging initia... {readOnly}
+ DiagnosticLoggerSystemClogMessage: String = "logging stop w... {readOnly}
+ DiagnosticLoggerSystemStartMessage: String = "logging start" {readOnly}
+ DiagnosticLoggerSystemStopMessage: String = "logging stop" {readOnly}
+ HydronixCalibrationIntercept: Single = -7.68f {readOnly}
+ HydronixCalibrationSlope: Single = 0.965f {readOnly}
+ HydronixFastTrendValueCount: Int32 = 30 {readOnly}
+ HydronixSearchTimeout: Int32 = 100 {readOnly}
+ HydronixSensorTimeout: Int32 = 600 {readOnly}
+ HydronixServiceBasePath: String = "Services/Senso... {readOnly}
+ HydronixSlowTrendValueCount: Int32 = 70 {readOnly}
+ LongSetValueCount: Int32 = 100 {readOnly}
+ MassPerCycle: Single = 1216.8f {readOnly}
+ PhidgetsBasePath: String = "System/Hardwar... {readOnly}
+ PhidgetsDefaultSerial: String = "73656" {readOnly}
+ PhidgetTimeoutPeriod: Int32 = 500 {readOnly}
+ SensorUpdatePeriod: Int32 = 8 {readOnly}
+ ServiceQueryPeriod: Int32 = 750 {readOnly}
+ ShortSetValueCount: Int32 = 25 {readOnly}
+ ValveDefaultClosePort: Int32 = 1 {readOnly}
+ ValveDefaultCycleTime: Single = 90f {readOnly}
+ ValveDefaultOpenPort: Int32 = 0 {readOnly}
+ ValveUpdatePeriod: Int32 = 24 {readOnly}
+ WcfBasePath: String = "System/Communi... {readOnly}
+ WcfSensorServiceBasePipeAddress: String = "http://localho... {readOnly}
+ WcfSensorServiceBaseTcpAddress: String = "http://localho... {readOnly}
+ WcfSensorServiceMexPipeAddress: String = "http://localho... {readOnly}
+ WcfSensorServiceMexTcpAddress: String = "http://localho... {readOnly}
+ WcfSensorServicePipeHostAddress: String = "net.pipe://loc... {readOnly}
+ WcfSensorServiceTcpHostAddress: String = "net.tcp://loca... {readOnly}
+ WcfValveServiceBasePipeAddress: String = "http://localho... {readOnly}
+ WcfValveServiceBaseTcpAddress: String = "http://localho... {readOnly}
+ WcfValveServiceMexPipeAddress: String = "http://localho... {readOnly}
+ WcfValveServiceMexTcpAddress: String = "http://localho... {readOnly}
+ WcfValveServicePipeHostAddress: String = "net.pipe://loc... {readOnly}
+ WcfValveServiceTcpHostAddress: String = "net.tcp://loca... {readOnly}
    
```

```

«interface»
SettingsProvider
+ GetSetting(string, T) : T
+ PutSetting(string, T) : void

XmlSettingsProvider {leaf}
- m_AddAsNeeded: Boolean = true
- m_DocumentPath: string = Application.Sta...
- m_Instance: XmlSettingsProvider = new XmlSettings... {readOnly}
- m_XmlDocument: XmlDocument = new XmlDocument()

- AbsolutePath(string, string) : string
- ConvertFromStringToT(string, T*) : bool
- ConvertFromTToString(string*, T) : bool
- CreateMissingNode(string) : XmlNode
+ GetFilenameSetting(string, string) : string
+ GetSetting(string, T) : T
- GetSetting(string, string) : string
- InitializeDocument(String) : void
+ PutFilenameSetting(string, string) : void
+ PutSetting(string, T) : void
+ PutSetting(string, string) : void
- RelativePath(string, string) : string
- XmlSettingsProvider()
- ~XmlSettingsProvider()

«property»
+ AddAsNeeded() : Boolean
+ DocumentPath() : string
+ Instance() : XmlSettingsProvider
    
```

```

«interface»
DataLogger
+ Log(T) : void
+ Log(List<T>) : void
+ LogWithTimestamp(T) : void
+ LogWithTimestamp(List<T>) : void

«interface»
CsvDataLogger
- m_filename: String = ""
+ CsvDataLogger(string)
+ Log(T) : void
+ Log(List<T>) : void
+ LogWithTimestamp(T) : void
+ LogWithTimestamp(List<T>) : void
- WriteString(String) : void
    
```

This is NOT an interface, EA seems to be having problems.

Figure 3 - Control Module Persistence Tier. Diagram IS THIS FIGURE BACWARDS?

The persistence tier provides functionality in three main areas. First, it stores a variety of constant values in GlobalVars. Currently this class just exposes several hard-coded constant values, but could easily be made to pull the constant values from somewhere else. These centrally collected values serve as the default settings for the module, see the inline code documentation for each value for more information.

The second area of functionality is the SettingsProvider, implemented in the concrete XmlSettingsProvider. This class provides a simple get setting, save setting interface for persisting settings to an XML file. In the case that the settings file doesn't exist this class creates the file and populates it with default values, those values are retrieved from GlobalVars. Thus specific instance changes are made to the settings file, and tuning of the overall application default settings are made to GlobalVars.

The third area of functionality is data logging, generically described by DataLogger and concretely implemented by CsvDataLogger which logs values or lists of values to a CSV file, with or without timestamps. Data logging scenarios requiring more functionality than a CSV file would need to provide another concrete implementation of DataLogger, probably utilizing a database.

# Business Logic

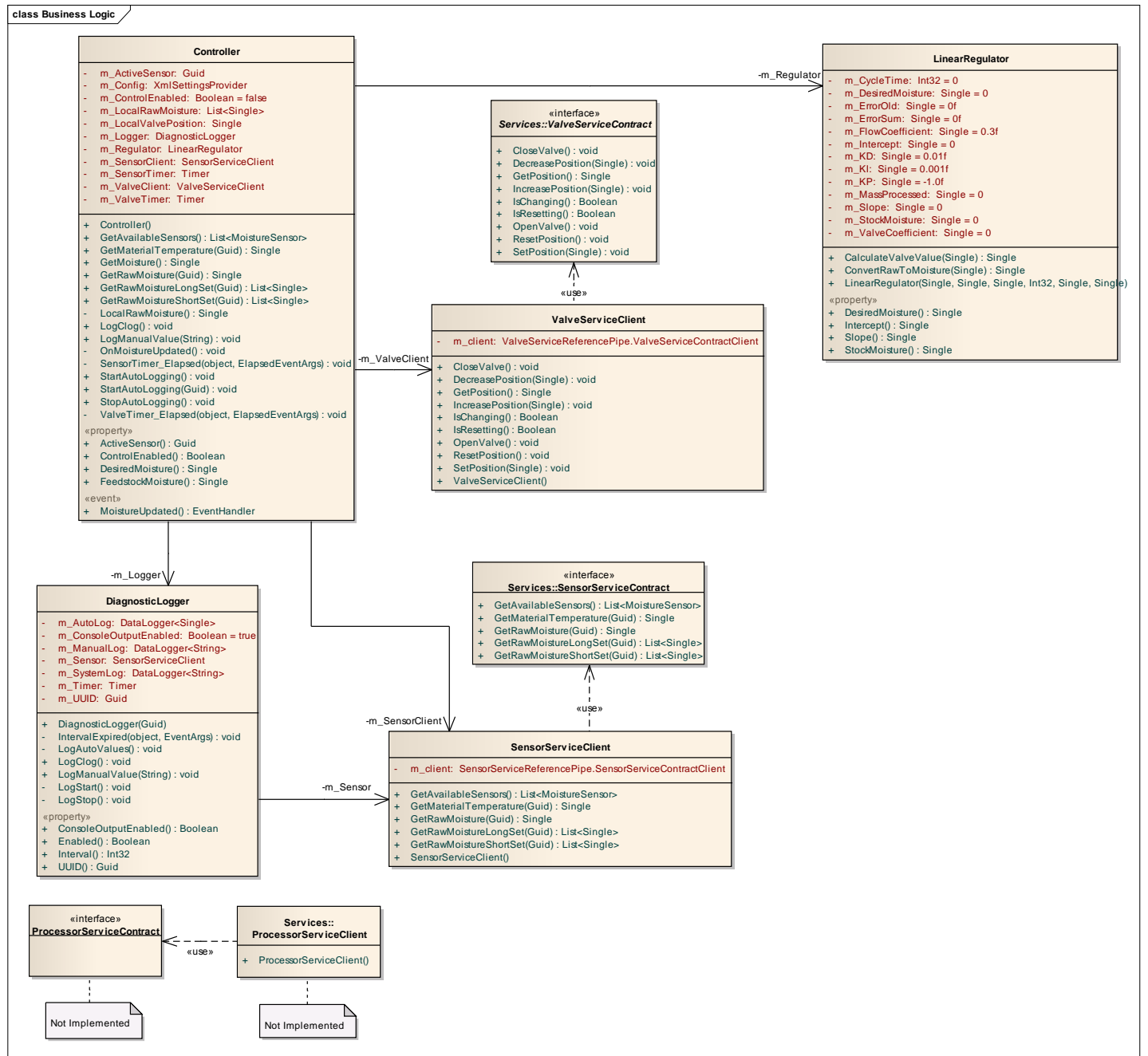


Figure 4 - Control Module Business Logic Class Diagram

The heart of the Control module's business logic is the Controller, it is the primary point of interaction between the gui and the rest of the system.

The LinearRegulator contains the control algorithm of the application, it takes raw sensor readings and calibrates them for the sensor and material being used, then takes the "real" moisture readings and calculates a value to be applied to

the water valve. This class contains several constant values and a control algorithm developed by Marc Timmerman, though unfortunately the current constants being used require further tuning.

In addition to the Controller and LinearRegulator are the classes needed to access the valve and sensor via a web service, ValveServiceClient and SensorServiceClient respectively.

Finally there is the DiagnosticLogger which logs values to the files "autolog.csv", "systemlog.csv", and "manuallog.csv". Currently it logs moisture sensor readings, system operation events such as starting, stopping, and clogging, and values manually entered such as those generated by a standalone moisture meter. "Autolog.csv" starts with a timestamp column, followed by multiple columns holding the RawMoistureShortSet (sensor specific) values, with a final column for material temperature. An example of a RawMoistureShortSet would be raw 25 moisture values taken over a short period of time. "Systemlog.csv" starts with a timestamp and follows with a message describing a system event, such as initialization, and the serial number of the related sensor. "Manuallog.csv" simply contains a timestamp and the value manually logged.

## Presentation

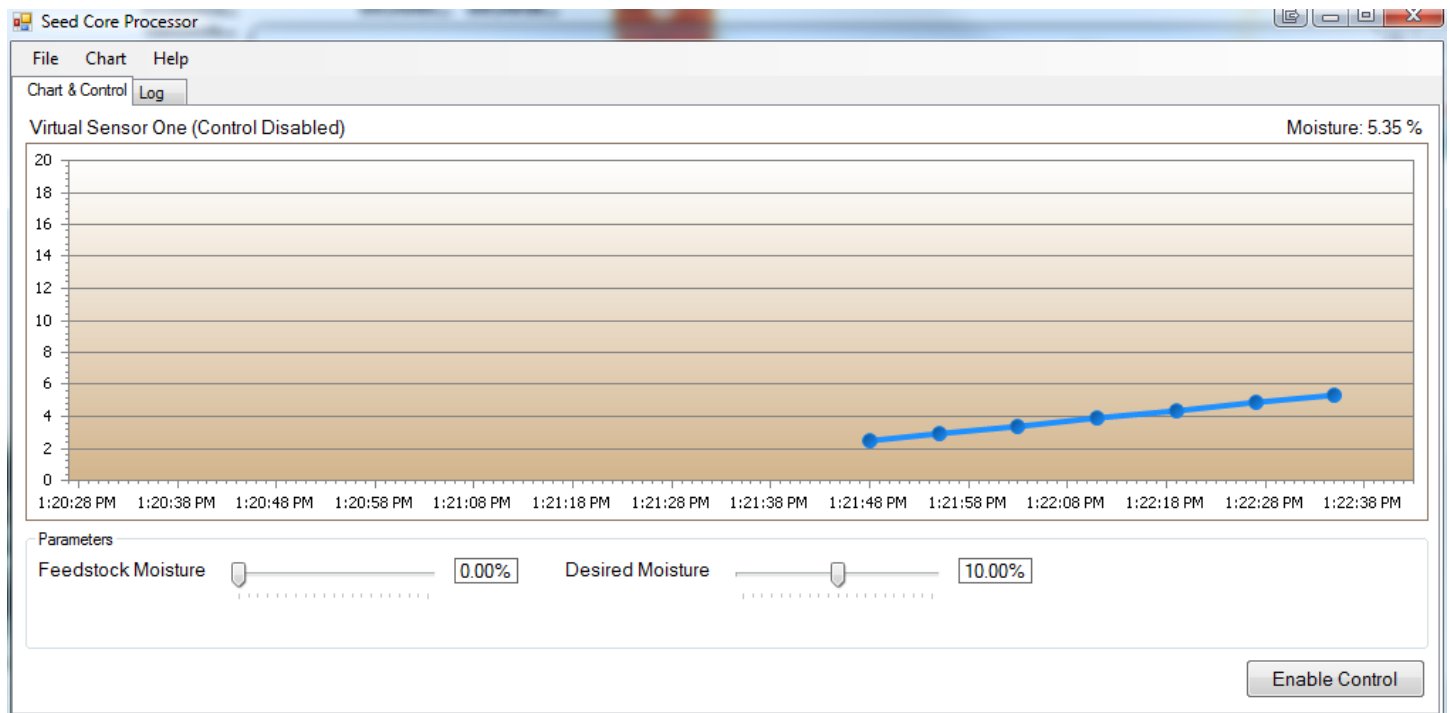


Figure 5 - Main UI: Chart and Moisture Control

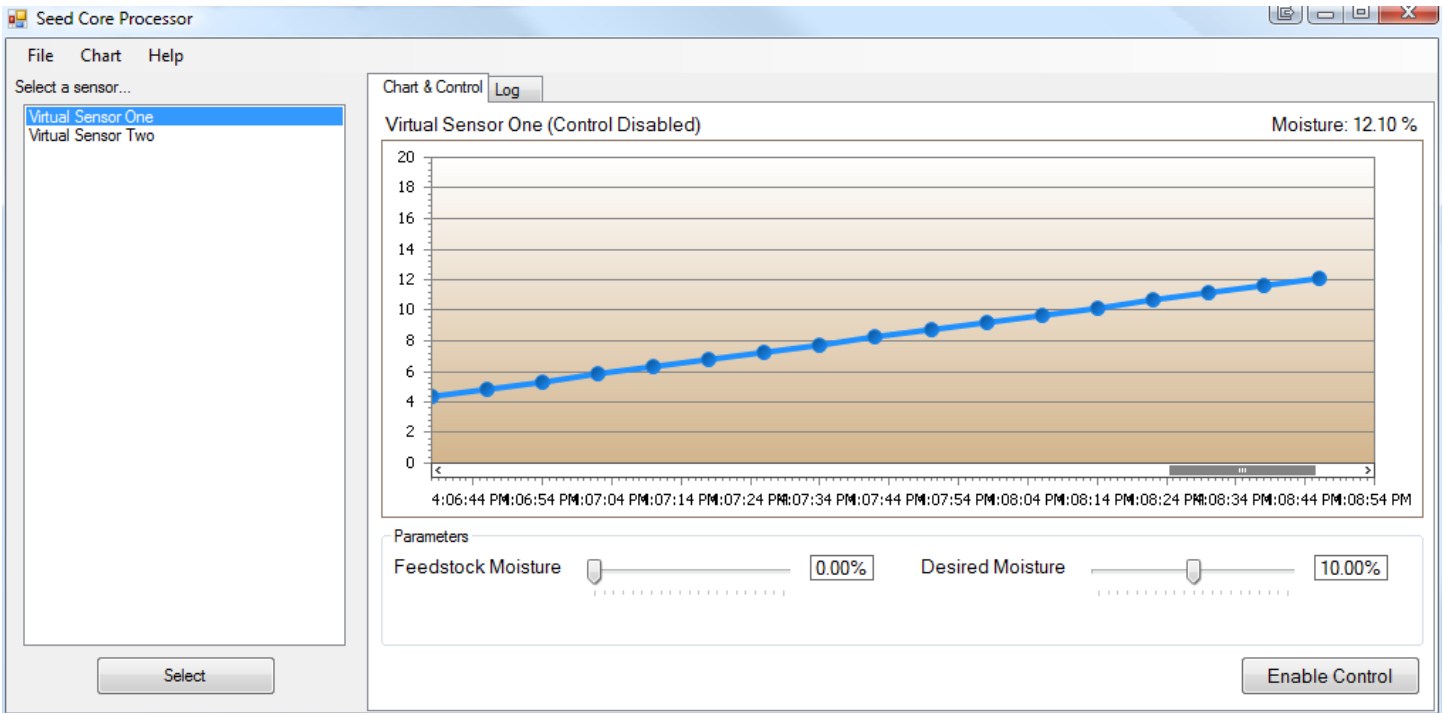


Figure 6 - Main UI: Select Active Sensor

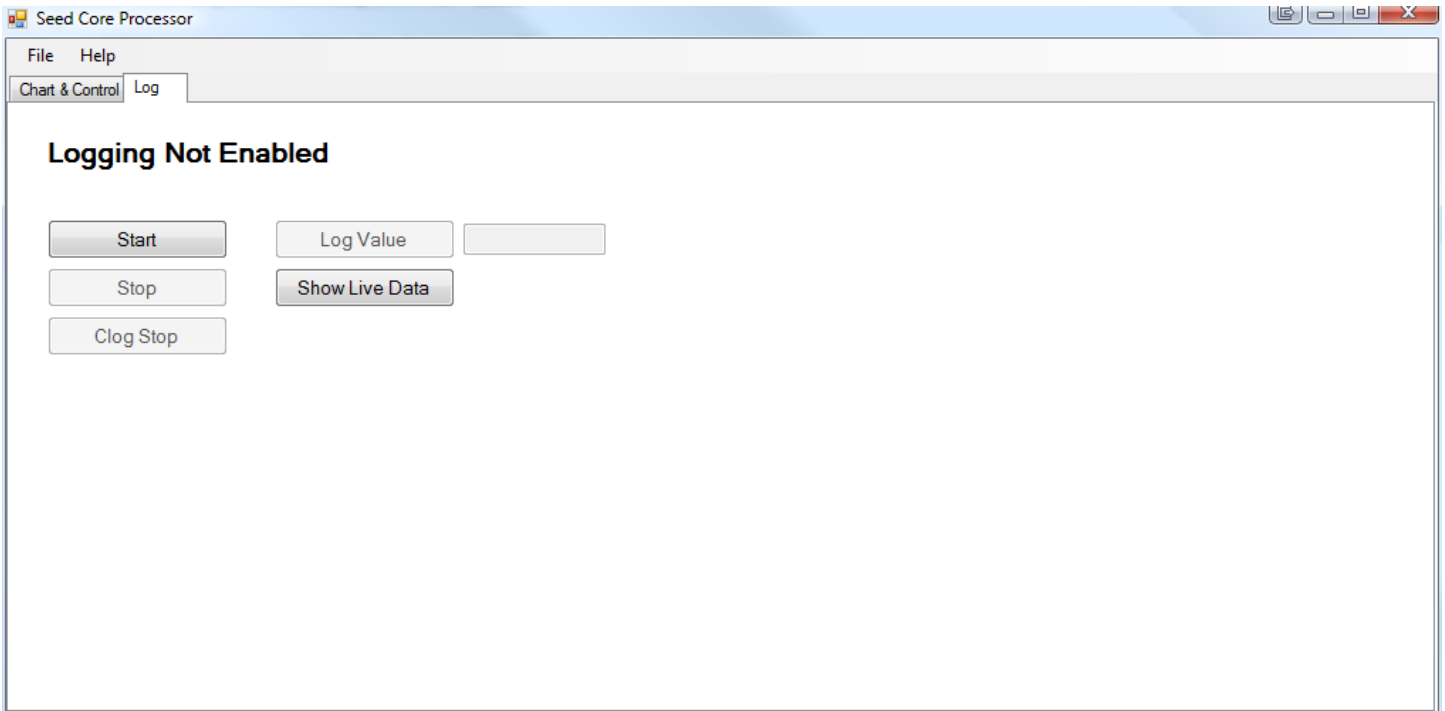


Figure 7 - Main UI: Logging and Live Data

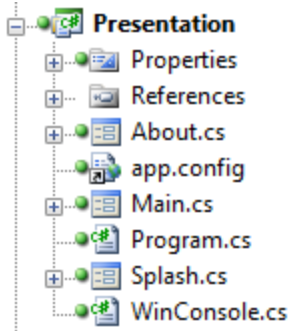


Figure 8 - Control Module Presentation Project Elements

Presentation utilizes conditional compilation to provide a working but "fake" UI instance for debugging when "Debug" is the current configuration in Visual Studio, and a real instance when "Release" is the current Visual Studio configuration. The Debug configuration uses a simple timer and incrementing variable as a "sensor" and enabling control has no effect, while the Release configuration attempts to access a real sensor via the web service endpoint defined in the settings and enabling control begins manipulation of the water valve similarly accessible via a web service.

At startup the application displays a splash screen. The splash screen is necessary because connecting or timing out while attempting to connect to the sensor and valve services takes some time and the UI is unresponsive during this period. To display moisture data in chart form a commercial UI control library was used, DXperience 8.2 from DevExpress. While a one year license was donated to the Seed Core Processor project that license has since expired, leaving the control operating in "demo" mode.

Currently there are two main areas of functionality in the UI, charting/control, and logging. Charting and control displays live moisture (calibrated) readings and allows enabling/disabling moisture control. Logging handles automatic logging and manual logging, and toggles the display of live data in number form via a separate window. A third area of the UI is needed but isn't currently developed, and that's a tab for manipulating the constants of the control algorithm which currently is in need of more work.

# Services

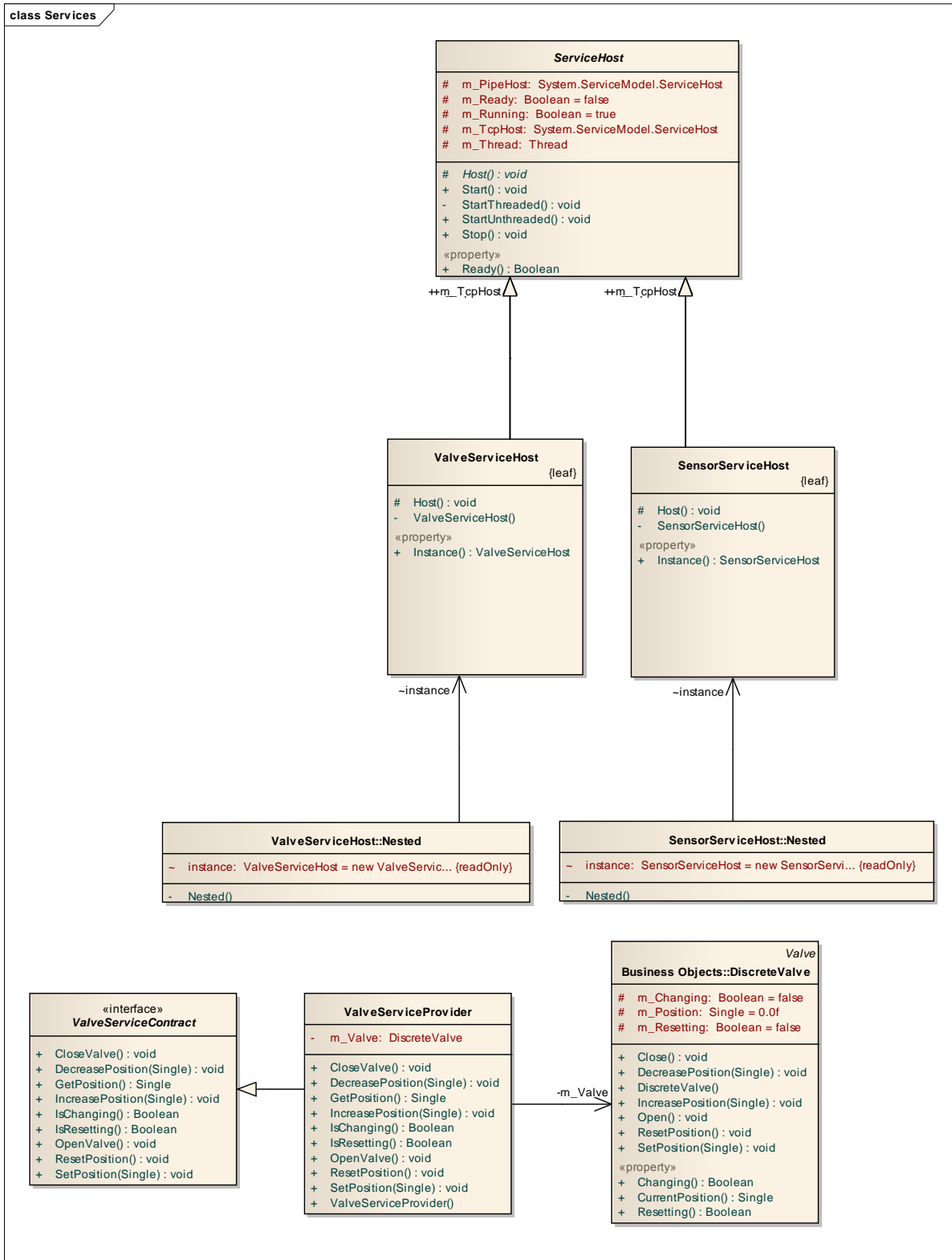


Figure 9 - Service Host and Valve Interaction



## Figure 10 - Moisture Service Interaction

The Interface module is really a separate n-tier application that wasn't split into separate projects and diagrams as it should have been, and has multiple areas of functionality.

The first area of functionality is controlling the Hydronix moisture sensor used for the project while leaving open the likelihood of different sensors being used in the future, the hub of this is the SensorController. The SensorController interacts with the SensorLink, which is a representation of the physical communication link with the sensor and is only responsible for the real-world interaction with the sensor. The SensorController is an abstract base class that is specialized by classes for controlling specific sensor models. The HydronixSensorController is a concrete implementation of the SensorController, and is specific to Hydronix sensors. Another concrete implementation of the SensorController is the VirtualSensorController which provides access to a "fake" sensor for debugging. The SpecificMoistureSensor interface is the key gateway between the system and a specific sensor, this interface is implemented by the HydronixDetailedSensor class which inherits from the Hydronix supplied Sensor class. The Hydronix Sensor class is not displayed in the diagrams above due to its size. Of note in this architecture are the different classes representing a link to a sensor and the sensor itself, rather than having the link be a property of the sensor. This implementation is directly due to the code provided by Hydronix that was developed in this manner, future sensors and implementations may require re-factoring.

Making use of the SensorController and the hardware it provides access to is the SensorServiceProvider, whose job is to provide concrete methods for a web service that gives access to the SensorController. The SensorServiceContract defines the contract of this web service, and is consumed by the SensorServiceClient Control business logic class. Similar is the ValveServiceProvider and ValveServiceContract, which are the interface for a web service and the concrete implementation of that interface.

The final area of functionality is the WCF service hosts for the valve and sensor services. The base class ServiceHost provides the threaded framework for hosting a service, while ValveServiceHost and SensorServiceHost are the concrete service host implementations. Hosting the services makes them available either locally or remotely depending upon the parameters specified, and exposes a simple SOAP API for manipulating the sensor and valve.

## Usage

Usage consists of two main parts, running the service and consuming and controlling the valve and sensor via web service. Visual Studio is required to compile the source code, and .NET 3.5 is required to run it.

Running the Service requires compiling in Visual Studio's Release mode, then browsing to the output directory of the Services project which will contain the files needed. Delete the settings.xml file to cause it to be recreated with default values, or edit the values as needed. The next step is to launch the Sensor\_Service executable which will host the service in a console window. Note that the service will not be able to start if the correct hardware isn't connected; a Hydronix moisture sensor connected via USB interface and a control valve connected via Phidget USB relay board are required. If this hardware isn't available the Control portion can be run in Debug mode, but the Service cannot. After launching the executable a console window should appear indicating the Service Host was started, closing this window closes the service host.

Once the Service has been launched the Controller can be started, compile in Release mode and launch the executable in the Presentation output directory. It is currently configured to be run on the same machine as the Service, modify the settings file to connect to a remote sensor service if desired. Upon launching the Controller it will connect to the Services for the valve and sensor, and begin displaying moisture readings. To control the moisture of the seed set the approximate moisture content of the current feedstock using the slider on the bottom, then set the desired final moisture content. Press enable control to begin operation. Logging can be enabled or disabled via the Log tab. If everything is working correctly the sensor will be queried for raw moisture readings, those values will be converted to calibrated values, the calibrated values will be used to determine the amount of water needed, and the water valve will be adjusted.

## Troubleshooting

Both Hydronix and Phidgets provide basic applications for testing the functionality of their hardware. If these fail both vendors supply further documentation and troubleshooting information. Further documentation about this system is available from the inline code documentation. As a final resort the original author, Drew Loika, can be contacted at [drew.loika@gmail.com](mailto:drew.loika@gmail.com).