# Jurica Bioreactor

## Design notes and schematics*

*Engineering reference only. Not intended to replace the user's manual

Mike Rightmire

June 24, 2011

## Contents

## General Overview

The Jurica bioreactor project was implemented to improve and maximize the growth rate, cell density, and general cellular activity of suspension HeLa cells at low (3 liter) volumes. A secondary motivation was to create a growth environment where sterility could be maintained outside of a class II cabinet or incubator even during sampling and media addition.

***The primary design requirement is to create an easily reproducible bioreactor that could be built for less than $5000 from existing off-the-shelf components so additional vessels could be assembled quickly as required.***

The Jurica Lab cultures HeLa cells for the harvest of active macromolecular spliceosome. We have conceptually designed the Jurica Bioreactor with three primary concerns towards maximization of this harvest. The first is the collection of a very active and healthy spliceosome. The spliceosome is most active when the cells are rapidly dividing, and our goal is to culture cells which are doubling less than every 24 hours.

The second concern is the volume of spliceosome harvested. Since more active cells create more of the macromolecule, a larger number of these highly active cells will increase the overall harvest. In a standard spinner flask, the cells are able to reach a maximum of $4x10^5$ cells per mL based on a number of constraints. Our reactor is intended to alleviate these constraints and thus increase the maximum cell density. Some of the more recognized constraints are:

- Surface contact.
  As cells have greater cell-to-cell contact rate, they become less active. There are a number of chemotactic factors (both positive and negative) which are affected by the direct adhesion of cells to one another. There is an ideal contact rate which is difficult to attain without a highly controlled and monitored environment. A balance must be preserved between “breaking up” the cells through agitation - and the damage to cells from shearing.

- Chemical signaling down regulation.
  A more significant factor affecting the density of HeLa are the chemotactic agents which trigger down regulation of mitosis. It is understood that cells almost immediately begin to secret down regulators in response to both mitosis and cell adhesion. Even when adhesion is controlled, a buildup this signaling chemistry in the closed spinner environment can significantly down regulate cell division. Adding medium traditionally reduces this effect, but its effectiveness is limited as volumes and cell density increases.

- $CO_2$ Partial pressure
  Some research implies the single greatest impact on cell density and division rates may be the concentration of dissolved $CO_2$. During initial incubation cells need to additional environmental $CO_2$, traditionally through an incubator’s 5% atmospheric partial pressure. However as the cells begin producing waste $CO_2$, not only does the addition of $CO_2$ become unnecessary but the waste $CO_2$ can reach toxic levels. In our previous cultures we have seen a significant drop off in cell division rates once the culture reaches 1.5 liters at $4x10^5$ cells/mL. This corresponds with the research’s estimates of when waste $CO_2$ exceeds a spinner flask’s ability to aerate it.

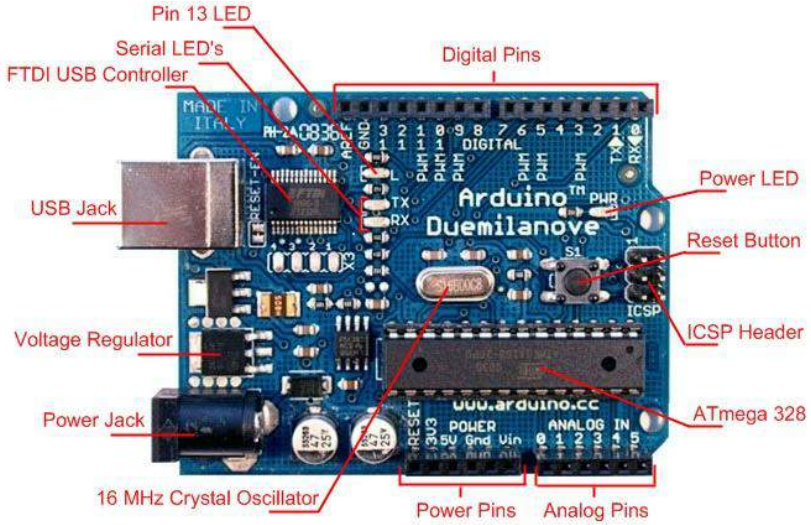
- $O_2$ partial pressure
  Similar to $CO_2$, dissolved $O_2$ influences cell activity and division. Again, the inherent design of a closed and vented spinner flask is not adequate for the dissolved $O_2$ needs of larger volumes of spinner culture. Although, less significant than dissolved $CO_2$ concentrations, controlled addition of dissolved $O_2$ often allows for peak activity to be maintained throughout density and volume curves.

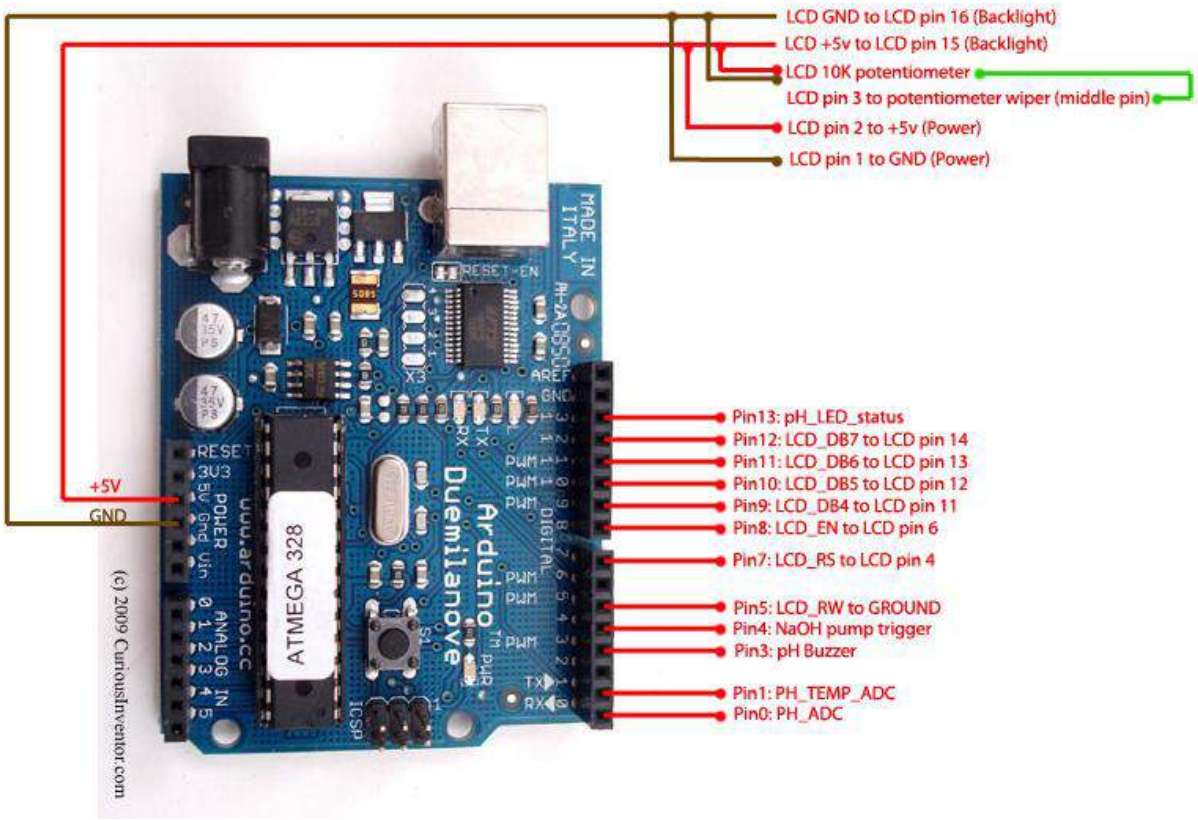
- Finally, the issue of contamination has been of significant concern. The need to open the flask almost daily to perform cell counts and add medium has led to significant losses due from contamination, even within class II environments. The Jurica bioreactor design is intended to alleviate this by allowing sampling and medium addition via sterile, off-the-shelf medical valves. Testing to date has allowed both the introduction and removal of medium in even the most hostile of environments without contamination issues.

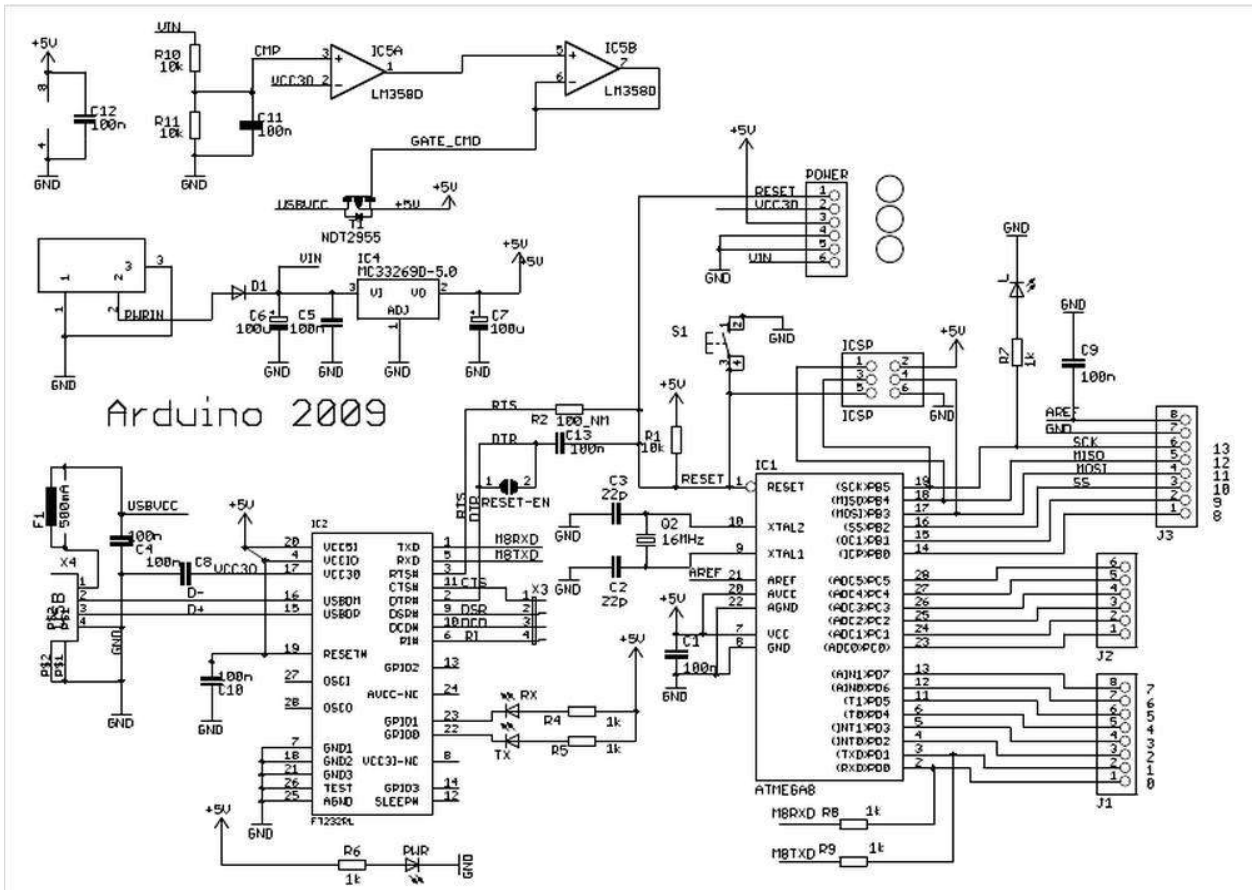
# Programmable logic controller (PLC)

## Overview

The Arduino Duemilanove was chosen as the highest performing board in its price range. With analog and ADC pinouts sensitive to the mV/mA ranges, it can communicate directly with pH, $O_2$ and temperature sensors with minimal noise and amplification. It is also capable of running full Java and Shell scripts. The USB port and Buildroot linux support wireless, giving the option to interface with the Bioreactor wirelessly. Additional memory was added using external USB flash RAM.




Arduino 2009
IC1
ATMEGA8
IC2
FT232RL
IC4
MC33269D-5.0
IC5A
IC5B
LM358D
NDT2955
POWER
ICSP
J1
J2
J3
RESET-EN
USBVCC
VCC30
VIN
GATE_CMD
M8RXD
M8TXD
RX
TX
PWR
AREF
SCK
MISO
MOSI
SS

# PLC Code

## Overview

The Arduino uses a variation of the C programming language. These are a series of customized "sketches" derived from an opensource model written originally by Carlos Neves (et al). pH system input parameters and the control logic for pumps and displays were added. Mr. Neves is appropriately credited in the comments of the script and should not be removed.

## pH_subsystem_Bioreactor1_v1_0 - 5.25.2010.pde

```
/*
 pHduino.pde - pHduino firmware for Arduino.
 Copyright (c) 2009 Carlos A. Neves
 http://code.google.com/p/phduino

 This program is free software: you can redistribute it and/or
modify
 it under the terms of the GNU General Public License as
published by
 the Free Software Foundation, either version 3 of the License,
or
 (at your option) any later version.

 This program is distributed in the hope that it will be useful,
 but WITHOUT ANY WARRANTY; without even the implied warranty of
 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 GNU General Public License for more details.

 You should have received a copy of the GNU General Public
License
 along with this program.  If not, see
<http://www.gnu.org/licenses/>.

 Version 0.4_20090920
 Author: Carlos A. Neves <caneves at google>

 Version   0.4_20120525
           5/25/2012
 Author:   Mike Rightmire
           Jurica Labs
           Center for the Study of RNA
           University of California, Santa Cruz
           mrightmi@ucsc.edu
 */

#include "WProgram.h"
#include <LiquidCrystal.h>
#include "pHduino.h"
#include "float2string.h"
#include "commandline.h"
#include "string.h"
#include "stdlib.h"
#include <avr/eeprom.h>
//#include "freemem.h"

// Creates the LCD instance
LiquidCrystal lcd(PH_LCD_RS_PIN, PH_LCD_RW_PIN, PH_LCD_E_PIN,
PH_LCD_D4_PIN, PH_LCD_D5_PIN, PH_LCD_D6_PIN, PH_LCD_D7_PIN);

// ordinary variables
char myStr[16];
int val_E;
int val_R;
int val_T;
float E;
float R;
float T;
float pH;
int i;
float f, f2;

unsigned long int my_timer0;
unsigned long int my_timer;

boolean pH_alarm_status = true;

// struct of parameters
struct myRec_T{
  boolean start_flag;
  unsigned int time_between_acquisitions_ms;
  boolean temperature_sensor_flag;
  float temperature_constant_celcius;
  boolean pH_alarm_flag;
  float pH_inf;
  float pH_sup;
  float pH_add_NaOH;
  float pH_add_HCl;
}
myRec;

void parameters_reset(void){

  myRec.start_flag = true;
  myRec.time_between_acquisitions_ms =
PH_INITIAL_TIME_BETWEEN_ACQUISITIONS_MS;
  myRec.temperature_sensor_flag = false;
  myRec.temperature_constant_celcius =
PH_TEMPERATURE_VALUE_INITIAL_CELCIUS;
  myRec.pH_alarm_flag = true;
  // The following sets pH ALARM boundaries
  myRec.pH_inf = 7.0;
  myRec.pH_sup = 8.0;
  // The following sets target pH boundaries
  myRec.pH_add_NaOH = 7.3;
  myRec.pH_add_HCl = 7.9;

  eeprom_write_block(&myRec, (void *)0, sizeof(myRec));
}

//// Start setup
void setup(void){
  parameters_reset();
  // Init serial port
  Serial.begin(9600);

  // setup LED, button, and buzzer
  pinMode(PH_LED_PIN, OUTPUT);
  pinMode(PH_BUTTON_CTRL_PIN, INPUT);
  pinMode(PH_BUZZER_PIN, OUTPUT);

  // initialize LED and buzzer
  digitalWrite(PH_LED_PIN, LOW);
  digitalWrite(PH_BUZZER_PIN, LOW);

  // begin LCD
  lcd.begin(16, 2);
  lcd.clear();
  lcd.noCursor();
  lcd.home();
  lcd.display();

  // Buzzer test
  digitalWrite(PH_BUZZER_PIN, HIGH);
  delay(500);
  digitalWrite(PH_BUZZER_PIN, LOW);

  //LCD write tests
  digitalWrite(PH_LED_PIN, HIGH);
  lcd.setCursor(0,0);
  lcd.print("Jurica");
  lcd.setCursor(0,1);
  lcd.print("     Labs");
  delay(5000);
  lcd.setCursor(0,0);
  lcd.print("Bioreactor1");
  lcd.setCursor(0,1);
  lcd.print("pH subsys v1.0");
  delay(5000);
  digitalWrite(PH_LED_PIN, LOW);
  lcd.clear();

  // Serial write tests
  Serial.println("Jurica Labs");
  Serial.println("Bioreactor1");
  Serial.println("pH subsys v1.0");


  // read parameters from EEPROM
  eeprom_read_block(&myRec, (void *)0, sizeof(myRec));
```

```
  // Free memory check
  //Serial.println(availableMemory());

  }// End Setup

//// Start Loop
void loop(void){

  my_timer = millis();

  if ((my_timer-my_timer0) > myRec.time_between_acquisitions_ms){
    // reset the timer
    my_timer0 = my_timer;

    if (myRec.start_flag == true){
      process_data();
    }
  }
  // Check for input from serial line
  process_cmd();
}// End loop

///// METHODS
void process_data(void){
  char* tempType;
  digitalWrite(PH_LED_PIN, HIGH);
  if (myRec.temperature_sensor_flag == true){
    val_T = readADC(PH_TEMP_ADC_PIN,
PH_TIME_ADC_READINGS_MICROSECONS);
    T = 100 * val_T * 5.0 / 1024;
    tempType="Actual: ";
  } else {
    T = myRec.temperature_constant_celcius; // celcius
temperature to operates without a temperature sensor.
    tempType="Assume: ";
  }//endif

  // Send temp data to readouts
  floatToString(myStr, T, 1);

  lcd.setCursor(0,0);
  lcd.print(tempType);
  lcd.print(myStr);
  lcd.print((char)223); // degree symbol
  lcd.print("C  ");

  Serial.print("info:");
  Serial.print(tempType);
  Serial.print(myStr);
  Serial.print(" Degrees");
  Serial.println(" C ");

  //val_R = readADC(PH_REF_ADC_PIN,
PH_TIME_ADC_READINGS_MICROSECONS);
  //R = PH_GAIN_STAGE2_REF * (val_R * 5.0 / 1024);

  // print ADC value and voltage of electrochemical potential
  // val_E = readADC(PH_PH_ADC_PIN,
PH_TIME_ADC_READINGS_MICROSECONS);
  // E = -1 * ((val_E * 5.0 / 1024) - R) / PH_GAIN_STAGE1_PH;
  val_E = analogRead(1);
  E = ((val_E * 5.0 / 1024));
  Serial.print("info:val_E(pH sensor pin) = ");
  Serial.println(val_E);
  Serial.print("info: E = "); // mili volts
  Serial.print(E);
  Serial.println("V ");

  pH = 7 - ((2.5 - (val_E / 200.00)) / (0.257179 + 0.000941468 *
T)); // Calc takes into account temp "T"
  //pH = 0.0178 * val_E - 1.889; // Most basic calc, no adjust
for temp
  //pH = 0.0178 * (E * 200) - 1.889; // Basic calc, using voltage
not ADC input
  floatToString(myStr, pH, 2);

  lcd.setCursor(0, 1);
  lcd.print("pH: ");
  lcd.print(myStr);
  lcd.print("  ");

  Serial.print("info:pH = ");
  Serial.println(myStr);

  // print alarm signal
  if (myRec.pH_alarm_flag == true){
    if ((pH < myRec.pH_inf) || (pH > myRec.pH_sup)){
      Serial.println("alert:pH ALARM ACTIVE");
      pH_alarm_status = true;
      lcd.setCursor(11, 1);
      lcd.print("!!!!!");
      digitalWrite(PH_BUZZER_PIN, LOW);
      delay(200);
      lcd.setCursor(11, 1);
      lcd.print("ALARM");
      digitalWrite(PH_BUZZER_PIN, HIGH);
     }else{
      pH_alarm_status = false;
      lcd.setCursor(11, 1);
      lcd.print("    ");
      lcd.print(" ");
      digitalWrite(PH_BUZZER_PIN, LOW);
    }
  }else{
   pH_alarm_status = false;
   digitalWrite(PH_BUZZER_PIN, LOW);
  }
}

void process_cmd(void){
 // placeholder for serial line input
}
```

## pHduino.h

```
/*
  pHduino.h - pHduino library for Arduino & Wiring
  Copyright (c) 2009 Carlos A. Neves
  http://code.google.com/p/phduino

  This program is free software: you can redistribute it and/or
modify
  it under the terms of the GNU General Public License as
published by
  the Free Software Foundation, either version 3 of the License,
or
  (at your option) any later version.

  This program is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public
License
  along with this program.  If not, see
<http://www.gnu.org/licenses/>.

  Version 0.4_20090920
  Author: Carlos A. Neves <caneves at google>

  Version  0.4_20120525
           5/25/2012
  Author:  Mike Rightmire
           Jurica Labs
           Center for the Study of RNA
           University of California, Santa Cruz
           mrightmi@ucsc.edu

*/

/* Bioreactor1 pH Subsystem based on pHduino by Carlos A. Neves
   v. 1.0
   Author: Mike P. Rightmire
   University of California, Santa Cruz
   June 2010
   MRightmi@UCSC.Edu
   Jurica@Biology.UCSC.Edu
*/


// firmware version
#define PH_PHDUINO_VERSION  "0.4_20090920"
#define PH_PHDUINO_CODENAME  "acetic acid"
#define PH_BIOREACTOR1_VERSION  "1.0.20100610"
#define PH_BIOREACTOR1_CODENAME  "BIOREACTOR1"

// button pin
//#define PH_BUTTON_CTRL_PIN  2

// buzzer pin
#define PH_BUZZER_PIN  3

// LCD pins
#define PH_LCD_RS_PIN  7
//#define PH_LCD_RW_PIN  7 unused
#define PH_LCD_E_PIN   8
```

```
#define PH_LCD_D4_PIN  9
#define PH_LCD_D5_PIN  10
#define PH_LCD_D6_PIN  11
#define PH_LCD_D7_PIN  12

// Status led pin
#define PH_LED_PIN  13

// Sensor pins
// (Previously ADC pins)
#define PH_PH_ADC_PIN    0
//#define PH_REF_ADC_PIN   1
#define PH_TEMP_ADC_PIN  1

// Add NaOH trigger pin
#define NAOH_TRIGGER_PIN 4

// Sketch parameters
#define PH_PH_LOWER_LIMIT  0.0
#define PH_PH_UPPER_LIMIT  14.0

#define PH_PH_VOLTAGE_ADC_LOWER_LIMIT  0.0
#define PH_PH_VOLTAGE_ADC_UPPER_LIMIT  5.0

/* How many CONSECUTIVE pH senses ABOVE
   setpoint will trigger a single addition
   of NaOH. Each sense happens at APPROXIMATELY one second */
#define PH_ADD_NAOH_NUM_READINGS 15

// Time between NaOH additions REGARDLESS of pH sensings
#define PH_TIME_BETWEEN_ADD_NAOH_MILLISECONDS 00000 //60 seconds
#define NAOH_LENGTH_TRIGGER_MILLIS 1000

// constants
// currently unused #define PH_R        8.31451 // general gases
constant, J * K^-1 * mol^-1
// currently unused #define PH_F        96485   // Faraday
constant, C * mol^-1

// Gain of the non-inverter amplifier (stage1 of the schematic
circuit),
// considering a nernstiane glass electrode (delta_E=-59.2mV/pH)
and
// no influence from offset voltage divider.
// Gains currently usued thanks to Phidgets 1130 pH interface
// currently unused #define PH_GAIN_STAGE1_PH       4.8262 //
gain = ((4000mV)/(14*59.2mV)=828.8mV)

// Stage2 gain applyed to VREF.
// currently unused #define PH_GAIN_STAGE2_REF      2.0

// One 60Hz cycle cycle has 16.66ms or 16666us (limit up to
1,000,000us)
// One 50Hz cycle cycle has 20.00ms or 20000us (limit up to
1,000,000us)
// uncomment the right line.
#define PH_TIME_ADC_READINGS_MICROSECONS     16666 // 60Hz
//#define PH_TIME_ADC_READINGS_MICROSECONS     20000 // 50Hz

// Initial temperature for non temperature sensor mode
#define PH_TEMPERATURE_VALUE_INITIAL_CELCIUS    37.0

// Initial interval between acquisition
#define PH_INITIAL_TIME_BETWEEN_ACQUISITIONS_MS    1000

// Initial pH range to the pH alarm
#define PH_ALARM_INF      7.0
#define PH_ALARM_SUP      8.0

// functions
// readADC function unused.
unsigned int readADC(unsigned char channel, unsigned int
reading_time) {

  double d;
  int i;
  long t0_us;

  d = 0.0;
  i = 0;
  t0_us = micros();
  while((micros()-t0_us)<reading_time){
    i++;
    d += analogRead(channel);
  }
  d /= i;

  return (unsigned int)(d);
}
```

## freemem.h

```
// this function will return the number of bytes currently free in RAM
// written by David A. Mellis
// based on code by Rob Faludi http://www.faludi.com

int availableMemory() {
  int size = 1024; // Use 2048 with ATmega328
  byte *buf;

  while ((buf = (byte *) malloc(--size)) == NULL)
    ;

  free(buf);

  return size;
}
```

## float2string.h

```
// floatToString.h
//
// Tim Hirzel
// tim@growdown.com
// March 2008
// float to string
//
// If you don't save this as a .h, you will want to remove the
default arguments
//     uncomment this first line, and swap it for the next.  I
don't think keyword arguments compile in .pde files

//char * floatToString(char * outstr, float value, int places,
int minwidth=, bool rightjustify) {
char * floatToString(char * outstr, float value, int places, int
minwidth=0, bool rightjustify=false) {
    // this is used to write a float value to string, outstr.
oustr is also the return value.
    int digit;
    float tens = 0.1;
    int tenscount = 0;
    int i;
    float tempfloat = value;
    int c = 0;
    int charcount = 1;
    int extra = 0;
    // make sure we round properly. this could use pow from
<math.h>, but doesn't seem worth the import
    // if this rounding step isn't here, the value  54.321 prints
as 54.3209

    // calculate rounding term d:   0.5/pow(10,places)
    float d = 0.5;
    if (value < 0)
        d *= -1.0;
    // divide by ten for each decimal place
    for (i = 0; i < places; i++)
        d/= 10.0;
    // this small addition, combined with truncation will round
our values properly
    tempfloat +=  d;
```

```
    // first get value tens to be the large power of ten less
than value
    if (value < 0)
        tempfloat *= -1.0;
    while ((tens * 10.0) <= tempfloat) {
        tens *= 10.0;
        tenscount += 1;
    }

    if (tenscount > 0)
        charcount += tenscount;
    else
        charcount += 1;

    if (value < 0)
        charcount += 1;
    charcount += 1 + places;

    minwidth += 1; // both count the null final character
    if (minwidth > charcount){
        extra = minwidth - charcount;
        charcount = minwidth;
    }

    if (extra > 0 and rightjustify) {
        for (int i = 0; i< extra; i++) {
            outstr[c++] = ' ';
        }
    }

    // write out the negative if needed
    if (value < 0)
        outstr[c++] = '-';

    if (tenscount == 0)
        outstr[c++] = '0';

    for (i=0; i< tenscount; i++) {
        digit = (int) (tempfloat/tens);
        itoa(digit, &outstr[c++], 10);
        tempfloat = tempfloat - ((float)digit * tens);
        tens /= 10.0;
    }

    // if no places after decimal, stop now and return

    // otherwise, write the point and continue on
    if (places > 0)
    outstr[c++] = '.';


    // now write out each decimal place by shifting digits one by
one into the ones place and writing the truncated value
    for (i = 0; i < places; i++) {
        tempfloat *= 10.0;
        digit = (int) tempfloat;
        itoa(digit, &outstr[c++], 10);
        // once written, subtract off that digit
        tempfloat = tempfloat - (float) digit;
    }
    if (extra > 0 and not rightjustify) {
        for (int i = 0; i< extra; i++) {
            outstr[c++] = ' ';
        }
    }


    outstr[c++] = '\0';
    return outstr;
}


/*  EXAMPLE APPLICATION
 #include "floatToString.h"  //set to whatever is the location of
floatToStrig

 void setup() {

 Serial.begin(9600);

 char buffer[25]; // just give it plenty to write out any values
you want to test
 // =======================================
 // now run a series on the floatToString function

 // looking at the precision of the float

 Serial.println("floatToString(buffer, 1000000.321 , 5);");
 Serial.println(floatToString(buffer, 1000000.321, 5));
 Serial.println();
 Serial.println("floatToString(buffer, 100000.321 , 5);");
 Serial.println(floatToString(buffer, 100000.321, 5));
 Serial.println();
 Serial.println("floatToString(buffer, 10000.321 , 5);");
 Serial.println(floatToString(buffer, 10000.321, 5));
 Serial.println();
 Serial.println("floatToString(buffer, 1000.321 , 5);");
 Serial.println(floatToString(buffer, 1000.321, 5));
 Serial.println();
 Serial.println("floatToString(buffer, 100.321 , 5);");
 Serial.println(floatToString(buffer, 100.321, 5));
 Serial.println();

 // =======================================
 // looking at effect of changing precision
 Serial.println("floatToString(buffer, 100000.321 , 6);");
 Serial.println(floatToString(buffer, 100000.321, 6));
 Serial.println();
 Serial.println("floatToString(buffer, 100000.321 , 7);");
 Serial.println(floatToString(buffer, 100000.321, 7));
 Serial.println();
 Serial.println("floatToString(buffer, 100000.321 , 8);");
 Serial.println(floatToString(buffer, 100000.321, 8));
 Serial.println();
 Serial.println("floatToString(buffer, 100000.321 , 9);");
 Serial.println(floatToString(buffer, 100000.321, 9));
 Serial.println();


 // =======================================

 // check negatives and rounding, and some edge cases
 Serial.println("floatToString(buffer, -5004.321 , 5);");
 Serial.println(floatToString(buffer, -5004.321 , 5));
 Serial.println();

 Serial.println("floatToString(buffer, 99.999 , 3); ");
 Serial.println(floatToString(buffer, 99.999 , 3));
 Serial.println();

 Serial.println("floatToString(buffer, 100, 1);");
 Serial.println(floatToString(buffer, 100, 1));
 Serial.println();

 Serial.println("floatToString(buffer, -100.999 , 1);");
 Serial.println(floatToString(buffer, -100.999 , 1));
 Serial.println();

 Serial.println("floatToString(buffer, -54.321 , 0);");
 Serial.println(floatToString(buffer, -54.321 , 0));
 Serial.println();

 Serial.println("floatToString(buffer, 0.321 , 5);");
 Serial.println(floatToString(buffer, 0.321 , 5));
 Serial.println();

 Serial.println("floatToString(buffer, -1.0001 , 5);");
 Serial.println(floatToString(buffer, -1.0001 , 5));
 Serial.println();

 Serial.println("floatToString(buffer, -0.000001 , 5);");
 Serial.println(floatToString(buffer, -0.000001 , 5));
 Serial.println();

 Serial.println("floatToString(buffer, 0.000001 , 5);");
 Serial.println(floatToString(buffer, 0.000001 , 5));
 Serial.println();

 Serial.println("floatToString(buffer, -0.00001 , 5);");
 Serial.println(floatToString(buffer, -0.00001 , 5));
 Serial.println();

 Serial.println("floatToString(buffer, 0.000099 , 5);");
 Serial.println(floatToString(buffer, 0.000099 , 5));
 Serial.println();

 // alternative technique that depends on <stdio.h>
 //Serial.println("Serial.println(dtostrf(-1.0001, 2, 5, s));");
 //char s[32];
 // Serial.println(dtostrf(-1.0001, 2, 5, s)));

 delay(1000);
 Serial.println();

 }

 void loop() {
 }
*/
```

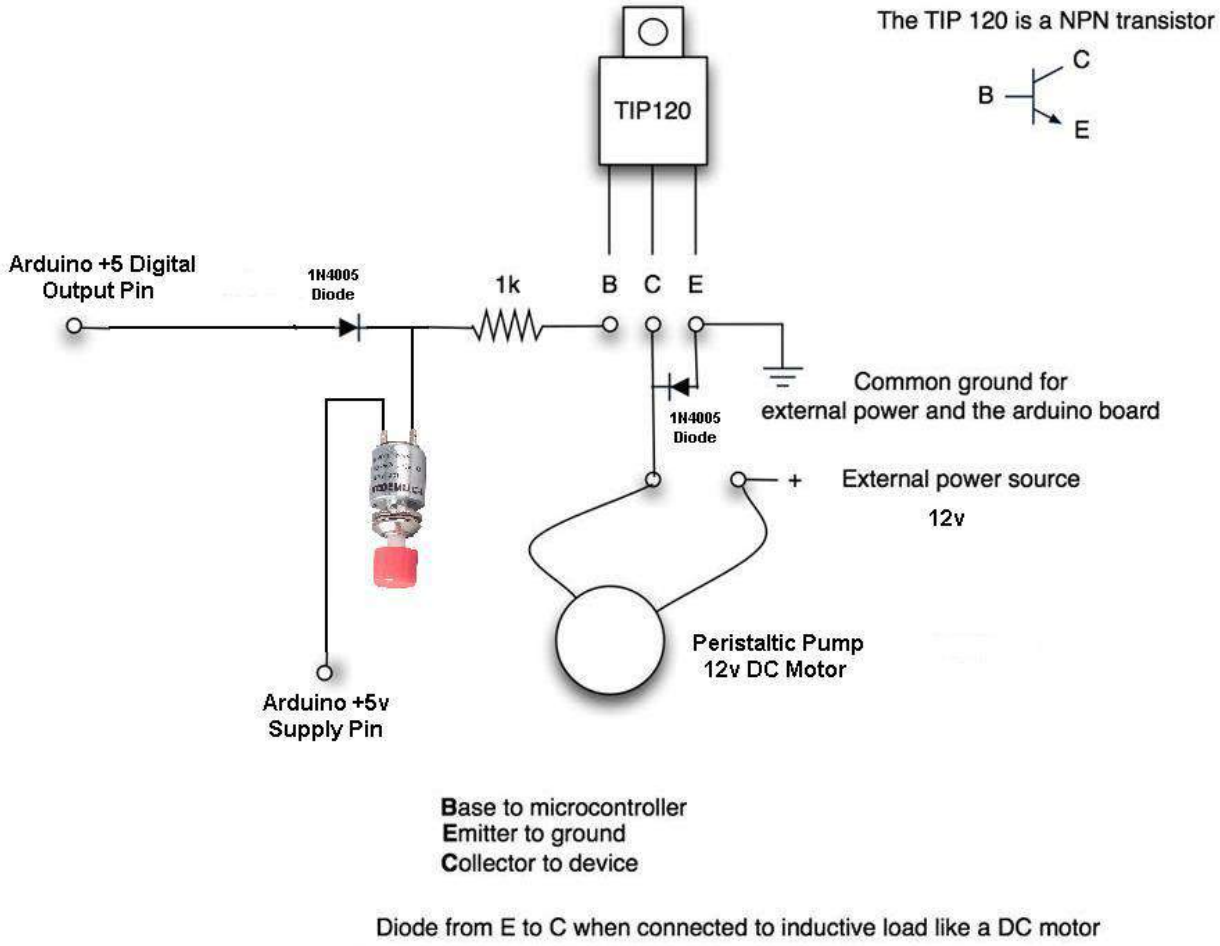
## commandline.h

```
// commandline.h
//
// http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1250265171/1#1
//
// With some modifications.

#define CL_MAX_CMD_CHAR_NUM      10  // max length of the command
string
#define CL_MAX_PARAMS_NUM        3 // max number of parameters
#define CL_MAX_PARAM_LEN          20 // max length of the
parameters
#define CL_MAX_PARAM_CHAR_NUM    63 // max length of the
parameters string
#define CL_MAX_CMD_LINE          80 // max length of the string
(cmd + spacers + parameters)

char cmd_line[CL_MAX_CMD_LINE]; //entire input command line
char cmd_str[CL_MAX_CMD_CHAR_NUM]; //first word of command line
char cmd_parm_str[CL_MAX_PARAM_CHAR_NUM]; // rest of the line
parsed out
int cmd_n_parms;    //populated size of cmd_parm up to max_params
char cmd_parm[CL_MAX_PARAMS_NUM][CL_MAX_PARAM_LEN]; //up to six
parameters, of twelve characters each, separated by commas or
spaces

unsigned int cl_i, cl_j, cl_k, cl_l;
char cl_in_ch;

char process_command_line(){

  if (Serial.available())
  {
    //Serial.println("Receiving....");

    delay(500);
    cl_i=0;
    cl_j=0;
    cl_k=0;
    cl_l=0;
    cmd_line[0]='\0';
    cmd_parm_str[0]='\0';
           // clear param string
    cmd_n_parms=0;
    for (int x;x<CL_MAX_PARAMS_NUM;x++)
    {
      cmd_parm[x][0]=0; // clear out previous parameters
    }

    //get a command line
    boolean first_word=true;
    while (Serial.available())
    {
      //cl_in_ch = upper(Serial.read());
      cl_in_ch = Serial.read();
      cmd_line[cl_i] = cl_in_ch;
      if (first_word && (cl_in_ch == ' '))
      {
        first_word=false;
        cmd_str[cl_i]='\0'; // add null terminator
        cl_i++;
        continue;
      }
      if (first_word) cmd_str[cl_i]=cl_in_ch;
      else // build cmd_parm_str
      {
        cmd_parm_str[cl_j++]=cl_in_ch;
        if (cl_in_ch==' ' || cl_in_ch==',')
        {
          cmd_parm[cl_k][cl_l]='\0';
          cl_k++;
          cl_l=0;
        }
        else
        {
          cmd_parm[cl_k][cl_l++]=cl_in_ch;
          cmd_n_parms++;
        }
      }
      cl_i++;
    } // end while

    if (!first_word)
    {                              // we had parameters so...
      cmd_parm_str[cl_j]='\0';   // null terminate param string
      cmd_parm[cl_k][cl_l]='\0';    // and the last param
      cmd_n_parms = cl_k+1;   // remember number of parameters (k
doesn't get a chance to increment for last parm so add 1

        // we have a command line
      return 1;

    }
    else
    {
      cmd_str[cl_i]='\0';          // terminate command str which
normally happens when we start parsing parameters

      // we don't have a command line
      return 0;
    }

  }

}
```

# Pump control circuit

## Overview

This is a simple pump trigger circuit to allow pump activation from either the PLC +5 volt output pin or a manual button mounted on the control case. The pumps require a +12v power source. +5v is used simply to trigger the circuit and not run the device.



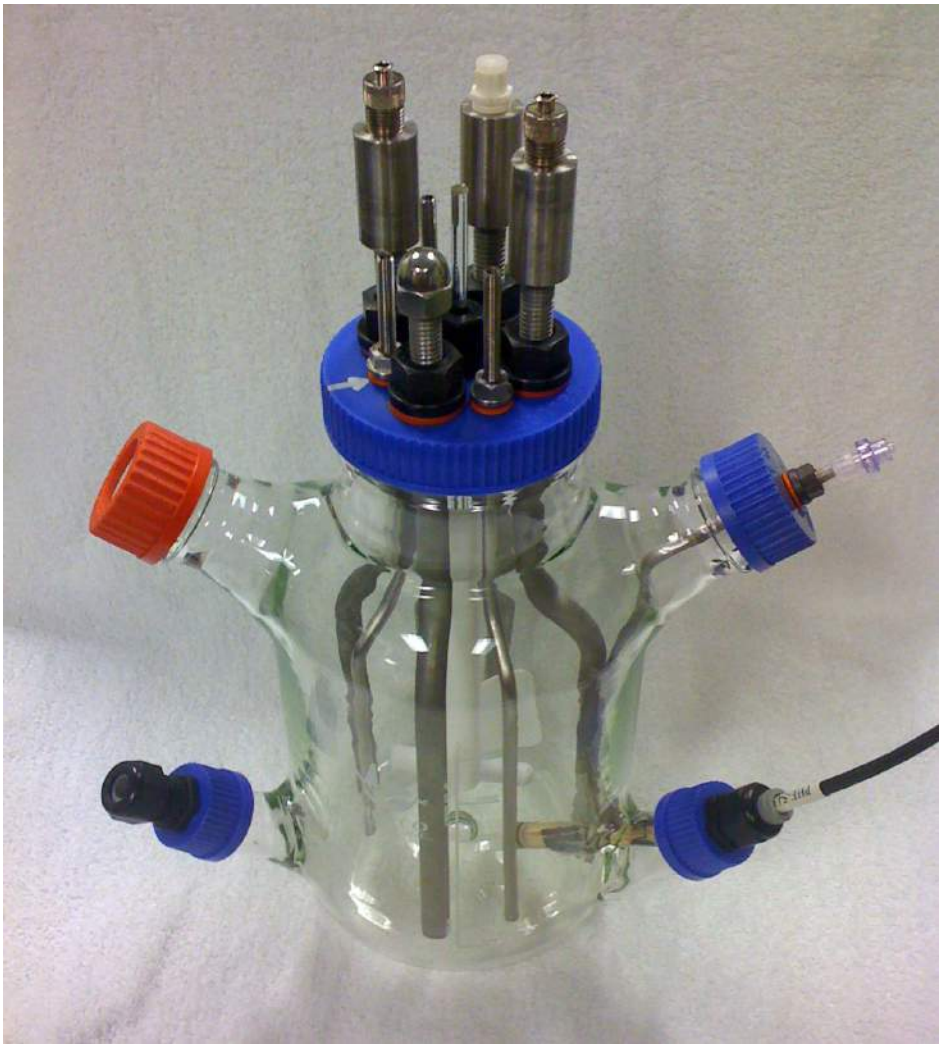
# Vessel schematics

## Overview

The vessel was designed using an off-the-shelf 3 liter spinner flask. The flask has four ½ inch and four ¼ inch steel pipes through the cap, and one ¼ inch steel pipe at the upper sidearm used exclusively for sampling. The second upper sidearm houses a large gauge filtered port for ventilation. This will ultimately be replaced with a forced air system to sweep the headspace.

The pipes are threaded to allow medical grade luer-lock fittings for medical tubing attachment and one way flow check-valves. Removable luer attachments were chosen for disposability, sterility, and so flow direction was non-dedicated and pipes could be allocated for different uses based on circumstance.

The lower sidearms are dedicated for probe housing. Shown is the Omega PHE-5431-10 high temperature/pressure pH electrode. The probe housings are high temperature/pressure marine hull-cable-run anc-765000. Silicon o-rings provide the sealing.



Click to Enlarge

## Liquid Tight Wire Seal

Provides low cost watertight cable entries into panels and junction boxes. UL and CSA approved to work with round and flat cable. Resistant to saltwater, gasoline, alcohol, ketones and mineral oil. Safe underwater to 300 ft. (150 PSIG).

| | |
|---|---|
| Vendor: | Ancor Marine Grade Products |
| SKU: | ANC 765000 |
| Weight: | .060 |
| NPT Size" | 1/2 |
| Min./Max. | .05 - .47 |
| AWG Range | #20-#4 |
| Description | Round |
| Cost: | **$15.29** |
| Quantity: | 1 |

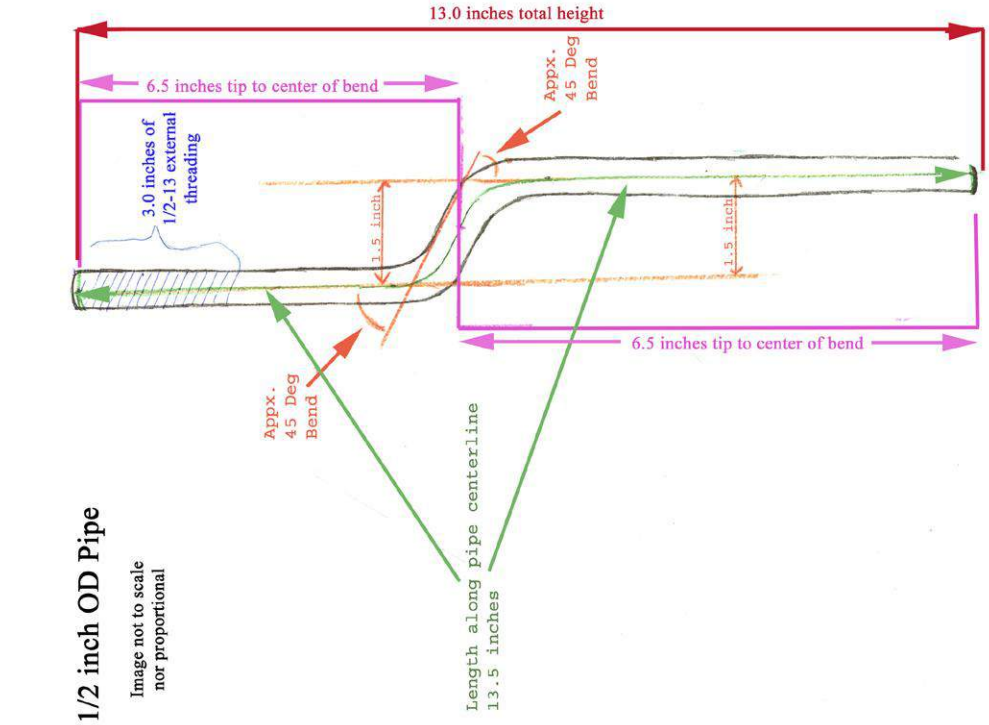
Add to Cart

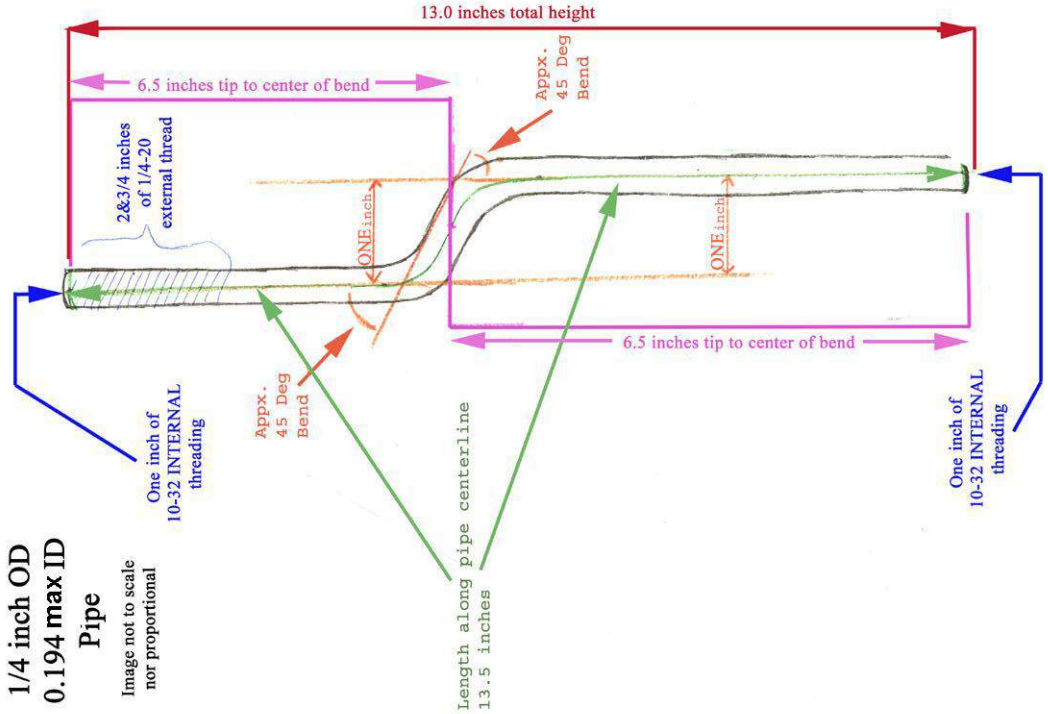
**Product Pictures:**

anc-765000 autoclavable marine hull cable run

# Piping

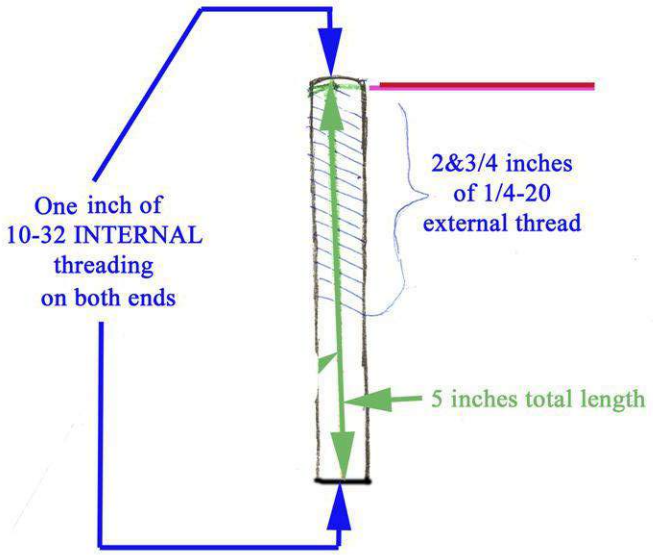
## Schematics



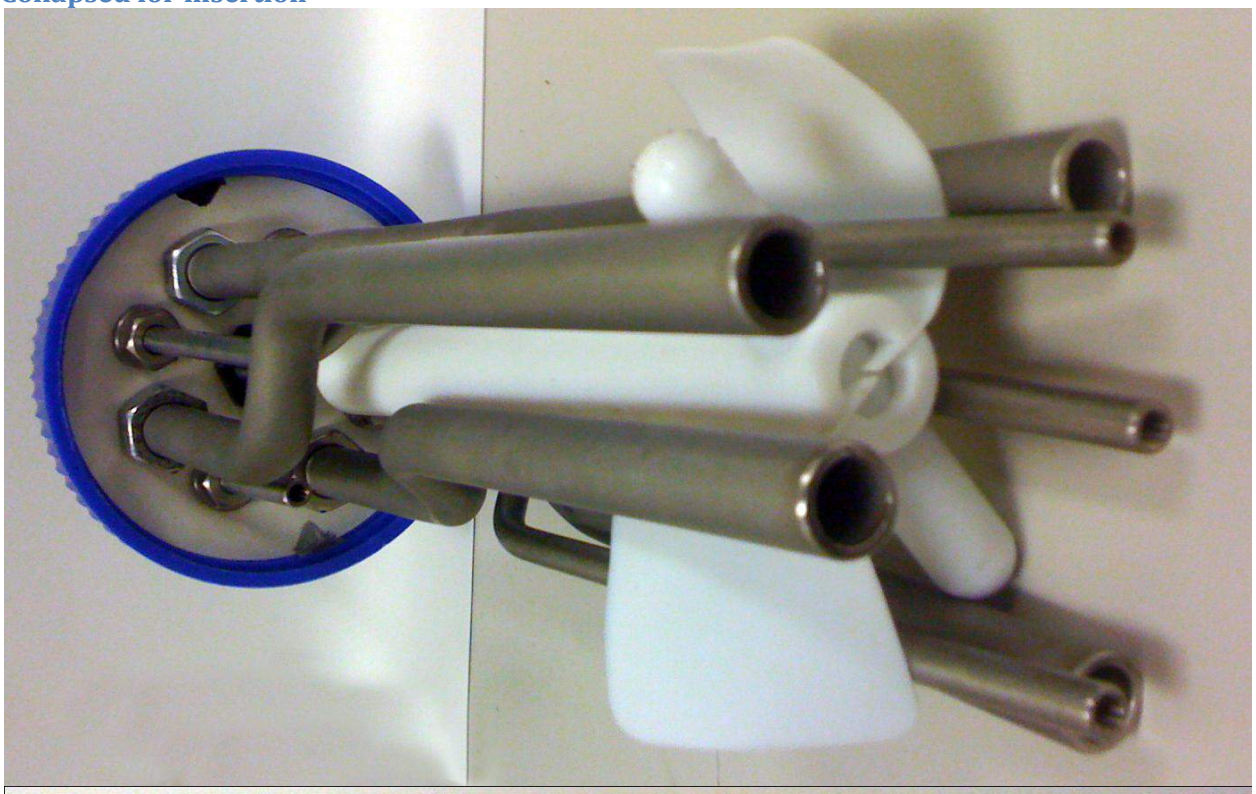
# 1/4 inch OD
# 0.194 max ID
# Short Pipe

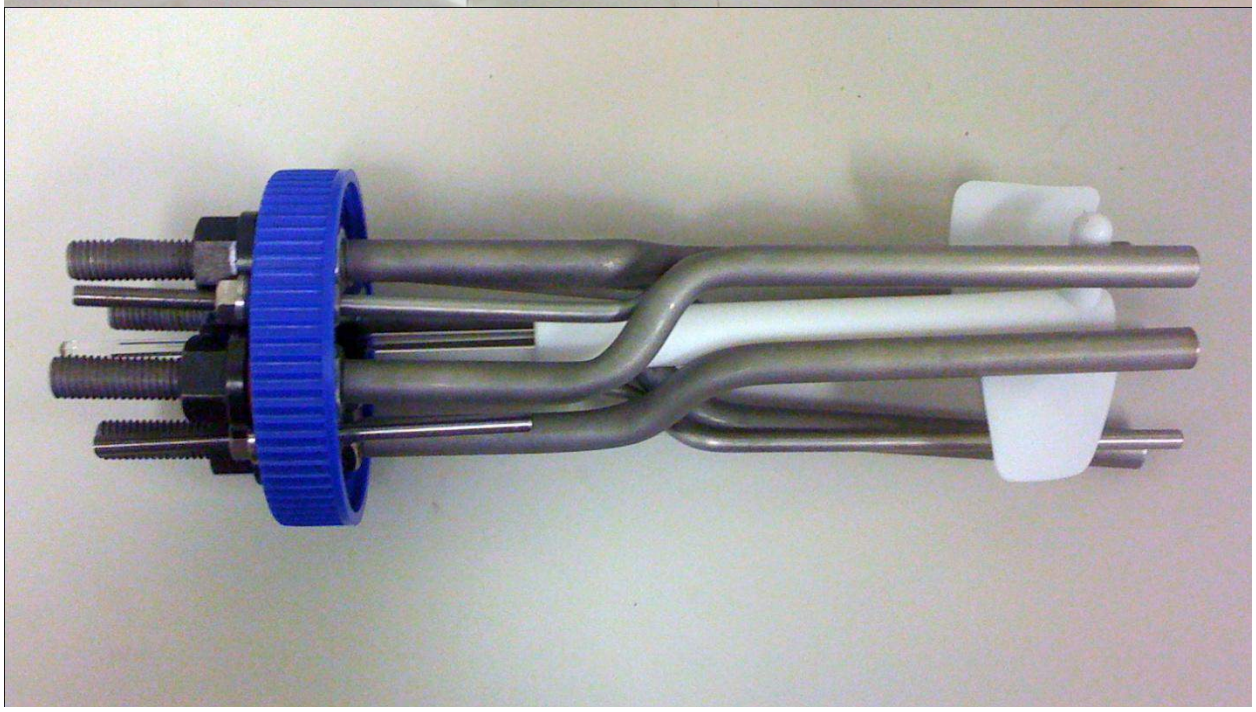
Image not to scale
nor proportional



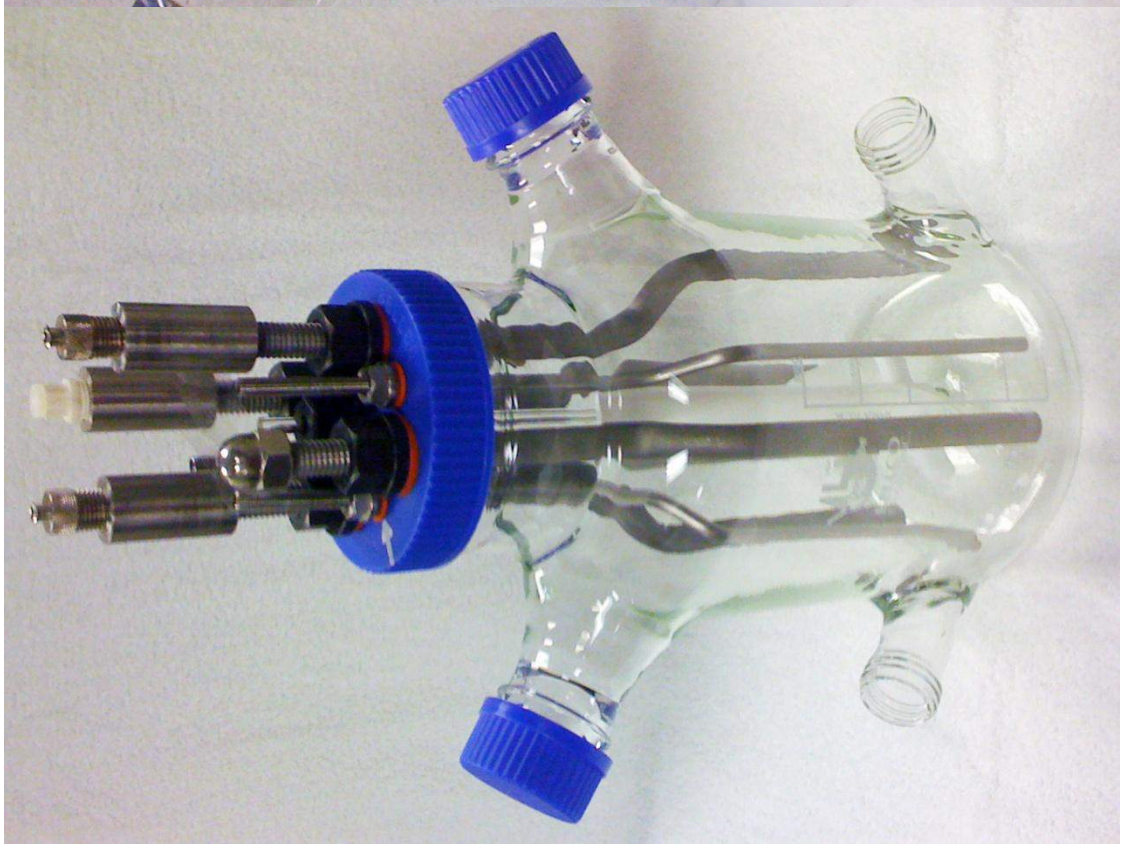
## Pipe assembly

### Collapsed for insertion

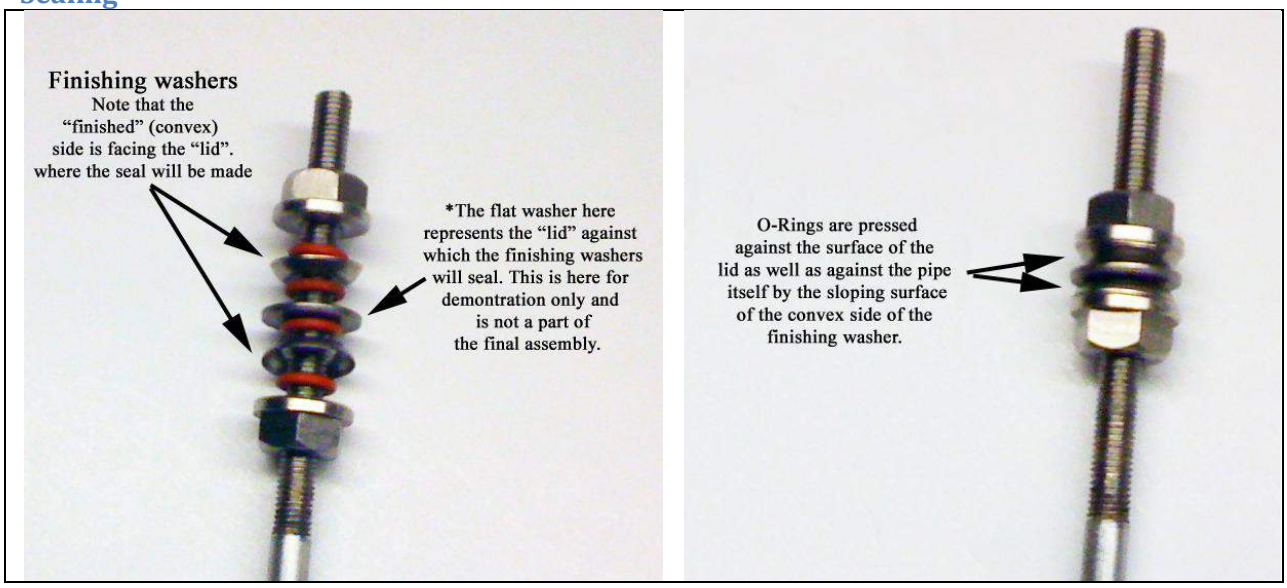




**Inserted**

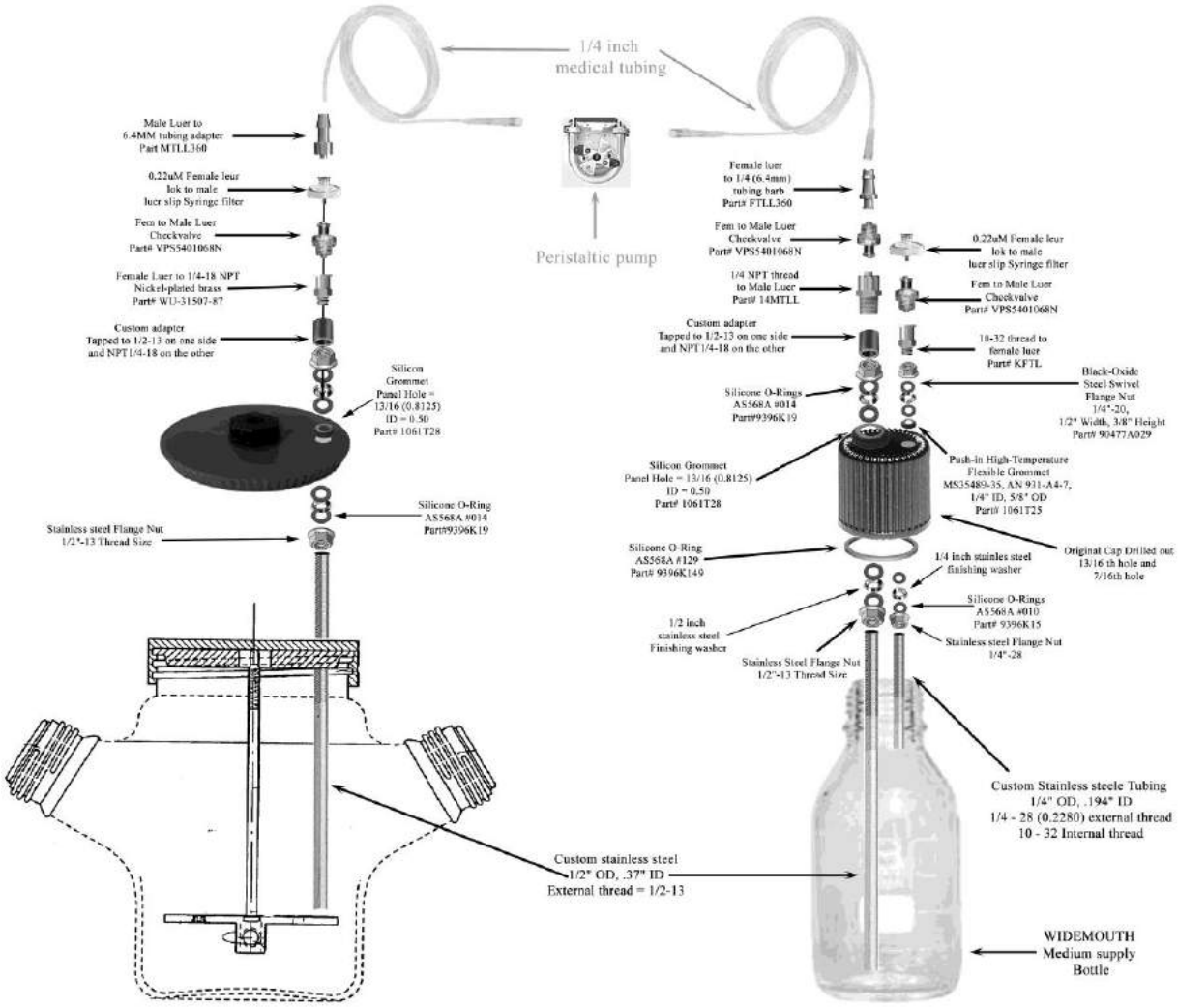


## Sealing


Finishing washers
Note that the "finished" (convex) side is facing the "lid". where the seal will be made
*The flat washer here represents the "lid" against which the finishing washers will seal. This is here for demontration only and is not a part of the final assembly.
O-Rings are pressed against the surface of the lid as well as against the pipe itself by the sloping surface of the convex side of the finishing washer.


Teflon flat washer

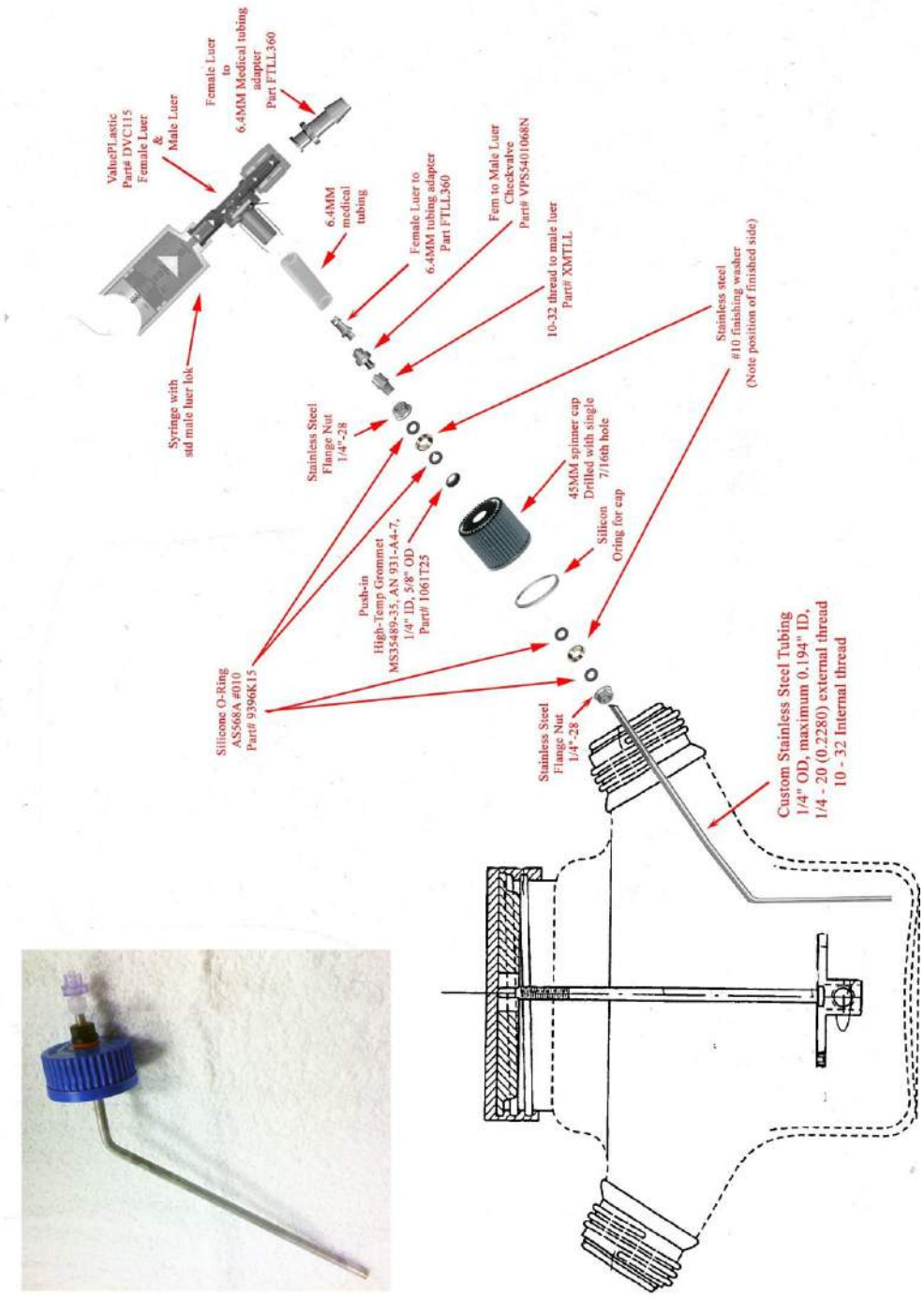
## Medium addition



# Sampling

## Overview

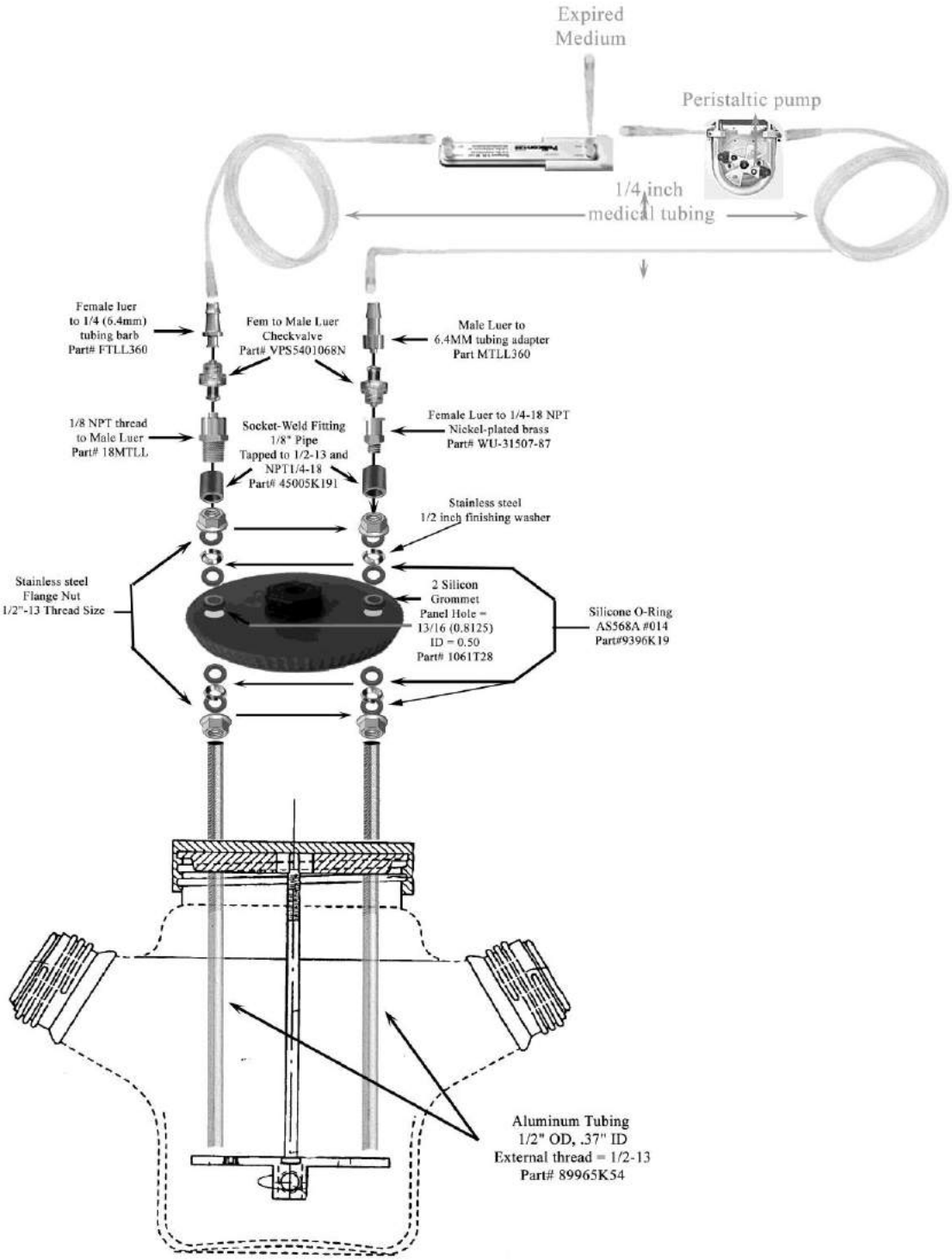
A significant issue has been contamination whilst adding medium or sampling for cell density. The sampling system uses off-the-shelf, medical-grade, luer one-way check valves to withdraw active medium without potential backwash from the exterior. This has been successfully tested in extremely contaminated environments (active bacterial incubators).



# Filtration

## Overview

Pressures were tested up to pump maximum generation with $H_2O$.



# Spinner paddle

## Overview

This impeller design is a “proof of concept”. It was hand-shaped from teflon, and is to be replaced with a stainless steel version after flow patterns can be verified.

The goal is not only dis-aggregation of cells, but a continuous up-draft of medium along the shaft and down-draft returning along the vessel sides. This constantly brings $CO_2$ saturated medium to the surface where forced ventillation can dissipate the $CO_2$ and allows for additional $O_2$ absorption. This may or may not be sufficient to avoid $O_2$ sparging.

Should this design be insufficient for $CO_2$ removal, a full screw-type impeller reaching all the way to the surface my be employed. However, this is non-optimal as it removes room for additional pipes and sensors.

# pH subsystem

## Overview

pH balance is, of course, critical for cell culture. This system uses a high temperature/pressure pH electrode which communicates directly with the PLC. Based on readings, the PLC will trigger drop-wise addition of NaOH solution every 30-60 seconds. HCl addition can be used in conjunction however is not recommended. In a healthy vessel, pH creep is always to the acidic and allowing for both base and acid additions creates the possibility of “chasing one’s tail” in a constant oscillation.

This probe was chosen for its (and its cable’s) ability to withstand autoclaving, and thus avoid the need for assembly in a class II hood after sterilization. The #1058_0-PhidgetPhSensor transducer is used to interface the BNC probe with the PLC board.

**INDUSTRIAL pH INSTRUMENTATION & ELECTRODES**

***High Temperature Insertion Type Electrode***

**PHE-5431-10 Series**

MADE IN USA RoHS

- **High Temperature**
- **High Pressure**
- **Steam Sterilization**
- **Easy Installation**

***Applications***

- **High-Temperature Environment**
- **Continuous Processing Applications**
- **Harsh Conditions**
- **Steam Sterilization**

The high temperature insertion type electrode is designed for periodic exposure to steam sterilization or continuous high temperature. This pH electrode is housed in a durable thermoplastic (PAS) body and is rated for temperatures up to 135°C (275°F). The use of the double Porous PTFE liquid junctions with matched viscosity electrolytes provides a reference cell which permits extended periods of pH measurements in the presence of sulfides or other silver complexing agents. This electrode also incorporates pressure compensating devices to protect the electrode from extreme temperature induced pressure deviations. Ten foot cable length and BNC connector are standard.

**Specifications**

**pH Range:** 0 to 14 pH
**Temperature Range:** -5 to 135°C (25 to 275°F) @ 25 psig
**Pressure Range:** 500 psig @ 25°C
**Accuracy:** ±0.1% over full range
**Impedance:** 125 Ω (standard version)
**Reference Cell:** Double junction, KCl/AgCl, $KNO_3$
**Reference Junction:** Porous PTFE
**Wetted Materials:** PTFE, glass membrane, polymer outer body, EPR "O" Rings
**Response Time:** 10 seconds to 95% of reading
**Drift:** Less than 2 mV per week

PHE-5431-10-T shown smaller than actual size.



PHE-5431-10 shown smaller than actual size.

**To Order** ***Visit omega.com/phe543110 for Pricing and Details***

| Model No. | Description |
|---|---|
| PHE-5431-10 | High temperature insertion type pH electrode |
| PHE-5431-10-(*) | High temperature insertion type pH electrode with ATC |
| PHE-5431-10-T | T-handle high temperature pH electrode |
| PHE-5431-10-T-(*) | T-handle high temperature pH electrode with ATC |
| ORE-5431-10 | High temperature insertion type ORP electrode |
| ORE-5431-10-T | T-handle high temperature ORP electrode |

** Specify ATC sensor: "-PT100" for 100 Ω Pt RTD or "-PT1K" for 1000 Ω Pt RTD, for additional cost.*
*Note: PHEH-51 (1/2 MNPT) installation fitting is required for first time installation see omega.com for installation fitting. Comes complete with operator's manual.*
*Ordering Examples: PHE-5431-10, high temperature pH electrode. PHE-5431-10-T, T-handle high temperature pH electrode, and PHEH-51.*



*Our Industrial pH Instrumentation and Electrodes product line continues to expand, visit omega.com for new details!*

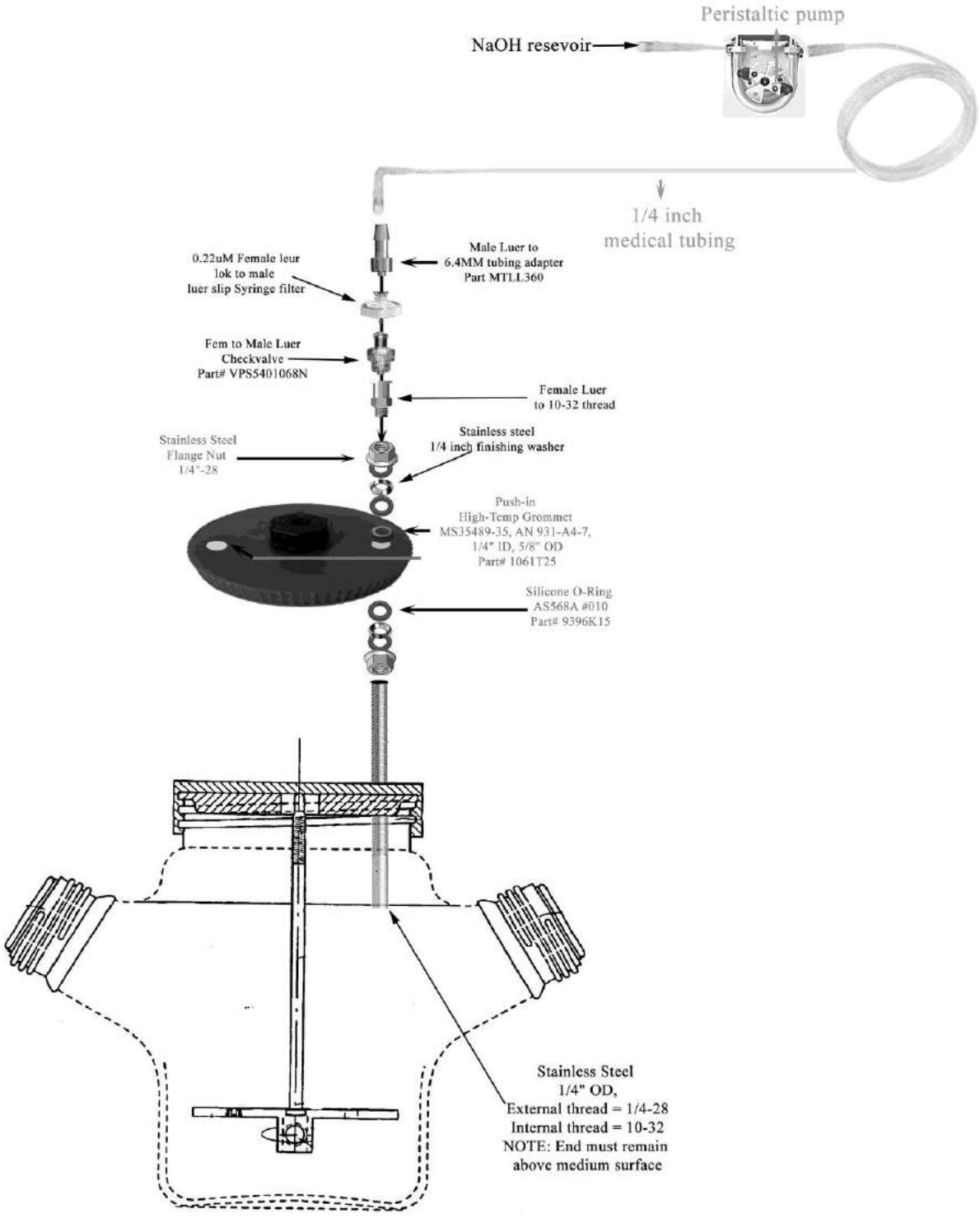
For Sales & Service 1-800-82-66342 1-800-TC-OMEGA® U.S.A. and Canada

Cable secured (assembled)


Cable unsealed (Disassembled)

The #1058_0-PhidgetPhSensor transducer

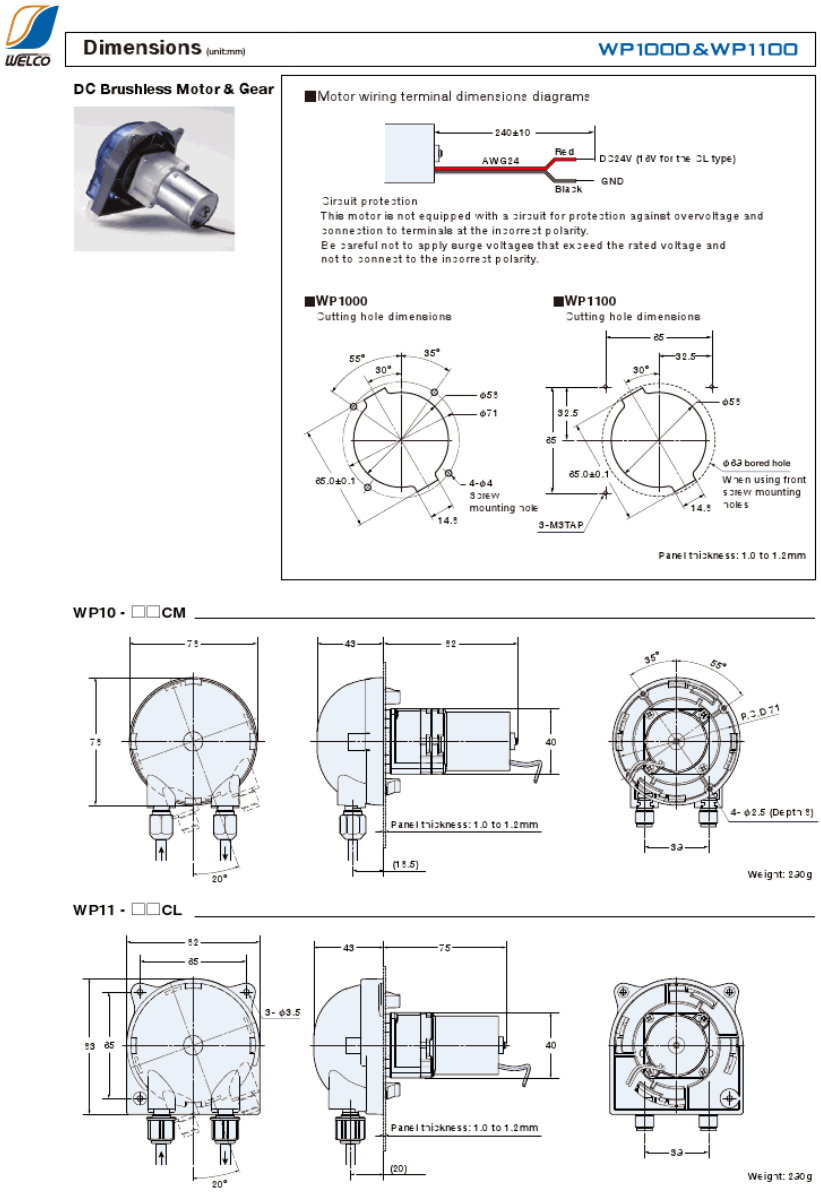
## NaOH addition line



# Peristaltic pumps

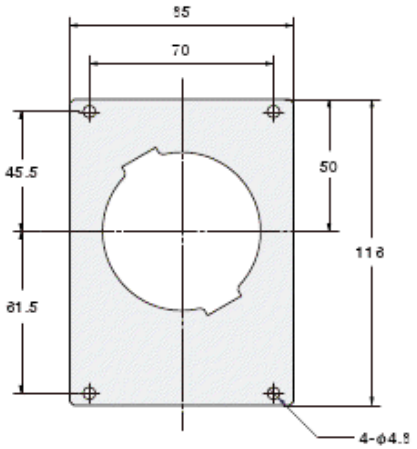
## Overview

Two Welco WP1000 peristaltic pumps (DC motor 8-24V/20-150RPM, WP10-P1/4M2-W4-CP) reside on the control box and are mounted in with the optional mounting panel. ¼ inch sterilized medical tubing is threaded through the pumps and attached to vessel pipes with medical grade luer locks. “zip-tie” secure ties on the tubing/luer connection is recommended to add additional slip-resistance for high pressure applications (including the medium addition and NaOH systems). In this configuration, pressure has been field tested to the maximum capability of the pump. The pumps are driven by outside power directed by the Pump control circuit (see above). The pump can be triggered by either the PLC or a manual button on the control case.



## Using an optional panel

Thickness: 1.2
Material properties: SUS304



# Heating

## Overview

The heating system uses an external, wrap around spinner flask heating element. Heating control is provided by an external Auqualogic thermostat containing its own high temperature probe, and control circuitry. The PLC is not needed for temperature control.

Aqualogic TR115SN

Part #: 7909-23115

aqualogic

**TR115SN, TR230SN, EC115R and EC230R Controller Operation and programming:**

| Step | Enunciator | Description | Display |
|---|---|---|---|
| 1 | F or C | Fahrenheit or Celsius | F |
| 2 | S1 (Blinking) | Setpoint Temperature | S1 77 |
| 3 | DIF (Blinking) | Differential Temperature | DIF 1 |
| 4 | C1 or H1 | Cooling or Heating Mode | C1 |


**Liquid Crystal Display (LCD)**

The LCD display provides a constant readout of the sensor temperature and indicates if the output relay is energized. When the **S1** enunciator is constantly Illuminated during operation, the relay is energized. the display is also used in conjunction with the keypad to allow the user to adjust the set point temperature, differential and heating /cooling modes.

**Programming Steps and Display**

The control can be programmed in four simple steps using the LCD display and the three keys on the face of the control. (See photo for display and keys.)

1. To start programming, press the **SET** key once to access the Fahrenheit/Celsius mode. The display will show the current status, either **F** for degrees Fahrenheit or **C** for degrees Celsius. Then press either the up ⇧ arrow or down ⇩ arrow key to toggle between the **F** or **C** designation.
2. Press the **SET** key again to access the set point temperature. The LCD will display the current set point temperature and the set point enunciator will be blinking on and off to indicate that the control is in the set point mode. Then press either the up ⇧ key to increase or down ⇩ key to decrease the set point to the desired temperature.
3. Press the SET key again to access the differential. The LCD will display the current differential and the **DIF** enunciator will be blinking on and off to indicate that the control is in the differential mode. Then press either up ⇧ key to increase or the down ⇩ key to decrease the differential to the desired setting (minimum 1℉, maximum 30℉).
4. Press the **SET** key again to access the heating mode. The LCD will display the current mode, **C1** for chiller mode and **H1** is for heater mode. Press the **SET** key once more and programming is complete. Controller **MUST** in the **C1** mode for correct operation.

**Controller will automatically drop out of "program mode" and return to "operating mode" 30 seconds after last key press.**

**Troubleshooting Controller Error Messages:**

**Display Messages**

- **E1** - Appears when the up ⇧ or down ⇩ key is pressed when not in the programming mode.
  **To correct:** If the E1 message appears even when no keys are being pressed, replace the control.
- **E2** - Appears if the control settings are not properly stored in memory.
  **To correct:** Check all settings and correct if necessary.
- **EP** - Appears when the probe and or flow switch is open , shorted or sensing a temperature that is out of range.
  **To correct:** Check to see if the sensed temperature is out of range. If not , check for probe damage by comparing it to a known ambient temperature between -30℉ and 220℉. Replace the probe is necessary. Also check for proper water flow through heater. If water flow is correct, flow switch.
- **EE** - Appears if the EEPROM data has been corrupted.
  **To correct:** This condition cannot be field repaired. Replace the control.
- **CL** - Appears if calibration mode has been entered.
  **To correct:** Remove power to the control for least five seconds. Reapply power. If the **CL** message still appears, replace the control.

**Aqua Logic®, Inc.**
8268 Clairemont Mesa Blvd. Suite 302 San Diego, CA 92111
Tel: (858) 292-4773 Fax: (858) 279- 0537 Email: info@aqualogicinc.com www.aqualogicinc.com