Supporting Information

Scale-Up of N-Alkylation Reaction using Phase-Transfer Catalysis with Integrated Separation in Flow

María José Nieves-Remacha,*[a] Myriam Torres,[a,b] María Ruiz-Abad,[a,c] and Juan A. Rincón[a], Graham R. Cumming[a], and Pablo Garcia-Losada*[a]

[a.] *Centro de Investigación Lilly S.A., Avda. de la Industria, 30, 28108 Alcobendas (Madrid) Spain.*
[b.] *Present address: Centro de Investigaciones Científicas Isla de la Cartuja, Avda. Americo Vespucio 49. 41092 Sevilla (Spain)*
[c.] *Present address: Chemical and Environmental Engineering Group, Universidad Rey Juan Carlos, Calle Tulipán s/n, 28933 Móstoles (Madrid), Spain*

Index Page

1. **Batch experiments for N-alkylation in DMSO** **2**
2. **Batch phase transfer catalysis experiments** **3**
3. **Fill-empty setup** **4**
4. **Control software for fill-empty** **5**
5. **Phase transfer catalysis in coil reactor** **7**
6. **Mass transfer experiments in coil reactor** **10**
7. **NMR spectra for selected compounds** **11**
8. **Appendix A: Source code for software** ***12***

1. Batch experiments for N-alkylation in DMSO

Described as Alternative Method#1 in Scheme 2, the substrate undergoes N-alkylation in DMSO in presence of a strong base at ambient conditions and 5-10 number of equivalents of alkylating agent. SI Table 1 summarizes the batch experiments conducted to screen reaction conditions following the experimental procedure described here.

Experimental Procedure. In a typical experiment, to a 50 mL screw-cap vial 2-bromo-6,6-dimethyl-5H-thieno[2,3-c]pyrrol-4-one (100 mg, 0.40 mmol, 1.0 equiv), potassium hydroxide (122 mg, 2.17 mmol, 5.35 equiv), and ethanol (1.7 mL, 29 mmol, 72 equiv) were added. The mixture was stirred during 5 min at room temperature (r.t.). Then 1-bromo-2-methoxy-ethane (100 μL, 0.72 mmol, 1.8 equiv) was added and stirred at r.t. during 1h. A sample was analysed by LC/MS. The reaction mixture was then heated to 60ºC and then analysed by LC/MS. The reaction was further heated during 2 h at 90ºC and then analysed by LC/MS.

SI Table 1: Batch screening experiments in DMSO

Base	pKa	Solvent	1h/22ºC	2h/60ºC	2h/90ºC	4h/90ºC
KOH	27	DMSO	95%	n.a.	n.a.	n.a.
t-BuOK	17	THF	10%	92%	n.a.	n.a.
KOH	17	EtOH	2%	8%	5%*	n.a.
K2CO3	10.2	DMF	1%	12%	28%	n.a.
K2CO3	10.2	Acetone	0%	0%	7%	n.a.
TEA/KI	10.7	DMF	0%	0%	n.a.	n.a.
K3PO4	12	DMSO	8%	n.a.	70%	80%
KOH	17	Dioxane	0%	n.a.	86%	n.a.
t-BuOK	17	Dioxane	7%	n.a.	76%	n.a.
K3PO4	12	Dioxane	6%	n.a.	16%	n.a.

2. Batch phase transfer catalysis experiments

The experiments performed in batch for the phase-transfer catalysis approach are included in SI Table 2**.**

SI Table 2: Batch screening results for phase-transfer catalysis N-alkylation

Amount	Solvent	Catalyst	Equiv.Cat	Base	% S.M	% Product	% Others	T (ºC)	Time
1	Toluene	Bu4N+ Cl-	0.5	NaOH 50%	-	98	-	25	overnight
1	Toluene	Bu3BnN+ Cl-	0.5	NaOH 50%	4	96	-	25	overnight
1	Toluene	Et3BnN+ Cl-	0.5	NaOH 50%	51	48	-	25	overnight
1 (100 mg)	Toluene	Hex4N+ Cl-	0.5	NaOH 50%	-	99	-	25	overnight
1 (100 mg)	THF	Bu4N+ Cl-	0.5	NaOH 50%	-	98	-	25	after 3h
1 (100 mg)	THF	Bu3BnN+ Cl-	0.5	NaOH 50%	32	67	-	25	after 3h
1 (100 mg)	THF	Et3BnN+ Cl-	0.5	NaOH 50%	32	67	-	25	overnight
1 (100 mg)	THF	Hex4N+ Cl-	0.5	NaOH 50%	-	98	-	25	overnight
1 (100 mg)	DCM	Bu4N+ Cl-	0.5	NaOH 50%	6	66	25	25	overnight
1 (100 mg)	DCM	Bu3BnN+ Cl-	0.5	NaOH 50%	5	62	21	25	overnight
1 (100 mg)	DCM	Et3BnN+ Cl-	0.5	NaOH 50%	-	77	23	25	overnight
1 (100 mg)	DCM	Hex4N+ Cl-	0.5	NaOH 50%	-	74	26	25	overnight
1 (100 mg)	THF	Bu4N+ Cl-	0.5	NaOH 50%	68	32	-	25	after 1h
1 (100 mg)	Toluene	Bu4N+ Cl-	0.5	KOH 50%	52	48	-	25	after 1h
1 (100 mg)	THF	Bu4N+ Cl-	0.5	NaOH 50%	49	51	-	25	after 2h
1 (100 mg)	Toluene	Bu4N+ Cl-	0.5	KOH 50%	27	73	-	25	after 2h
1 (100 mg)	THF	Bu4N+ Cl-	0.2	NaOH 50%	3	97	-	25	overnight
1 (100 mg)	THF	Bu4N+ Cl-	0.1	NaOH 50%	23	77	-	25	overnight
1 (100 mg)	THF	Bu4N+ Cl-	0.5	NaOH 50%	17	83	-	25	overnight
1 (100 mg)	Toluene	Bu4N+ Cl-	0.5	NaOH 50%	3	97	-	25	overnight
2 (100 mg)	THF	Bu4N+ Cl-	0.5	NaOH 50%	-	99	-	25	overnight
1 (100 mg)	MeTHF	Bu4N+ Cl-	0.2	NaOH 50%	61	39	-	25	overnight
1 (100 mg)	CPME	Bu4N+ Cl-	0.2	NaOH 50%	61	39	-	25	overnight
1 (100 mg)	MTBE	Bu4N+ Cl-	0.2	NaOH 50%	15	85	-	25	overnight
1 (2g)	THF	Bu4N+ Cl-	0.2	NaOH 50%	98	2	-	25	after 1h
1 (100 mg)	THF	Bu4N+ Cl-	0.2	NaOH 50%	95	5	-	25	after 2h
1 (100 mg)	THF	Bu4N+ Cl-	0.2	NaOH 50%	95	5	-	25	after 5h
1 (100 mg)	THF	Bu4N+ Cl-	0.2	NaOH 50%	87	13	-	25	overnight
1 (100 mg)	THF	Bu4N+ Cl-	0.2	NaOH 25%	83	17	-	25	overnight
1 (100 mg)	THF	Bu4N+ Cl-	0.2	KOH 25%	85	15	-	25	overnight
1 (100 mg)	THF	Bu4N+ Cl-	0.2	NaOH 10%	94	6	-	25	overnight
2 (2g)	THF	Bu4N+ Cl-	0.5	NaOH 50%	28	72	-	25	after 1h
2 (2g)	THF	Bu4N+ Cl-	0.5	NaOH 50%	8	92	-	25	after 2h
2 (2g)	THF	Bu4N+ Cl-	0.5	NaOH 50%	1	99	-	25	after 5h
2 (2g)	THF	Bu4N+ Cl-	0.5	NaOH 50%	-	100	-	25	overnight
1 (2g)rep	THF	Bu4N+ Cl-	0.2	NaOH 50%	75	25	-	25	after 1h
1 (2g)rep	THF	Bu4N+ Cl-	0.2	NaOH 50%	51	50	-	25	after 3h
1 (2g)rep	THF	Bu4N+ Cl-	0.2	NaOH 50%	39	61	-	25	after 5h
1(100 mg)	THF	Bu4N+ Cl-	0.5	NaOH 50%	34	66	-	35	after 1h
1(100 mg)	THF	Bu4N+ Cl-	0.5	NaOH 50%	13	87	-	50	after 1h
1(100 mg)	THF	Bu4N+ Cl-	0.5	NaOH 50%	1	99	-	35	overnight
1(100 mg)	THF	Bu4N+ Cl-	0.5	NaOH 50%	-	100	-	50	overnight

3. Fill-empty setup

SI Table 3 includes the equipment and main parts used in our experimental setup for the fill-empty reactor. The transfer of materials between vessels is performed by peristaltic pumps. Other approaches for the fill-empty setup use several

valves and perform the transfer via nitrogen feed and vacuum. With such valves, the transfer is faster and is more suitable when there is presence of solids. The setup proposed in our experiment is a simplified approach, with less required automation, and the technology can be easily transferred.

SI Table 3: Components of Fill-Empty setup

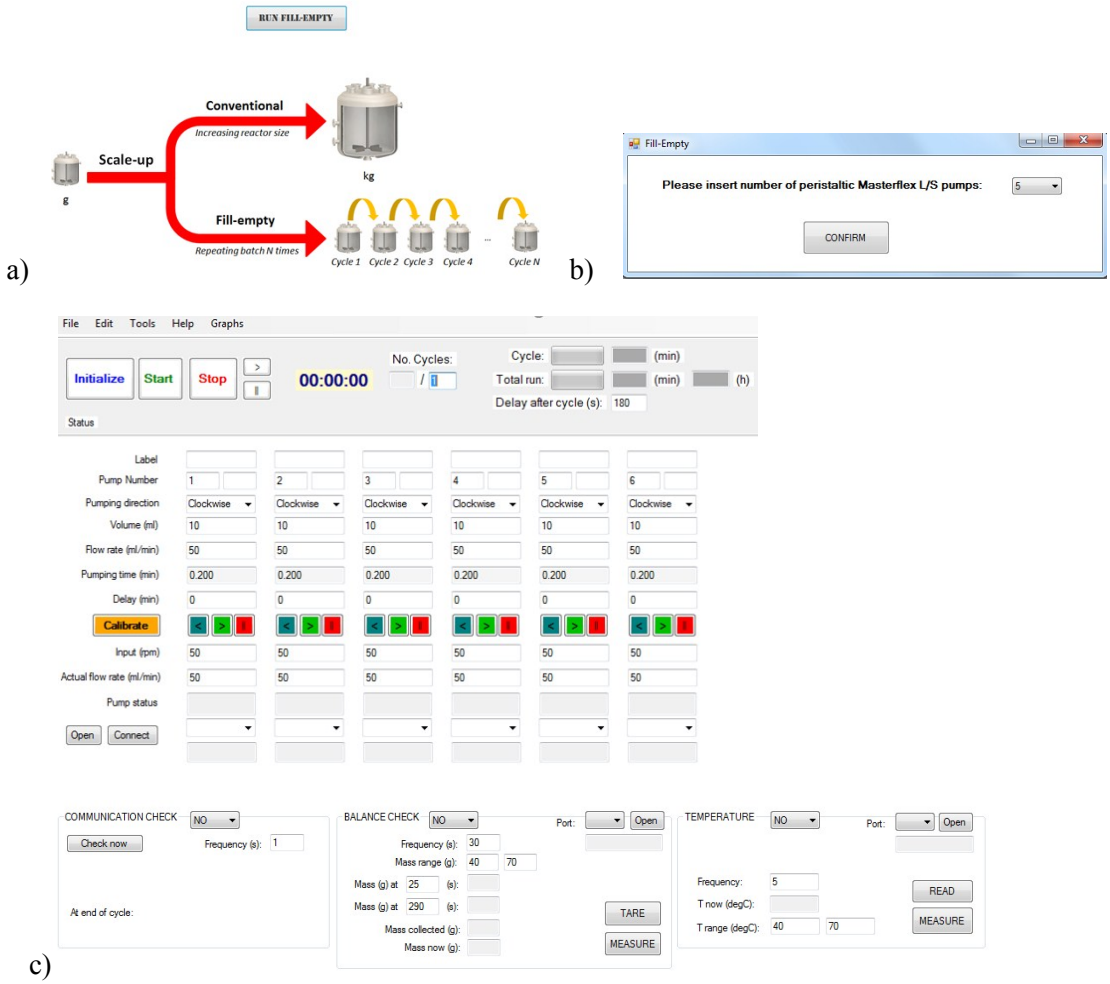
No.	Equipment	Model/Vendor	Purpose	Material
1	Peristaltic pump	Masterflex L/S	Feed	NaOH
2	Peristaltic pump	Masterflex L/S	Feed	Catalyst
3	Peristaltic pump	Masterflex L/S	Feed	Substrate
4	Peristaltic pump	Masterflex L/S	Dilutor	Water
5	Peristaltic pump	Masterflex L/S	Transfer	Reactor to dilution vessel
6	Peristaltic pump	Masterflex L/S	Transfer	Dilution vessel to separator
7	Thermocouple	Omega UTC-USB	Measure	Reactor temperature
8	Jacketed reactor	Spinchem	Mix	Reactor contents
9	Heater	Huber Ministat 125	Heat	Reactor
10	Bottle	ChemGlass	Feed	NaOH soln.
11	Bottle	ChemGlass	Feed	Catalyst soln.
12	Bottle	ChemGlass	Feed	Substrate + alkylating agent soln.
13	Bottle	ChemGlass	Product	Organic phase
14	Bottle	ChemGlass	Waste	Aqueous phase
15	Dilution flask	ChemGlass	Mix	Water + Multiphase mixture from reactor
16	Knock-out vessel	ChemGlass	Safety	Collects reactor contents
17	Continuous decanter	Custom-made	Separate	Organic from aqueous phase
18	Agitator	-	Agitate	Multiphase mixture in reactor

4. Control software for fill-empty

The control software for the fill-empty was developed in .NET C# using Microsoft Visual Studio[1]. The computer is the controller that sends the commands to the equipment and monitoring devices to request information or a specific action[2]. All pumps are peristaltic Masterflex L/S PC-compatible and are connected to an 8-port RS232-to-USB hub

[1] Microsoft Visual Studio, 2015, Microsoft, Redmond, WA.

(StartTech) which is connected to the PC through the USB port. A balance and a temperature module are also connected to the PC via RS-232.The software is composed by several panels: a) welcome page; b) selection of number of peristaltic pumps; c) main control panel (Figure SI 1).



SI Figure SI 1: Main screenshots of the control software program: a) Home page; b) Setup page for the selection of number of pumps; c) Main panel

The number of pumps is user-defined, which confers the program flexibility for the straightforward automation of different experimental setups (in our setup, 6 pumps were used). Each pump has a set of required inputs for the process run (label, pumping direction, volume per cycle to pump, flow rate, delay time to start the pump) and inputs for calibration (input revolutions per minute –RPM- and actual flow rate). Depending on the tubing size used, the number of RPM corresponding to a certain flow rate varies considerably (i.e. for STA Pure Tubing #17: 60 RPM correspond to approximately 180 ml/min, and for STA Pure Tubing #16, 50 RPM correspond to approximately 50 ml/min). For calibration purposes, the button "Calibrate" Calibrate starts the pumps and stops them after 1 minute running time. At the bottom of each pump there is also a dropdown list of the COM ports available. Each pump has a COM port which needs to be selected and opened for communication with the PC. There are also buttons to manually and remotely start (counter and clockwise) and stop each pump.

[2] J. Axelson. *Serial Port Complete,* LakeView Research LLC, Madison, WI, 2007

The number of cycles and a delay time after each cycle can also be specified if needed. With varying number of cycles, the total production rate can be adjusted according to the demand.

Regarding the software development for the control of the pumps a class of type user-control has been defined to allow the control of a user-defined number of peristaltic pumps. Thus, when the user specifies N peristaltic pumps, the software creates N user controls of class 'peristaltic pump', each one having the same control commands:

Interface	*Function*	*Units*	*Input/output*	*.NET Controls*
	Label	-	User-defined	textbox
	Pump number	-	Generated	textbox
	Pumping direction	-	Selectable	comboBox
	Volume (ml)	ml	User-defined	textbox
	Flow rate (ml/min)	ml/min	User-defined	textbox
	Pumping time (min)	min	Calculated	textbox (read-only)
	Delay (min)	min	User-defined	textbox
	Controls	-	cw/cc/stop	buttons
	Input (rpm)	rpm	User-defined	textbox
	Actual flow rate (ml/min)	ml/min	User-defined	textbox
	Status	-	Read	textbox (read-only)
	Serial COM Port	-	Selectable	comboBox

SI Figure 2: User control for peristaltic pump

All commands are sent to the equipment in hexadecimal. While the temperature module only registers temperature, the balance has the option to "tare" as well as "measure" mass. The peristaltic pumps have more commands available: set flow rate (rpm), set pumping direction, establish connection, set pump number, set remote control, ask for pump status, start, and stop, among others.

SI Table 4: Typical commands for equipment

Equipment	**Vendor**	**Event**	**Command**
Peristaltic pump	Masterflex L/S	Connect pump	05
		Set pump number	02 50 30 3*n* 0d
		Set pump to remote	02 50 30 3*n* 52 0d
		Ask for pump status	02 50 30 3*n* 49 0d
		Start	02 50 30 3*n* 53 *pdir speed* 56 *nrev* 2e 30 30 47 0d
		Stop	02 50 30 3*n* 5a 0d
Temperature module	Omega	Measure	2a 47 31 31 30 0d
Balance	OHaus	Tare	54 0d 0a
		Measure	49 50 0d 0a

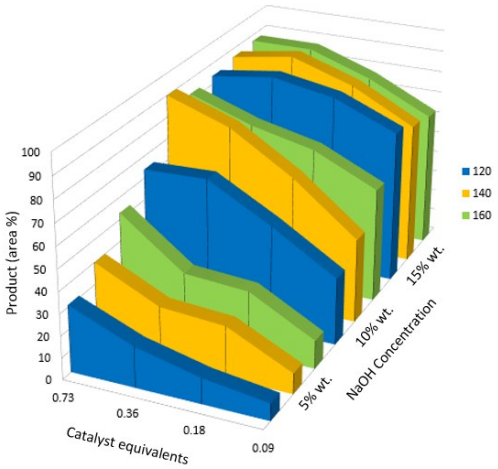
[1] *n*: pump number; [2] *pdir*: pump direction, "2b" if clockwise, "2d" if counterclockwise; [3] *speed*: rpm in hexadecimal ; [4] *nrev*: max number of revolutions to run pump;

The source code can be found at the end of this document as Appendix A.

5. Phase transfer catalysis in coil reactor

SI Figure 3 summarizes several results obtained in a stainless steel 20 ml coil reactor for the phase-transfer catalysis experiments conducted at high temperatures. The influence of the NaOH concentration and catalyst equivalents is also shown.

- 10% increase in conversion at low NaOH concentrations for temperature increase of 20 °C
- 30% increase in conversion at low temperature for NaOH concentration increase of 5%
- Increase in conversion with number of catalyst equivalents



SI Figure 3: Effect of NaOH concentration and catalyst equivalents on product % area by LCMS at different temperatures (°C) in a flow reactor. Legend: 120 °C, 140 °C, 160 °C

SI Table 5 includes the experiments performed to optimize reaction conditions in the stainless steel reactor, modifying: a) temperature, b) catalyst equivalents, c) NaOH equivalents, d) residence time.

SI Table 5: Results Flow Reactor Experiments for Phase-Transfer Catalysis

a) 5% NaOH

Catalyst conc. (M)	T (°C)	NaOH aq. (mL/min)	Substrate in toluene (mL/min)	Catalyst aqueous soln. (mL/min)	Equivalents NaOH (mmol NaOH/mmol substrate)	Equivalents catalyst (mmol catalyst/mmol substrate)	Residence Time (min)	Product (% area LCMS)	Reagent (% area LCMS)	Byproduct (% area LCMS)
0,4	120	0,7	1	0,2	8	0,73	10	30,72	69,28	0
0,2	120	0,7	1	0,2	8	0,36	10	18,79	81,21	0
0,1	120	0,7	1	0,2	8	0,18	10	12,38	87,62	0
0,05	120	0,7	1	0,2	8	0,09	10	8,08	91,92	0
0,4	140	0,7	1	0,2	8	0,73	10	40,36	59,64	0
0,2	140	0,7	1	0,2	8	0,36	10	26,46	73,54	0
0,1	140	0,7	1	0,2	8	0,18	10	24,15	75,85	0
0,05	140	0,7	1	0,2	8	0,09	10	9,75	90,25	0
0,4	160	0,7	1	0,2	8	0,73	10	64,52	35,48	0
0,2	160	0,7	1	0,2	8	0,36	10	30,99	69,01	0
0,1	160	0,7	1	0,2	8	0,18	10	28,93	71,07	0
0,05	160	0,7	1	0,2	8	0,09	10	12,82	87,18	0

b) 10% NaOH

Catalyst conc. (M)	T (°C)	NaOH aq. (mL/min)	Substrate in toluene (mL/min)	Catalyst aqueous soln. (mL/min)	Equivalents NaOH (mmol NaOH/mmol substrate)	Equivalents catalyst (mmol catalyst/mmol substrate)	Residence Time (min)	Product (% area LCMS)	Reagent (% area LCMS)	Byproduct (% area LCMS)
0,4	100	0,7	1	0,2	17	0,73	10	58,32	41,68	0
0,2	100	0,7	1	0,2	17	0,36	10	42,64	57,36	0
0,1	100	0,7	1	0,2	17	0,18	10	33,74	66,26	0
0,05	100	0,7	1	0,2	17	0,09	10	17,88	82,12	0
0,4	120	0,7	1	0,2	17	0,73	10	60,06	39,94	0
0,2	120	0,7	1	0,2	17	0,36	10	65,16	34,84	0

0,1	120	0,7	1	0,2	17	0,18	10	49,29	50,71	0
0,05	120	0,7	1	0,2	17	0,09	10	31,24	68,76	0
0,4	130	0,7	1	0,2	17	0,73	10	82,62	17,38	0
0,2	130	0,7	1	0,2	17	0,36	10	75,63	24,37	0
0,1	130	0,7	1	0,2	17	0,18	10	77,72	22,28	0
0,05	130	0,7	1	0,2	17	0,09	10	38,09	61,91	0
0,4	140	0,7	1	0,2	17	0,73	10	90,2	9,8	0
0,2	140	0,7	1	0,2	17	0,36	10	80,18	19,82	0
0,1	140	0,7	1	0,2	17	0,18	10	62,45	37,55	0
0,05	140	0,7	1	0,2	17	0,09	10	39,72	60,28	0
0,4	160	0,7	1	0,2	17	0,73	10	83,44	16,36	0,2
0,2	160	0,7	1	0,2	17	0,36	10	71,02	25,43	3,55
0,1	160	0,7	1	0,2	17	0,18	10	62,79	37,21	0
0,05	160	0,7	1	0,2	17	0,09	10	72,32	27,68	0
0,4	180	0,7	1	0,2	17	0,73	10	77,33	17,53	5,14
0,05	180	0,7	1	0,2	17	0,09	10	52,84	47,16	0

c) 15% NaOH

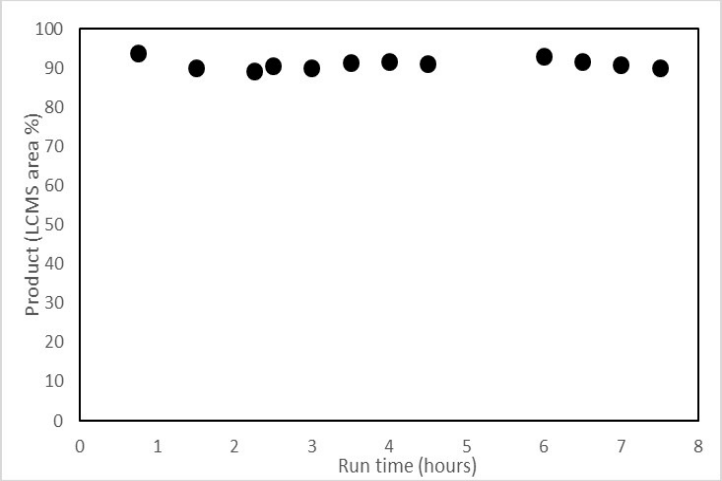
Catalyst conc. (M)	T (°C)	NaOH aq. (mL/min)	Substrate in toluene (mL/min)	Catalyst aqueous soln. (mL/min)	Equiv.NaOH (mmol NaOH/mmol substrate)	Equiv. catalyst (mmol cat./mmol substrate)	Residence Time (min)	Product (% area LCMS)	Reagent (% area LCMS)	Byproduct (% area LCMS)
0,4	100	0,7	1	0,2	26	0,73	10	71,59	25,37	3,04
0,2	100	0,7	1	0,2	26	0,36	10	72,26	27,94	0
0,1	100	0,7	1	0,2	26	0,18	10	65,75	34,25	0
0,05	100	0,7	1	0,2	26	0,09	10	40,10	59,90	0
0,4	120	0,7	1	0,2	26	0,73	10	83,64	12,17	4,19
0,2	120	0,7	1	0,2	26	0,36	10	87,57	8,52	3,91
0,1	120	0,7	1	0,2	26	0,18	10	83,21	13,53	3,26
0,05	120	0,7	1	0,2	26	0,09	10	70,66	29,34	0
0,4	140	0,7	1	0,2	26	0,73	10	83,61	8,66	7,73
0,2	140	0,7	1	0,2	26	0,36	10	90,52	5,01	4,47
0,1	140	0,7	1	0,2	26	0,18	10	80,52	14,3	5,18
0,05	140	0,7	1	0,2	26	0,09	10	66,34	29,39	4,27
0,4	160	0,7	1	0,2	26	0,73	10	87,63	6,52	5,85
0,2	160	0,7	1	0,2	26	0,36	10	87,41	7,93	4,66
0,1	160	0,7	1	0,2	26	0,18	10	77,01	19,12	3,87
0,05	160	0,7	1	0,2	26	0,09	10	63,03	33,88	3,09

d) 20% NaOH

Catalyst conc. (M)	T (°C)	NaOH aq. (mL/min)	Substrate in toluene (mL/min)	Catalyst aqueous solution (mL/min)	Equiv. NaOH (mmol NaOH/mmol substrate)	Equiv. catalyst (mmol catalyst/mmol substrate)	Residence Time (min)	Product (% area LCMS)	Reagent (% area LCMS)	Byproduct (% area LCMS)
0,1	120	0,284	1,42	0,284	10	0,18	10	75,10	24,90	0
0,1	120	0,57	2,85	0,57	10	0,18	5	56,68	43,32	0
0,1	120	0,4	2	0,4	10	0,18	7	70,93	26,19	2,88
0,1	130	0,4	2	0,4	10	0,18	7	86,96	13,04	0
0,1	120	0,284	1,42	0,284	10	0,18	10	78,38	17,07	4,55
0,1	130	0,284	1,42	0,284	10	0,18	10	81,34	11,47	7,19
0,1	120	0,284	1,42	0,284	10	0,18	10	68,92	31,08	0
0,1	130	0,284	1,42	0,284	10	0,18	10	71,47	25,43	3,1
0,1	180	0,284	1,42	0,284	10	0,18	10	71,03	26,09	2,88
0,1	120	0,14	0,7	0,14	10	0,18	20	80,08	19,92	0

0,1	120	0,12	0,42	0,12	15	0,26	30	93,24	6,76	0

A proof-of-concept of the scale-up process was validated during 7.5 hours run in a 1/16'' O.D. stainless steel tubular reactor for product 2e. An average value of 91% area product by LCMS was consistently and steadily observed during the course of the reaction as shown in SI Figure 4.

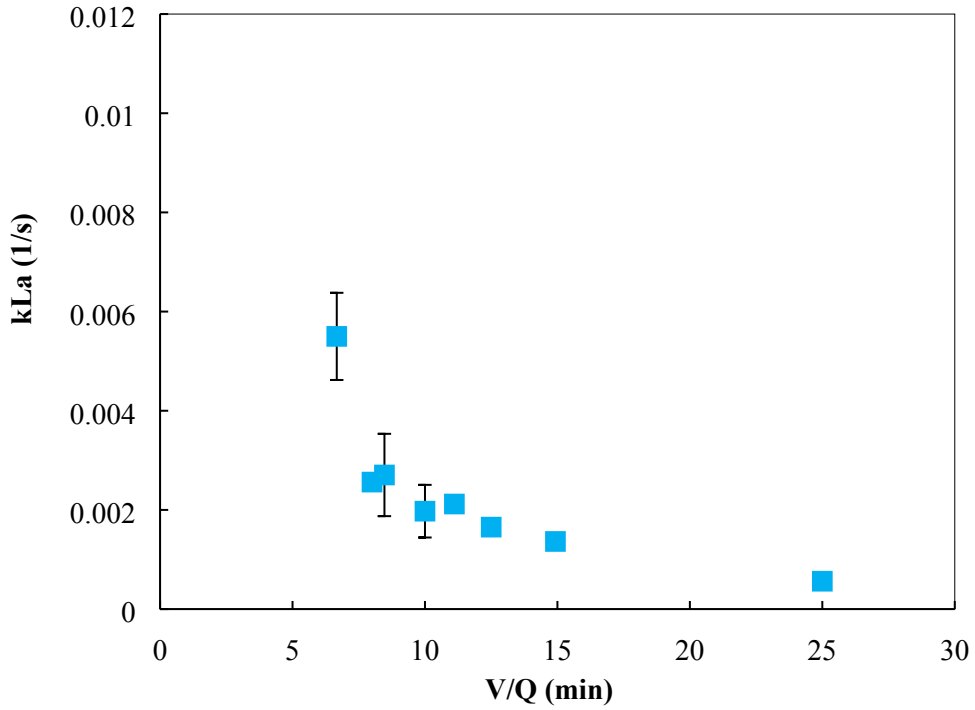


SI Figure 4: Evolution of product area percentage by LCMS in tubular reactor during 7.5 hours run

6. Mass transfer experiments in coil reactor

A model system water/toluene was used to experimentally analyze overall mass transfer coefficients in a 20 ml stainless steel coil reactor of 1/16'' O.D. at 25 ºC. The flowrate ratio of water:toluene was fixed to 1:1 and the residence time changed in the range 6.7 – 25 min. The experimental setup and procedure was similar to a flow experiment for PTC in flow described in the manuscript, but using only solvents. In this case, three samples at steady state were collected and analyzed by LCMS to obtain the amount of toluene dissolved in water. The LCMS area count was previously correlated with standard solutions of known toluene concentration. Overall mass transfer coefficient estimated as $k_La = Ln((C_e-C_i)/(C_e-C_o))/tau$, where Ce is the equilibrium concentration of toluene in water, C_i the toluene concentration in water at the inlet, C_o the outlet concentration of toluene in water at the outlet, and *tau* the residence time. The equilibrium

concentration for toluene in water at ambient temperature is 0.00053 g/ml (experimentally measured and in accordance with literature values from NIST [3]). There results of the overall mass transfer coefficients evolution with the residence time are shown in SI Figure 5. Shorter residence times enable higher mass transfer coefficents.



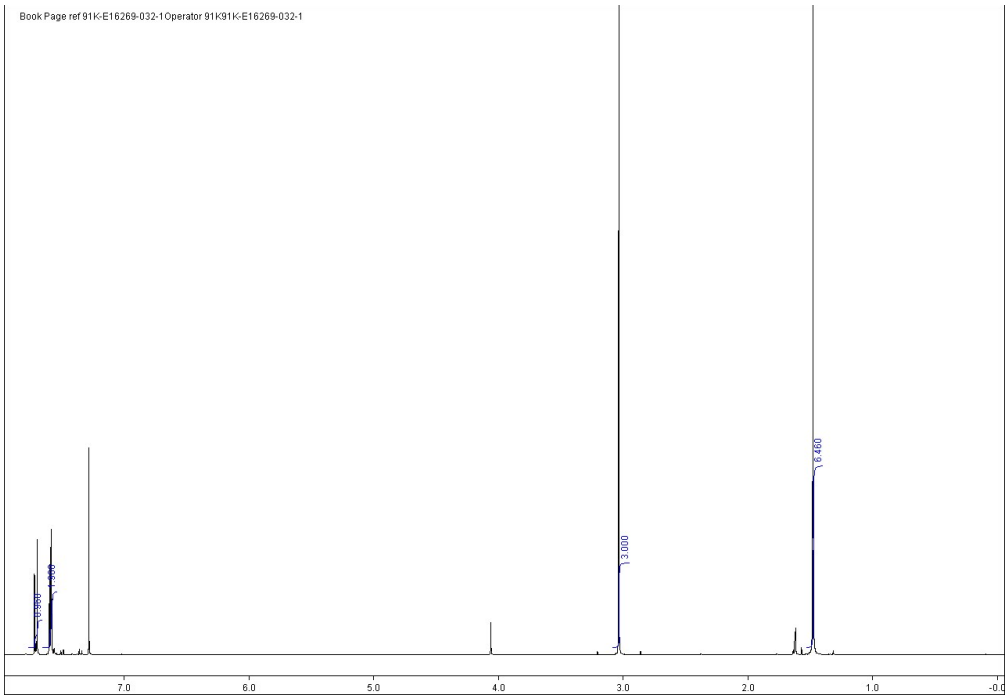
SI Figure 5: Overall mass transfer coefficients in 20 ml SS 1/16'' O.D. coil reactor for toluene/water at 25 °C and 1:1 flowrate ratio

7. NMR spectra for selected compounds

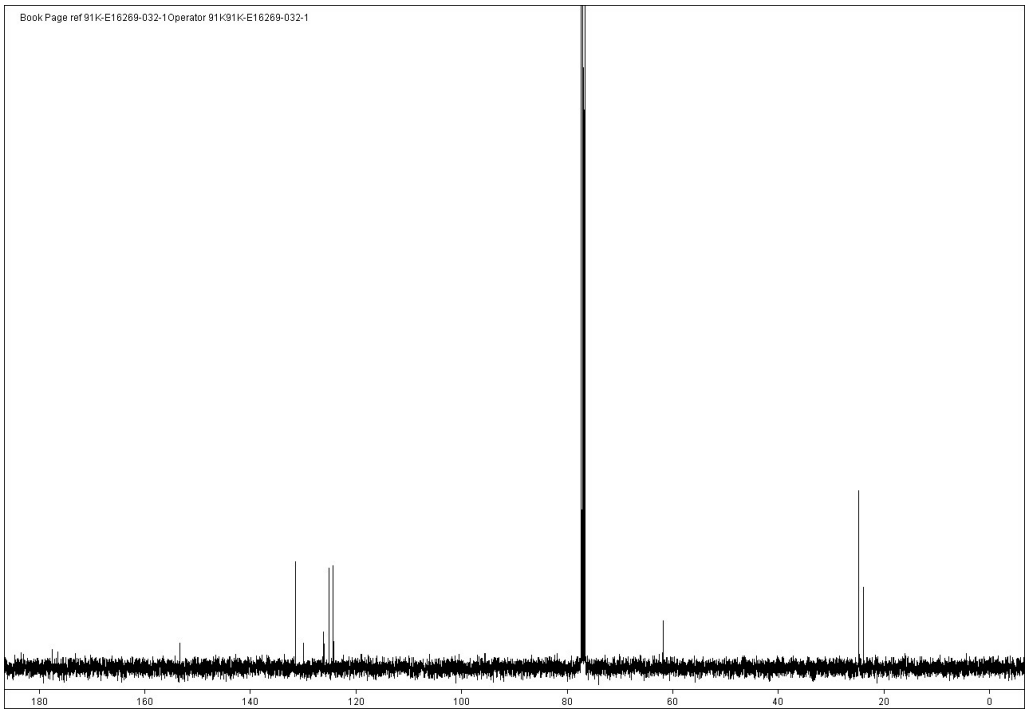
The NMR spectra corresponding to compounds selected for scale-up (2a and 2e) are presented here.

2a: ^{1}H NMR (400.15 MHz, CDCl3): 7.70 (d, J= 9Hz, 1H), 7.60-7.57 (m, 2H), 3.03 (s, 3H), 1.47 (s, 6H).

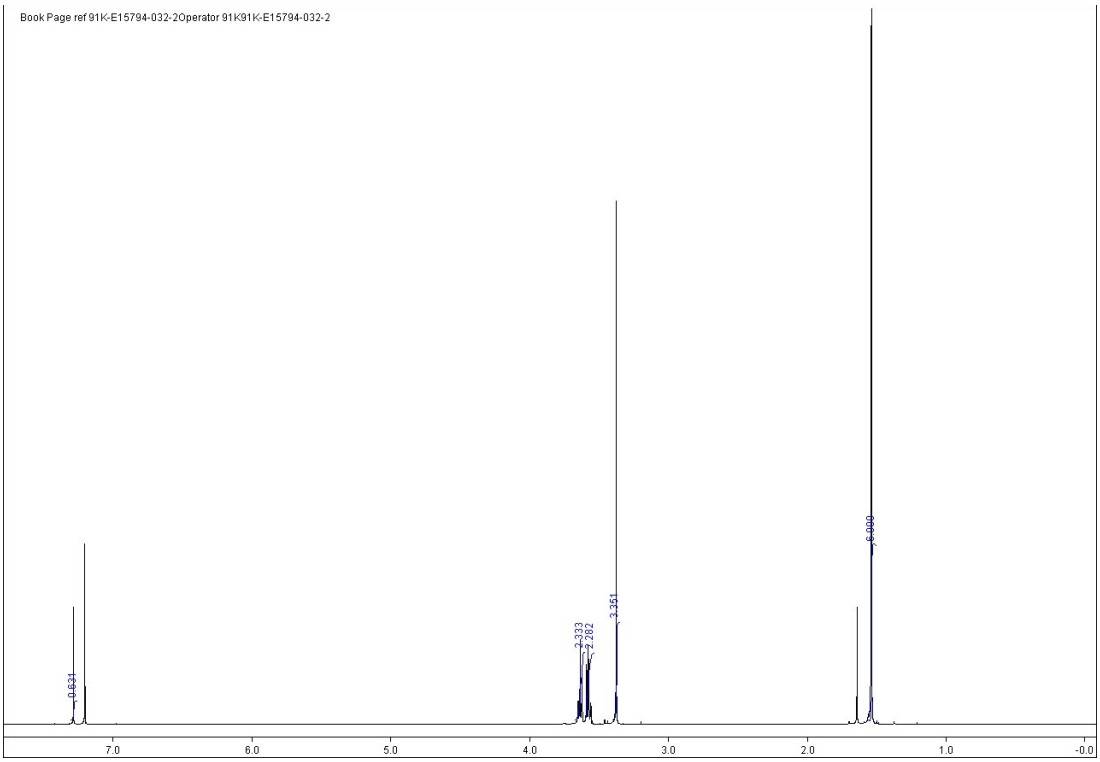
[3] IUPAC-NIST solubility database. Solubility data for toluene: https://srdata.nist.gov/solubility/IUPAC/SDS-37/SDS-37-pages_369.pdf (accessed on 23/10/2018)



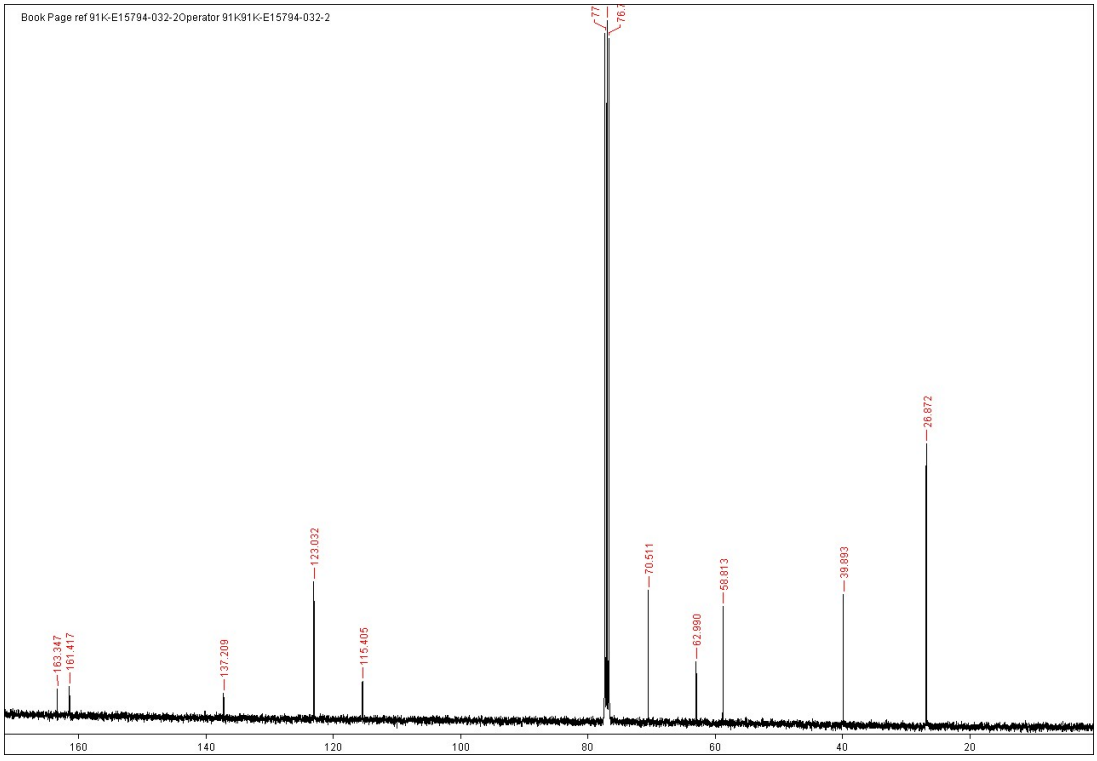
2a: ^{13}C NMR (300.16 MHz, CDCl3): 177.4, 153.2, 131.4, 129.9, 126.1, 125.1, 124.2, 61.8, 24.8, 23.9.



2e: ^{1}H NMR (400.21 MHz, CDCl3): 7.28 (s, 1H), 3.65-3.62 (m, 2H), 3.59-3.56 (m, 2H), 3.38 (s, 3H), 1.54 (s, 6H).



2e: ^{13}C NMR (100.63 MHz, CDCl3): 163.3, 161.4, 137.2, 123.0, 115.4, 70.5, 62.9, 58.8, 39.8, 26.8



Appendix A: Source code for software

Form I: Home Page

Button	Input	Output
Runfillempty	Click	Open Setup Page

HomePage.cs

```
using System;
using System.Windows.Forms;

namespace Peristaltic_General
{
    public partial class HomePage : Form
    {
        public HomePage()
        {   InitializeComponent();        }

        private void buttonRunFillEmpty_Click(object sender, EventArgs e)
        {
            SetupPage FillEmptyForm = new SetupPage();
            FillEmptyForm.Show();
        }
    }
}
```

Form II: Setup Page

Variable	Type	Description
nPumps	public static int	Number of pumps

Button	Input	Output
Confirm	nPumps	Main Panel (Form) or error message

SetupPage.cs

```
using System;
using System.Windows.Forms;

namespace Peristaltic_General
{
    public partial class ConnectionPort : Form
    {
        public static int nPumps;

        public ConnectionPort()
        {
            InitializeComponent();
        }

private void buttonConfirm_Click(object sender, EventArgs e)
        {
            try
            {
                nPumps = Convert.ToInt16(SelectNumberOfPumps.Text);
                Form NewExperiment = new
Main(int.Parse(SelectNumberOfPumps.Text));
                NewExperiment.Show();
            }
            catch (Exception)
            {
                MessageBox.Show("Error while loading form");
            }
        }

        private void ConnectionPort_FormClosing(object sender,
FormClosingEventArgs e)
        {
            e.Cancel = this.closeSystem();
        }

        public Boolean closeSystem()
        {
            try
            {
                DialogResult res;
                res = MessageBox.Show("Do you really want close the
system?", "Close System", MessageBoxButtons.OKCancel,
MessageBoxIcon.Question);

                if (serialPort1.IsOpen == true)
                {
                    serialPort1.Close();
                }
                if (res != DialogResult.OK)
                    return true;
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error when trying to close the system");
            }
            return false;
        }

    }
}
```

Form III: Main Panel

```
using System;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Windows.Forms;
using Peristaltic_General.Controls;
using System.IO.Ports;
using System.Diagnostics;
using System.Threading;
using System.Net.Mail;
using System.Net.Mime;
using System.Drawing.Imaging;
using System.Runtime.Remoting.Messaging;

namespace Peristaltic_General
```

```
{
    public partial class Main : Form
    {
        public int nBombas = 1;
        public SerialPort myport;
        System.Windows.Forms.Control f =
System.Windows.Forms.Application.OpenForms["ctlPump"];

        public TimerCallback TimerCallbackTotalTime = null; //Callback Timer for
Total Time counting
        public System.Threading.Timer TimerTotalTime = null; //Timer for Total
Time Counting
        public Stopwatch StopWatchrunTime = new Stopwatch();//Elapsed time since
the method started
```

```
        public TimeSpan tsTotalTime;
        public TimeSpan tsTotalTime2;

        public static int cycleNumber = 0;
        public static double RunTimePerCycle = 0;
        public static double TotalRunTime = 0;
        public int TotalNumberCycles = 0;
        public int contar = 0;
        public int maxValueTime;

        public int[] accumTime;
        public string[] StatusCheck;
        public int[] PumpStatusResponse;
        public string[] CommunicationStatusResponse;

        public bool evalTest = true;
        public int[] CheckStatusResponse;
        public string[] StringReadLoadData;

        public ctlPump bomba;
        public ctlPump[] CtlArray;
        public static SerialPort BalanceSerialPort;
        public static SerialPort OmegaTSerialPort;

        public string respuestaMyPort = " ";

        public double MassStart;
        public double MassEnd;
        public double MassDifference; // mass collected per cycle
        public double MassNow;
        public bool MassCheck = true;
        public string MassStringTrim;
        public string MassString;

        public static double Tmeasured;
        public static double TPlot;
        public static double timeXaxis;
        public static double MassPlot;

        public static int readingFrequency = 5;

        public GraphForm ChartTprofileFormCurrentCycle;
        public GraphForm ChartTprofileForm;
        public GraphForm ChartTprofileFormAll;
        public GraphForm testForm;
        public GraphForm ChartBalance;


        public string FolderName;
        public int counterError;
        public int counterEval;
        public int counterEvalMass;

        public string BalanceResponse0 = "";
        public string BalanceResponse = "";

        public System.Timers.Timer WaitUntilTrueTimer;
        public bool _waitUntilTrue = false;
        public string BalanceReadingTrim;

        // OmegaTserialPort

        internal delegate void SerialDataReceivedEventHandlerDelegate(object
sender, SerialDataReceivedEventArgs e);
        delegate void SetTextCallbackOmegaT(string textOmegaT);
        string InputDataOmegaT = String.Empty;
        internal delegate Boolean WriteToComPortDelegateOmegaT(string
textToWriteOmegaT);
        internal static WriteToComPortDelegateOmegaT
WriteToComPortDelegateOmegaT1 = new
WriteToComPortDelegateOmegaT(WriteToComPortOmegaT);

        public IAsyncResult arOmegaT;
        public string TextToWriteOmegaT = "";
        public static string msgOmegaT;

        // Balance Serial Port

        internal delegate void
SerialDataReceivedEventHandlerDelegateBalance(object sender,
SerialDataReceivedEventArgs e);
        delegate void SetTextCallbackBalance(string textBalance);
        string InputDataBalance = String.Empty;
        internal delegate Boolean WriteToComPortDelegateBalance(string
textToWriteBalance);
        internal static WriteToComPortDelegateBalance
WriteToComPortDelegateBalance1 = new
WriteToComPortDelegateBalance(WriteToComPortBalance);


        public IAsyncResult arBalance;
        public string TextToWriteBalance = "";
        public static string msgBalance;

        public int temperatureWarning = 0;
        public bool[] evalPump;

        public Main(int npumps)
        {
            InitializeComponent();

            //variables
            richTextBox1.Text = "5";
            readingFrequency = Convert.ToInt16(richTextBox1.Text);

            label7.Text = "";
            label10.Text = "";
            label11.Text = "";
            label12.Text = "";
            label18.Text = "";
            label19.Text = "";
            label21.Text = "";
            label22.Text = "";
            label23.Text = "";
            label24.Text = "";
            label25.Text = "";
            label26.Text = "";
            // limits for mass and temperature
            MinMass.Text = "40";
            MaxMass.Text = "70";
            minT.Text = "40";
            maxT.Text = "70";

            temperatureWarning = 0;
            textBoxNcycles.Text = "65";  // number of cycles
            PumpCheckFrequency.Text = "1"; // frequency to check pumps
            FrequencyBalance.Text = "30"; // frequency for balance

            Time1input.Text = "25";
            Time2input.Text = "290";
            DelayAfterCycle.Text = "180";
            textBox2.Text = "_name_@email.com";
            WarningT.Text = "";

            StatusCheckOption.Text = "NO";
            BalanceCheckOption.Text = "NO";
            TempCheckOption.Text = "NO";

            nBombas = npumps;

            CtlArray = new ctlPump[nBombas];
            textBoxNcycles.Text = Convert.ToString(1);
            StatusCheck = new string[nBombas];
            PumpStatusResponse = new int[nBombas];
            CommunicationStatusResponse = new string[nBombas];

            evalPump = new bool[nBombas];

            if (nBombas != 0)
            {
                for (int i = 1; i <= nBombas; i++)
                {
                    ctlPump bomba1 = new ctlPump();
                    bomba1.Location = new Point(150 + (100 * (i - 1)), 125);
                    panel1.Controls.Add(bomba1);
                    bomba1.Visible = true;
                    bomba1.Name = i.ToString();
                    bomba1.Enabled = true;
                    bomba = bomba1;
                    bomba.Name = i.ToString();
                    CtlArray[i - 1] = bomba;
                    evalPump[i - 1] = true;
                    CtlArray[i - 1].pumpNumber.Text =
Convert.ToString(bomba.Name);
                }
            }
        }

        internal static void WriteCompletedBalance(IAsyncResult ar)
        {
            WriteToComPortDelegateBalance deleg;
            Boolean success;

            deleg =
(WriteToComPortDelegateBalance)((AsyncResult)ar).AsyncDelegate;
            msgBalance = (string)ar.AsyncState;
            success = WriteToComPortDelegateBalance1.EndInvoke(ar);
        }

        internal static Boolean WriteToComPortBalance(string TextToWrite)
        {
            Boolean success = false;
            if (BalanceSerialPort.IsOpen)
            {
                byte[] bytes = TextToWrite.Split(' ').Select(s =>
Convert.ToByte(s, 16)).ToArray();
                BalanceSerialPort.Write(bytes, 0, bytes.Length);
                success = true;
            }
            return success;
        }
        internal static void WriteCompletedOmegaT(IAsyncResult ar)
        {
            WriteToComPortDelegateOmegaT deleg;
            Boolean success;

            deleg =
(WriteToComPortDelegateOmegaT)((AsyncResult)ar).AsyncDelegate;
            msgOmegaT = (string)ar.AsyncState;
            success = WriteToComPortDelegateOmegaT1.EndInvoke(ar);
        }

        internal static Boolean WriteToComPortOmegaT(string TextToWrite)
        {
            Boolean success = false;
            if (OmegaTSerialPort.IsOpen)
            {
                byte[] bytes = TextToWrite.Split(' ').Select(s =>
Convert.ToByte(s, 16)).ToArray();
                OmegaTSerialPort.Write(bytes, 0, bytes.Length);
                success = true;
            }
            return success;
        }

        public void InitializeCmd_Click(object sender, EventArgs e)
        {
            try
            {
                WarningT.Text = "";
```

```
                for (int j = 1; j <= nBombas; j++)
                {
                    CtlArray[j - 1].TemperatureOutsideOfRange = false;
                }
                if (TempCheckOption.Text == "YES")
                {
                    if ((maxT.Text == "") || (COMportOmega.Text == " ") ||
(maxT.Text == " ") || (COMportOmega.Text == ""))
                    {
                        MessageBox.Show("Please insert valid time inputs");
                    }
                    if (button8.Text == "Open")
                    {
                        MessageBox.Show("Please open Omega T port");
                    }
                }
                if (BalanceCheckOption.Text == "YES")
                {
                    if ((Time1input.Text == "") || (Time1input.Text == " ") ||
(Time2input.Text == "") || (Time2input.Text == " ") || ((Time1input.Text ==
Time2input.Text)))
                    {
                        MessageBox.Show("Please insert valid time inputs");
                    }
                    if ((MinMass.Text == "") || (MinMass.Text == " "))
                    {
                        MessageBox.Show("Please insert mass threshold (g)");
                    }
                    if (button5.Text == "Open")
                    {
                        MessageBox.Show("Please open balance port");
                    }
                }
                if (nBombas != 0)
                {
                    for (int j = 1; j <= nBombas; j++)
                    {
                        CtlArray[j - 1].StatusChanged = 0;
                        CtlArray[j - 1].AskForPumpStatus(CtlArray[j - 1]);
                        PumpStatusResponse[j - 1] = RetrievePumpStatus(j,
CtlArray[j - 1].PumpStatus.Text.Trim());

                        if (PumpStatusResponse[j - 1] == 1)
                        {
                            StatusExperiment.Text = "Pump " + Convert.ToString(j)
+ " waiting for instructions. Communication OK.";
                        }

                        if (PumpStatusResponse[j - 1] == 2)
                        {
                            StatusExperiment.Text = "Pump " + Convert.ToString(j)
+ " is instructed. Waiting to go. Communication OK.";
                        }
                        if (PumpStatusResponse[j - 1] == 3)
                        {
                            StatusExperiment.Text = "Pump " + Convert.ToString(j)
+ " is running. Communication OK.";
                        }
                        if ((PumpStatusResponse[j - 1] != 1) &
(PumpStatusResponse[j - 1] != 2) & (PumpStatusResponse[j - 1] != 3))
                        {
                            StatusExperiment.Text = "Error encountered. Pump
status error (different from 1, 2, 3).";
                        }
                    }
                }
                maxValueTime = this.DoCalculateMaxTime(); // calculates cycle
time (seconds)
                RunTimeCycleText.Text =
Convert.ToString(Convert.ToDouble(maxValueTime) / 60 +
Convert.ToDouble(DelayAfterCycle.Text) / 60);  // writes in textBox cycle time
(minutes)
                TotalRunTimeText.Text =
Convert.ToString(Convert.ToInt16(textBoxNcycles.Text) *
Convert.ToDouble(RunTimeCycleText.Text)); // writes total experiment time in
minutes

                RunTimePerCycle = Convert.ToDouble(RunTimeCycleText.Text);

                cycleNumber = 0;
                NumberCyclesCompletedText.Text = Convert.ToString("0");
                progressBar2.Value = 0;
                progressBar3.Value = 0;

                StopWatchrunTime.Restart();

                // stop pumps by default
                if (nBombas != 0)
                {
                    for (int j = 1; j <= nBombas; j++)
                    {
                        CtlArray[j - 1].StopPump(CtlArray[j - 1]);
                        StatusExperiment.Text = "Pump " + CtlArray[j - 1].Name +
" stopped";
                    }
                }
                MessageBox.Show("Initialization completed.");
            }
            catch (Exception)
            {
                MessageBox.Show("Error while initializing pumps");
            }
        }


        private int[] CheckAllPumpStatus()
        {
            try
            {
                StatusCheck = new string[nBombas];
                PumpStatusResponse = new int[nBombas];
                int[] stringResponseCheck = new int[nBombas];

                for (int i = 1; i <= nBombas; i++)
                {
                    CtlArray[i - 1].AskForPumpStatus(CtlArray[i - 1]);
                }

                for (int j = 1; j <= nBombas; j++)
                {
                    PumpStatusResponse[j - 1] = RetrievePumpStatus(j, CtlArray[j
- 1].PumpStatus.Text.Trim());
                }
                stringResponseCheck = PumpStatusResponse;
                return stringResponseCheck;
            }
            catch (Exception)
            {
                int[] stringResponseCheck = new int[nBombas];
                return stringResponseCheck;
            }
        }

        private int RetrievePumpStatus(int Npump, string Status)
        {
            try
            {
                string Message = "";
                int numberStatus = 0;
                string trimString = "";
                string trimString2 = "";
                string charString = "";
                string stringStatus = "";
                string CutStringStatus = "";
                int endString = Status.Length;
                charString = Status;


                for (int k = 0; k <= endString - 1; k++)
                {
                    if (k == 0)
                    {
                        trimString = "";
                        charString = Status;
                        trimString = charString.Remove(k + 1, endString - 1);

                        if (trimString == "P")
                        {
                            charString = Status;
                            stringStatus = charString.Remove(9, endString - 9);
                            CutStringStatus = stringStatus.Remove(0, 7);
                            k = endString - 1;
                        }
                    }
                    if (k != 0)
                    {

                        trimString = "";
                        charString = Status;

                        trimString = charString.Remove(0, k);
                        trimString2 = trimString.Remove(1, endString - k - 1);

                        if (trimString2 == "P")
                        {
                            charString = Status;
                            trimString = charString.Remove(0, k);
                            CutStringStatus = trimString.Remove(0, 7);

                            if (trimString.Length > 9)
                            {
                                trimString2 = trimString.Remove(9, endString - 9
- k);

                                CutStringStatus = trimString2.Remove(0, 7);
                            }
                            k = endString - 1;
                        }
                    }
                }


                if ((Convert.ToInt32(CutStringStatus) == 10) ||
(Convert.ToInt32(CutStringStatus) == 12) || (Convert.ToInt32(CutStringStatus) ==
15))
                {
                    Message = "Pump waiting for instructions. Communication OK.";
                    numberStatus = 1;
                }
                if ((Convert.ToInt32(CutStringStatus) == 20) ||
(Convert.ToInt32(CutStringStatus) == 22) || (Convert.ToInt32(CutStringStatus) ==
25))
                {
                    Message = "Pump instructed, waiting to go. Communication
OK.";
                    numberStatus = 2;
                }
                if ((Convert.ToInt32(CutStringStatus) == 30) ||
(Convert.ToInt32(CutStringStatus) == 32) || (Convert.ToInt32(CutStringStatus) ==
35))
                {
                    Message = "Pump running. Communication OK.";
                    numberStatus = 3;
                }
                if (CutStringStatus == "40")
                {
                    Message = "Pump stopped by local switch. Communication OK.";
                }
                if (CutStringStatus == "50")
                {
```

```
                    Message = "No motor feedback. Communication OK.";
                }
                if (CutStringStatus == "60")
                {
                    Message = "Overload. Communication OK.";
                }
                if (CutStringStatus == "70")
                {
                    Message = "Excessive motor feedback. Communication OK.";
                }

                else
                {
                    Message = "Error occurred while retrieving pump status " +
Message;
                }
                return numberStatus;
            }
            catch (Exception)
            {
                StatusExperiment.Text = "Error occurred while retrieving pump
status  ";
                return 0;
            }
        }

        private void CmdSTART()
        {
            try
            {
                WarningT.Text = "";

                for (int j = 1; j <= nBombas; j++)
                {
                    CtlArray[j - 1].TemperatureOutsideOfRange = false;
                }
                maxValueTime = this.DoCalculateMaxTime(); // calculates cycle
time (seconds)
                RunTimeCycleText.Text =
Convert.ToString(Convert.ToDouble(maxValueTime) / 60 +
Convert.ToDouble(DelayAfterCycle.Text) / 60);  // writes in textBox cycle time
(minutes)
                TotalRunTimeText.Text =
Convert.ToString(Convert.ToInt16(textBoxNcycles.Text) *
Convert.ToDouble(RunTimeCycleText.Text)); // writes total experiment time in
minutes
                RunTimePerCycle = Convert.ToDouble(RunTimeCycleText.Text);
                this.MeasureTemp();
                this.MeasureWeight();

                contar = 0;
                counterError = 0;

                StatusExperiment.Text = "Experiment Running";

                //start counting elapsed time
                TimerCallbackTotalTime = new TimerCallback(calcTotalTime);
                TimerTotalTime = new
System.Threading.Timer(TimerCallbackTotalTime, null, 0, 500);

                StopWatchrunTime.Start();
                StopWatchrunTime.Restart();

                progressBar2.Value = 0;
                progressBar3.Value = 0;

                // reset number of cycles
                NumberCyclesCompletedText.Text = Convert.ToString("0");
                cycleNumber = 0;
            }
            catch (Exception eTotal)
            {
                if (nBombas != 0)
                {
                    for (int j = 1; j <= nBombas; j++)
                    {
                        CtlArray[j - 1].StopPump(CtlArray[j - 1]);
                        StatusExperiment.Text = "Pump " + CtlArray[j - 1].Name +
" stopped";
                    }
                }
                MessageBox.Show("There was an error. All pumps stopped.");
            }
        }
        private void CmdRESTART()
        {
            try
            {
                WarningT.Text = "";
                for (int j = 1; j <= nBombas; j++)
                {
                    CtlArray[j - 1].StatusChanged = 0;
                }

                for (int j = 1; j <= nBombas; j++)
                {
                    CtlArray[j - 1].TemperatureOutsideOfRange = false;
                }
                StatusExperiment.Text = "Experiment Running";

                //start counting elapsed time
                TimerCallbackTotalTime = new TimerCallback(calcTotalTime);
                TimerTotalTime = new
System.Threading.Timer(TimerCallbackTotalTime, null, 0, 500);

                StopWatchrunTime.Start();
                StopWatchrunTime.Restart();
                temperatureWarning = 0;
            }
            catch (Exception eTotal)
            {
                if (nBombas != 0)
                {
                    for (int j = 1; j <= nBombas; j++)
                    {
                        CtlArray[j - 1].StopPump(CtlArray[j - 1]);
                        StatusExperiment.Text = "Pump " + CtlArray[j - 1].Name +
" stopped";
                    }
                }
                MessageBox.Show("There was an error. All pumps stopped.");
            }
        }
        private void StartExperimentCmd_Click(object sender, EventArgs e)
        {
            for (int j = 1; j <= nBombas; j++)
            {
                CtlArray[j - 1].StatusChanged = 0;
            }
            this.CmdSTART();
        }


        private void StopExperimentCmd_Click(object sender, EventArgs e)
        {
            try
            {
                if (nBombas != 0)
                {
                    for (int j = 1; j <= nBombas; j++)
                    {
                        CtlArray[j - 1].StopPump(CtlArray[j - 1]);
                        StatusExperiment.Text = "Pump " + CtlArray[j - 1].Name +
" stopped";
                    }
                    StopWatchrunTime.Start();
                    StopWatchrunTime.Restart();
                    StopWatchrunTime.Stop();
                    TimerTotalTime.Dispose();
                }
            }
            catch (Exception)
            {
                MessageBox.Show("Error while trying to stop experiment");
            }
        }


        private void Pause_Click(object sender, EventArgs e)
        {
            try
            {
                if (nBombas != 0)
                {
                    for (int j = 1; j <= nBombas; j++)
                    {
                        CtlArray[j - 1].StopPump(CtlArray[j - 1]);
                        StatusExperiment.Text = "Pump " + CtlArray[j - 1].Name +
" stopped";
                    }
                    StopWatchrunTime.Stop();
                }
            }
            catch (Exception)
            {
                MessageBox.Show("Error while trying to stop experiment");
            }
        }

        public void MeasureWeight()
        {
            try
            {
                this.balanceReading.Text = "";
                TextToWriteBalance = "49 50 0d 0a"; // weight
                msgBalance = DateTime.Now.ToString();
                arBalance =
WriteToComPortDelegateBalance1.BeginInvoke(TextToWriteBalance, new
AsyncCallback(WriteCompletedBalance), msgBalance);
            }
            catch
            {
                StatusExperiment.Text = "Balance warning/error encountered";
            }
        }

        public void TareWeight()
        {
            try
            {
                this.balanceReading.Text = "";
                TextToWriteBalance = "54 0d 0a";// tare
                msgBalance = DateTime.Now.ToString();
                arBalance =
WriteToComPortDelegateBalance1.BeginInvoke(TextToWriteBalance, new
AsyncCallback(WriteCompletedBalance), msgBalance);
                MeasureWeight();
            }
            catch
            {
                StatusExperiment.Text = "Barance Error";
            }
        }

        public void WriteTempToFile(int cumTime)
        {
            string pathString = System.IO.Path.Combine(@FolderName,
"FillEmptyDocuments");
            System.IO.Directory.CreateDirectory(pathString);
            this.WriteToFile(pathString, "TemperatureProfile",
```

```
Convert.ToString(cumTime) + "  " + Tnow.Text);
        }

        public void WriteMassToFile(int cumTime)
        {
            string pathString = System.IO.Path.Combine(@FolderName,
"FillEmptyDocuments");
            System.IO.Directory.CreateDirectory(pathString);
            this.WriteToFile(pathString, "MassBalance", Convert.ToString(cumTime)
+ "  " + ActualMass.Text);
        }

        public void WriteCommandToFile(int cumTime, string strInput)
        {
            string pathString = System.IO.Path.Combine(@FolderName,
"FillEmptyDocuments");
            System.IO.Directory.CreateDirectory(pathString);
            this.WriteToFile(pathString, "Log", Convert.ToString(cumTime) + ":  "
+ strInput);
        }
        public void MeasureTemp()
        {
            try
            {
                this.OmegaTreading.Text = "";
                TextToWriteOmegaT = "2a 47 31 31 30 0d";
                // measure temp.
                msgOmegaT = DateTime.Now.ToString();
                arOmegaT =
WriteToComPortDelegateOmegaT1.BeginInvoke(TextToWriteOmegaT, new
AsyncCallback(WriteCompletedOmegaT), msgOmegaT);
            }
            catch (Exception e3)
            {
                StatusExperiment.Text = " Error reading temperature" +
Convert.ToString(e3);
            }
        }

        private void EmergencyShutdown(string reasonForShutdown)
        {
            WarningT.Text = reasonForShutdown;
            this.SendEmailToUser(reasonForShutdown + " Cycles completed: " +
NumberCyclesCompletedText.Text);
            StopWatchrunTime.Stop();
            //StopWatchrunTime.Restart();

            for (int j = 1; j <= nBombas; j++)
            {
                CtlArray[j - 1].StopPump(CtlArray[j - 1]);
                StatusExperiment.Text = "Pump " + CtlArray[j - 1].Name + "
stopped";
            }
            TimerTotalTime.Dispose();
            NumberCyclesCompletedText.Text = textBoxNcycles.Text;
            return;
        }

        private void Shutdown()
        {

            WarningT.Text = "Experiment shutdown in progress..." + " Cycles
completed: " + NumberCyclesCompletedText.Text;
            this.SendEmailToUser("Experiment shutdown in progress..." + " Cycles
completed: " + NumberCyclesCompletedText.Text);
            StopWatchrunTime.Stop();
            //StopWatchrunTime.Restart();

            for (int j = 1; j <= nBombas; j++)
            {
                CtlArray[j - 1].StopPump(CtlArray[j - 1]);
                StatusExperiment.Text = "Pump " + CtlArray[j - 1].Name + "
stopped";
            }
            TimerTotalTime.Dispose();
            NumberCyclesCompletedText.Text = textBoxNcycles.Text;
            return;
        }

        private void GoToNextCycle()
        {
            WarningT.Text = "Starting next cycle..." + " Cycles completed: " +
NumberCyclesCompletedText.Text;
            StopWatchrunTime.Stop();
            TimerTotalTime.Dispose();
            this.SendEmailToUser("Completed cycle number " +
NumberCyclesCompletedText.Text + "\r" + "Mass (g) at time 1: " + SOCmass.Text +
"\r" + "Mass(g) at time 2: " + EOCmass.Text + "Collected mass (g): " +
MassCollectedPerCycle.Text + "\r\n" + "Temperature: " + Tnow.Text + " degC");

            System.Threading.Thread.Sleep(250);

            for (int j = 1; j <= nBombas; j++)
            {
                CtlArray[j - 1].StopPump(CtlArray[j - 1]);
                StatusExperiment.Text = "Pump " + CtlArray[j - 1].Name + "
stopped";
            }
            //StopWatchrunTime.Restart();
            System.Threading.Thread.Sleep(250);
            cycleNumber = cycleNumber + 1;
            NumberCyclesCompletedText.Text = Convert.ToString(cycleNumber);

            temperatureWarning = 0;

            for (int j = 1; j <= nBombas; j++)
            {
                CtlArray[j - 1].StatusChanged = 0;
            }
            // Restart clock counter
            this.CmdRESTART();
            return;
        }
        private void calcTotalTime(object state)
        {
            try
            {
                tsTotalTime = StopWatchrunTime.Elapsed;
                string elapsedTime = String.Format("{0:00}:{1:00}:{2:00}",
                tsTotalTime.Hours, tsTotalTime.Minutes, tsTotalTime.Seconds);
                UpdateTime(elapsedTime);
                int cumulativeTime = 0;
                // Cumulative time per cycle in seconds:
                cumulativeTime = tsTotalTime.Hours * 3600 + tsTotalTime.Minutes *
60 + tsTotalTime.Seconds;

                if (Convert.ToInt32(NumberCyclesCompletedText.Text) >=
Convert.ToInt32(textBoxNcycles.Text))
                {
                    this.Shutdown();
                }

                if (Convert.ToInt32(NumberCyclesCompletedText.Text) <
Convert.ToInt32(textBoxNcycles.Text))
                {
                    readingFrequency = Convert.ToInt16(richTextBox1.Text);
                    maxValueTime = this.DoCalculateMaxTime(); // calculates cycle
time (seconds)
                    RunTimeCycleText.Text =
Convert.ToString(Convert.ToDouble(maxValueTime) / 60 +
Convert.ToDouble(DelayAfterCycle.Text) / 60);  // writes in textBox cycle time
(minutes)
                    TotalRunTimeText.Text =
Convert.ToString(Convert.ToInt16(textBoxNcycles.Text) *
Convert.ToDouble(RunTimeCycleText.Text)); // writes total experiment time in
minutes
                    RunTimePerCycle = Convert.ToDouble(RunTimeCycleText.Text);
                    this.DoUpdateProgressBars(cumulativeTime);

                    if (((balanceReading.Text).Trim() ==
"ES")&(BalanceCheckOption.Text == "YES"))
                    { this.MeasureWeight(); }

                    for (int k = 1; k < (maxValueTime +
Convert.ToInt16(DelayAfterCycle.Text)); k++)
                    {
                        // check all pump status 1's and 0's at specified
frequency
                        if ((StatusCheckOption.Text == "YES") & (cumulativeTime <
(maxValueTime + Convert.ToInt16(DelayAfterCycle.Text))) & (cumulativeTime == k *
Convert.ToInt16(PumpCheckFrequency.Text)))
                        {
                            this.CheckAllPumpStatus();
                        }

                 // measure mass at specified balance frequency
                        if ((BalanceCheckOption.Text == "YES") & (cumulativeTime
< (maxValueTime + Convert.ToInt16(DelayAfterCycle.Text))) & (cumulativeTime == k
* Convert.ToInt16(FrequencyBalance.Text)))
                        {
                            this.MeasureWeight();
                            this.WriteMassToFile(cumulativeTime);
                            timeXaxis = Convert.ToDouble(cumulativeTime) / 60;
                            MassPlot = Convert.ToDouble(ActualMass.Text);
                            //ChartBalance.plotMyChartBalance();
                        }
                        // measure temperature at specified temperature frequency
                        if ((TempCheckOption.Text == "YES") & (cumulativeTime <
(maxValueTime + Convert.ToInt16(DelayAfterCycle.Text))) & (cumulativeTime == k *
readingFrequency))
                        {
                            this.MeasureTemp();
                            this.WriteTempToFile(cumulativeTime);
                            timeXaxis = Convert.ToDouble(cumulativeTime) / 60;
                            TPlot = Convert.ToDouble(Tnow.Text);

                            //ChartTprofileFormAll.plotMyChart_all(); // plots
all consecutive cycles
                            //ChartTprofileForm.plotMyChart(); // plots all
combined cycles (time x axis only 1 cycle time)
                            //ChartTprofileFormCurrentCycle.plotMyChart();  //
plots only current cycle (resets for new cycle)

                            if (((Convert.ToDouble((Tnow.Text).Trim()) <
Convert.ToDouble(minT.Text.Trim())) || (Convert.ToDouble((Tnow.Text).Trim()) >
Convert.ToDouble(maxT.Text.Trim()))))
                            {
                                temperatureWarning = temperatureWarning + 1;
                                if (temperatureWarning > 6)
                                {
                                    for (int j = 1; j <= nBombas; j++)
                                    { CtlArray[j - 1].TemperatureOutsideOfRange =
true; }
                                    this.EmergencyShutdown("Temperature
warning... Shutdown in progress. Actual temperature: " + Tnow.Text + " T range
(degC):  " + minT.Text + " - " + maxT.Text + ".");
                                }
                            }
                        }
                    }

                    // if BALANCE CHECK enabled:
                    if ((BalanceCheckOption.Text == "YES") & (cumulativeTime ==
Convert.ToInt16(Time1input.Text)))
                    {
                        try
                        { SOCmass.Text = ActualMass.Text; }
                        catch (Exception)
                        { StatusExperiment.Text = "Error within second if loop
(Time == Time 1 measuring weight at time 1"; }
```

```
                    }

                    if ((BalanceCheckOption.Text == "YES") & (cumulativeTime ==
Convert.ToInt16(Time2input.Text)))
                    {
                        try
                        {
                            EOCmass.Text = ActualMass.Text;
                            MassDifference = Convert.ToDouble(EOCmass.Text) -
Convert.ToDouble(SOCmass.Text);
                            MassCollectedPerCycle.Text =
Convert.ToString(MassDifference);
                        }
                        catch (Exception)
                        { StatusExperiment.Text = "Error within if loop (Time ==
Time 2) measuring weight at time 2"; }
                    }


                    // always do:
                    for (int j = 1; j <= nBombas; j++)
                    {
                        // before pumps start all should be stopped
                        if ((cumulativeTime < CtlArray[j - 1].DelayTimeSec) &
(cumulativeTime != 0))
                        {
                            PumpStatusResponse[j - 1] = RetrievePumpStatus(j,
CtlArray[j - 1].PumpStatus.Text.Trim());
                            if (PumpStatusResponse[j - 1] != 1)
                            {
                                CtlArray[j - 1].StopPump(CtlArray[j - 1]);
                            }
                            StatusExperiment.Text = "Pump " + CtlArray[j -
1].Name + " stopped. Status:   " + Convert.ToString(PumpStatusResponse[j - 1]);
                        }

                        // START PUMPS @ delay time
                        if (cumulativeTime == CtlArray[j - 1].DelayTimeSec)
                        {
                            CtlArray[j - 1].RunPump(CtlArray[j - 1], CtlArray[j -
1].Pumpingdirection, 99999, CtlArray[j -
1].ConvertFlowRateToRotorSpeed(CtlArray[j - 1]));

                            // takes status from PumpStatus, not from
Incoming.Text  that is refreshed
                            PumpStatusResponse[j - 1] = RetrievePumpStatus(j,
CtlArray[j - 1].PumpStatus.Text.Trim());
                            // send command again if not running
                            if (PumpStatusResponse[j - 1] != 3)
                            {
                                CtlArray[j - 1].RunPump(CtlArray[j - 1],
CtlArray[j - 1].Pumpingdirection, 99999, CtlArray[j -
1].ConvertFlowRateToRotorSpeed(CtlArray[j - 1]));
                            }
                            StatusExperiment.Text = "Pump " + CtlArray[j -
1].Name + " running. Status:   " + Convert.ToString(PumpStatusResponse[j - 1]);
                        }

                        // Check status while pumps are running
                        if ((cumulativeTime >= CtlArray[j - 1].DelayTimeSec) &
(cumulativeTime <= (CtlArray[j - 1].PumpingTimeSec + CtlArray[j -
1].DelayTimeSec)))
                        {
                            PumpStatusResponse[j - 1] = RetrievePumpStatus(j,
CtlArray[j - 1].PumpStatus.Text.Trim());
                            if (PumpStatusResponse[j - 1] != 3)
                            {
                                CtlArray[j - 1].RunPump(CtlArray[j - 1],
CtlArray[j - 1].Pumpingdirection, 99999, CtlArray[j -
1].ConvertFlowRateToRotorSpeed(CtlArray[j - 1]));
                            }
                            StatusExperiment.Text = "Pump " + CtlArray[j -
1].Name + " running. Status:   " + Convert.ToString(PumpStatusResponse[j - 1]);
                        }

                        // STOP PUMPS @ delay + pumping time
                        if (cumulativeTime == (CtlArray[j - 1].PumpingTimeSec +
CtlArray[j - 1].DelayTimeSec))
                        {
                            CtlArray[j - 1].StopPump(CtlArray[j - 1]);
                            PumpStatusResponse[j - 1] = RetrievePumpStatus(j,
CtlArray[j - 1].PumpStatus.Text.Trim());
                            if (PumpStatusResponse[j - 1] != 1)
                            {
                                CtlArray[j - 1].StopPump(CtlArray[j - 1]);
                            }
                            StatusExperiment.Text = "Pump " + CtlArray[j -
1].Name + " stopped. Status:   " + Convert.ToString(PumpStatusResponse[j - 1]);
                        }
                        // ensure that pumps are stopped at time greater than
(pumping time+delay)
                        if (cumulativeTime > (CtlArray[j - 1].PumpingTimeSec +
CtlArray[j - 1].DelayTimeSec))
                        {
                            PumpStatusResponse[j - 1] = RetrievePumpStatus(j,
CtlArray[j - 1].PumpStatus.Text.Trim());
                            if (PumpStatusResponse[j - 1] != 1)
                            {
                                CtlArray[j - 1].StopPump(CtlArray[j - 1]);
                            }
                            StatusExperiment.Text = "Pump " + CtlArray[j -
1].Name + " stopped. Status:   " + Convert.ToString(PumpStatusResponse[j - 1]);
                        }
                        // 2 seconds before cycle is completed, check pump status
                        if ((cumulativeTime == (maxValueTime +
Convert.ToInt16(DelayAfterCycle.Text) - 2)) & (StatusCheckOption.Text == "YES"))
                        {
                            // send warning message if pump did not change status
twice (ON-OFF)
                            if ((CtlArray[j - 1].StatusChanged % 2) != 0)
                            {
                                this.SendEmailToUser("Warning. Pump " +
CtlArray[j - 1].pumpNumber.Text + " did not change status twice. No actions
taken.");
                            }

                            try
                            {
                                evalPump[j - 1] = CtlArray[j -
1].EvaluateStatus(CtlArray[j - 1]);
                                if (evalPump[j - 1] == true)
                                {
                                    StatusExperiment.Text = "Pump " +
Convert.ToString(j) + " passed status check";
                                    evalTest = true;
                                }
                                if (evalPump[j - 1] == false)
                                {
                                    StatusExperiment.Text = "Eval Pump " +
Convert.ToString(j) + " is false in first attempt. Trying to reconnect... ";

                                    CtlArray[j - 1].ConnectPump(CtlArray[j - 1]);
// Send command 05 to pumps   ---- P?0
                                    System.Threading.Thread.Sleep(50);
                                    CtlArray[j - 1].SetPumpNumber(CtlArray[j -
1]);  // Set Pump number
                                    System.Threading.Thread.Sleep(50);
                                    CtlArray[j - 1].SetPumpToRemote(CtlArray[j -
1]); // Set pump to remote
                                    System.Threading.Thread.Sleep(50);
                                    CtlArray[j - 1].AskForPumpStatus(CtlArray[j -
1]);  // Send command to ask for pump status
                                    evalPump[j - 1] = CtlArray[j -
1].EvaluateStatus(CtlArray[j - 1]); // retrieve true/false status

                                    if (evalPump[j - 1] == false)
                                    {
                                        evalTest = false;
                                        break; // breaks (for) loop - stopps
checking other pumps when one is offline
                                    }
                                    if (evalPump[j - 1] == true)
                                    {
                                        evalTest = true;
                                    }
                                }
                                CtlArray[j - 1].label1.Text =
Convert.ToString(evalPump[j - 1]);
                            }
                            catch (Exception)
                            { StatusExperiment.Text = "Error within status check
assignment";
                            }
                        }
                    }
                    // MASS CHECK
                    if ((cumulativeTime == (maxValueTime +
Convert.ToInt16(DelayAfterCycle.Text) - 1)) & (BalanceCheckOption.Text == "YES"))
                    {
                        try
                        {
                            MassDifference = Convert.ToDouble(EOCmass.Text) -
Convert.ToDouble(SOCmass.Text);
                            MassCollectedPerCycle.Text =
Convert.ToString(MassDifference);
                            if ((MassDifference > Convert.ToDouble(MinMass.Text))
& (MassDifference < Convert.ToDouble(MaxMass.Text)))
                            {
                                MassCheck = true;
                            }
                            if ((MassDifference <=
Convert.ToDouble(MinMass.Text)) || (MassDifference >=
Convert.ToDouble(MaxMass.Text)))
                            {
                                MassCheck = false;

                                EOCmass.Text = ActualMass.Text;
                                MassDifference = Convert.ToDouble(EOCmass.Text) -
Convert.ToDouble(SOCmass.Text);
                                MassCollectedPerCycle.Text =
Convert.ToString(MassDifference);

                                if ((MassDifference >
Convert.ToDouble(MinMass.Text)) & (MassDifference <
Convert.ToDouble(MaxMass.Text)))
                                {
                                    MassCheck = true;
                                }
                                if ((MassDifference <=
Convert.ToDouble(MinMass.Text)) || (MassDifference >=
Convert.ToDouble(MaxMass.Text)))
                                {
                                    MassCheck = false;
                                }
                            }
                        }
                        catch (Exception)
                        {
                            StatusExperiment.Text = "Error within mass check
assignment";
                        }
                    }

                    // CHECK AT END OF CYCLE

                    /// if mass balance check is true then MassCheck==true
                    /// if status check is true then set evalTest == true

                    try {
                        if (cumulativeTime == (maxValueTime +
```

```
Convert.ToInt16(DelayAfterCycle.Text)))
                        {
                            if (((evalTest == true) & (StatusCheckOption.Text ==
"YES")) & ((MassCheck == false) & (BalanceCheckOption.Text == "YES")))
                            {
                                this.EmergencyShutdown("Experiment aborted.
Balance check failed. " + "Mass Collected (g): " + MassCollectedPerCycle.Text + "
Mass 1 (g): " + SOCmass.Text + " Mass 2 (g):" + EOCmass.Text);
                            }
                            if (((evalTest == false) & (StatusCheckOption.Text ==
"YES")) & ((MassCheck == false) & (BalanceCheckOption.Text == "YES")))
                            {
                                this.EmergencyShutdown("Experiment aborted. Pump
and balance check failed. " + "Mass Collected (g): " + MassCollectedPerCycle.Text
+ "  Mass 1 (g): " + SOCmass.Text + " Mass 2 (g):" + EOCmass.Text);
                            }
                            if (((evalTest == false) & (StatusCheckOption.Text ==
"YES")) & ((MassCheck == true) & (BalanceCheckOption.Text == "YES")))
                            {
                                this.EmergencyShutdown("Experiment aborted. Pump
check failed. " + "Mass Collected (g): " + MassCollectedPerCycle.Text + "  Mass 1
(g): " + SOCmass.Text + " Mass 2 (g):" + EOCmass.Text);
                            }
                            if (((evalTest == true) & (StatusCheckOption.Text ==
"YES")) & ((MassCheck == true) & (BalanceCheckOption.Text == "YES")))
                            {
                                this.GoToNextCycle();
                            }
                            if ((evalTest == false) & (StatusCheckOption.Text ==
"YES") & (BalanceCheckOption.Text == "NO"))
                            {
                                this.EmergencyShutdown("Experiment aborted. Pump
check failed. " + "Mass Collected (g): " + MassCollectedPerCycle.Text + "  Mass 1
(g): " + SOCmass.Text + " Mass 2 (g):" + EOCmass.Text);
                            }
                            if ((evalTest == true) & (StatusCheckOption.Text ==
"YES") & (BalanceCheckOption.Text == "NO"))
                            {
                                this.GoToNextCycle();
                            }
                            if ((StatusCheckOption.Text == "NO") &
(BalanceCheckOption.Text == "NO"))
                            {
                                this.GoToNextCycle();
                            }
                        }
                    }
                    catch (Exception)
                    { }
                }
            catch (Exception)
            {
                StatusExperiment.Text = "Error within calcTotalTime";
            }
        }

        private void WriteToFile(string folderLocation, string filename, string
msg)
        {
            try
            {
                using (System.IO.StreamWriter file = new
System.IO.StreamWriter(folderLocation + "\\" + filename + ".txt", true))
                {
                    file.WriteLine(msg);
                }
            }
            catch
            {
                StatusExperiment.Text = "error while writing to text file";
            }
        }
        private void SendEmailToUser(string MessageToSend)
        {
            try
            {
                SmtpClient mySmtpClient = new SmtpClient("xxx.yyy.com");
                mySmtpClient.UseDefaultCredentials = false;
                mySmtpClient.Port = 25;

                //mySmtpClient.EnableSsl = true;
                mySmtpClient.Timeout = 100000000;
                mySmtpClient.DeliveryMethod = SmtpDeliveryMethod.Network;

                // add from,to mailaddresses
                MailAddress from = new MailAddress("fill_empty@email.com", "Fill
Empty Experiment");
                MailAddress to = new MailAddress(textBox2.Text, "TestToName");
                MailMessage myMail = new System.Net.Mail.MailMessage(from, to);

                // set subject and encoding
                myMail.Subject = "Message from Fill Empty";
                myMail.SubjectEncoding = System.Text.Encoding.UTF8;

                // set body-message and encoding
                myMail.Body = MessageToSend;
                myMail.BodyEncoding = System.Text.Encoding.UTF8;
                // text or html
                myMail.IsBodyHtml = true;
                mySmtpClient.Send(myMail);
                myMail.Dispose();

                // send backup message
                MailAddress to2 = new MailAddress("_name_@email.com",
"TestToName");
                MailMessage myMail2 = new System.Net.Mail.MailMessage(from, to2);

                // set subject and encoding
                myMail2.Subject = "Message from Fill Empty";
                myMail2.SubjectEncoding = System.Text.Encoding.UTF8;
                // set body-message and encoding
                myMail2.Body = MessageToSend;
                myMail2.BodyEncoding = System.Text.Encoding.UTF8;
                // text or html
                myMail2.IsBodyHtml = true;
                mySmtpClient.Send(myMail2);
                myMail2.Dispose();
            }

            catch (Exception ex)
            {
                if (ex.InnerException != null)
                {
                    StatusExperiment.Text = Convert.ToString(ex.InnerException);
                }
            }
        }

        private void DoUpdateProgressBars(int timeInput)
        {
            try
            {
                double valueForProgressBar2 = 100 * Convert.ToDouble(timeInput) /
(Convert.ToDouble(maxValueTime) + Convert.ToDouble(DelayAfterCycle.Text));
                progressBar2.Value = Convert.ToInt32(valueForProgressBar2);

                double TotalTimeCyclesCompleted =
Convert.ToInt32(NumberCyclesCompletedText.Text) *
Convert.ToDouble(RunTimeCycleText.Text) * 60;
                double TotalRunTime = (Convert.ToInt32(textBoxNcycles.Text) *
Convert.ToDouble(RunTimeCycleText.Text)) * 60;

                double valueForProgressBar3 = 100 * (Convert.ToDouble(timeInput)
+ TotalTimeCyclesCompleted) / TotalRunTime;
                progressBar3.Value = Convert.ToInt32(valueForProgressBar3);
            }
            catch (Exception)
            {
                StatusExperiment.Text = "Error while updating progress bars";
            }
        }

        private delegate void delegateUpdatewithString(string s);
        private void UpdateTime(string newtime)
        {
            if (lbltotaltime1.InvokeRequired)
            {
                lbltotaltime1.Invoke(new delegateUpdatewithString(UpdateTime),
new object[] { newtime });
            }
            else
                lbltotaltime1.Text = newtime;
        }

        private int DoCalculateMaxTime()
        {
            try
            {
                accumTime = new int[nBombas];
                for (int j = 1; j <= nBombas; j++)
                {
                    accumTime[j - 1] = CtlArray[j - 1].PumpingTimeSec +
CtlArray[j - 1].DelayTimeSec;
                }
                int maxValue = accumTime.Max();
                return maxValue;
            }
            catch (Exception)
            {
                StatusExperiment.Text = "Error while calculating max time";
                return 0;
            }
        }

        private void Main_FormClosing(object sender, FormClosingEventArgs e)
        {
            e.Cancel = this.closeSystem();
        }


        public Boolean closeSystem()
        {
            try
            {
                DialogResult res;
                res = MessageBox.Show("Do you really want close the system?",
"Close System", MessageBoxButtons.OKCancel, MessageBoxIcon.Question);

                // close ports when closing Windows application form

                if ((OmegaTSerialPort.IsOpen == true)&(OmegaTSerialPort.PortName
!=""))
                {
                    OmegaTSerialPort.Close();
                }
                if ((BalanceSerialPort.IsOpen ==
true)&(BalanceSerialPort.PortName != ""))
                {
                    BalanceSerialPort.Close();
                }
                // scan all ports for pumps
                for (int j = 1; j <= nBombas; j++)
                {
                    if ((CtlArray[j - 1].myport.IsOpen == true) &(CtlArray[j -
1].myport.PortName != ""))
                    {
                        CtlArray[j - 1].myport.Close();
                    }
                }
```

```
                if (res != DialogResult.OK)
                    return true;
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error when trying to close the system");
            }
            return false;
        }

        private void textBoxNcycles_TextChanged(object sender, EventArgs e)
        {
            try
            {
                if ((textBoxNcycles.Text == " ") || (textBoxNcycles.Text == "")
|| (textBoxNcycles.Text == "0"))
                {
                    textBoxNcycles.Text = Convert.ToString(1);
                }
            }
            catch (Exception)
            {
                MessageBox.Show("Please insert valid number of cycles");
            }
        }

        private void button2_Click(object sender, EventArgs e)
        {
            try
            {
                for (int j = 1; j <= nBombas; j++)
                {
                    string rotor1hex =
ConvertRPMtoHex(Convert.ToString(CtlArray[j - 1].RPMInput));
                    CtlArray[j - 1].RunPump(CtlArray[j - 1], CtlArray[j -
1].Pumpingdirection, Convert.ToInt16(CtlArray[j - 1].RPMInput), rotor1hex);
                }
            }
            catch
            {
                MessageBox.Show("Error while calibrating pump");
            }
        }

        private string ConvertRPMtoHex(string RPMinputText)
        {
            try
            {
                string rotor1 = (Convert.ToInt16(RPMinputText)).ToString("D4");
                char[] characters1 = rotor1.ToCharArray();
                string rotor1hex = Convert.ToInt16(characters1[0]).ToString("x")
+ " " + Convert.ToInt16(characters1[1]).ToString("x") + " " +
Convert.ToInt16(characters1[2]).ToString("x") + " " +
Convert.ToInt16(characters1[3]).ToString("x");
                rotor1hex = rotor1hex + " 2e 30";
                return rotor1hex;
            }
            catch (Exception)
            {
                StatusExperiment.Text = "Error while converting RPM to
hexadecimal";
                return Convert.ToString(0);
            }
        }

        private void button1_Click(object sender, EventArgs e)
        {
            try
            {
                CheckStatusResponse = new int[nBombas];
                CheckStatusResponse = CheckAllPumpStatus();
            }
            catch (Exception)
            {
                StatusExperiment.Text = "Error while checking pump status";
            }
        }


        private void Continue_Click(object sender, EventArgs e)
        {
            try
            {
                StopWatchrunTime.Start();

                tsTotalTime2 = StopWatchrunTime.Elapsed;
                string elapsedTime2 = String.Format("{0:00}:{1:00}:{2:00}",
                tsTotalTime2.Hours, tsTotalTime2.Minutes, tsTotalTime2.Seconds);
                UpdateTime(elapsedTime2);
                int cumulativeTime2 = 0;

                // Cumulative time per cycle in seconds:
                cumulativeTime2 = tsTotalTime2.Hours * 3600 +
tsTotalTime2.Minutes * 60 + tsTotalTime2.Seconds;

                for (int j = 1; j <= nBombas; j++)
                {
                    if ((cumulativeTime2 >= CtlArray[j - 1].DelayTimeSec) &
(cumulativeTime2 <= (CtlArray[j - 1].PumpingTimeSec + CtlArray[j -
1].DelayTimeSec)))
                    {
                        CtlArray[j - 1].RunPump(CtlArray[j - 1], CtlArray[j -
1].Pumpingdirection, 99999, CtlArray[j -
1].ConvertFlowRateToRotorSpeed(CtlArray[j - 1]));
                        StatusExperiment.Text = "Pump " + CtlArray[j - 1].Name +
" running";
                    }
                }
            }
```

```
            catch (Exception)
            {
                MessageBox.Show("Error while trying to stop experiment");
            }
        }

        private void BalanceCheckOption_SelectedIndexChanged(object sender,
EventArgs e)
        {
            if (BalanceCheckOption.Text == "YES")
            {
                BalanceSerialPort = new SerialPort();
                getAvailablePorts(COMportBalance);
            }
        }

        void getAvailablePorts(ComboBox boxPorts)
        {
            string[] ports = SerialPort.GetPortNames();
            boxPorts.Items.AddRange(ports);
        }

        private void BalanceSerialPort_DataReceived(object sender,
SerialDataReceivedEventArgs e)
        {
            InputDataBalance = BalanceSerialPort.ReadExisting();
            if (InputDataBalance != String.Empty)
            {
                this.BeginInvoke(new SetTextCallbackBalance(SetTextBalance), new
object[] { InputDataBalance });
            }
        }
        private void SetTextBalance(string textBalance)
        {
            this.balanceReading.Text += textBalance;
        }

        private void OmegaTSerialPort_DataReceived(object sender,
SerialDataReceivedEventArgs e)
        {
            InputDataOmegaT = OmegaTSerialPort.ReadExisting();
            if (InputDataOmegaT != String.Empty)
            {
                this.BeginInvoke(new SetTextCallbackOmegaT(SetTextOmegaT), new
object[] { InputDataOmegaT });
            }
        }
        private void SetTextOmegaT(string textOmegaT)
        {
            this.OmegaTreading.Text += textOmegaT;
        }

        private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
        {
            BalanceSerialPort.PortName = COMportBalance.Text;
        }
        private void COMportOmega_SelectedIndexChanged(object sender, EventArgs
e)
        {
            OmegaTSerialPort.PortName = COMportOmega.Text;
        }


        private void button5_Click(object sender, EventArgs e)
        {
            this.OpenBalancePort();
        }


        private void OpenBalancePort()
        {
            try
            {
                if ((BalanceSerialPort.IsOpen) == false)
                {
                    BalanceSerialPort.PortName = COMportBalance.Text;
                    BalanceSerialPort.BaudRate = 38400;
                    BalanceSerialPort.Parity = Parity.None;
                    BalanceSerialPort.StopBits = StopBits.One;
                    BalanceSerialPort.Handshake = Handshake.None;
                    BalanceSerialPort.DataBits = 8;
                    BalanceSerialPort.DataReceived += new
SerialDataReceivedEventHandler(BalanceSerialPort_DataReceived);

                    BalanceSerialPort.Open();
                    MessageBox.Show("Port Opened Successfully !");
                    button5.Text = "Close COM Port";
                }


                else if ((BalanceSerialPort.IsOpen) == true)
                {
                    BalanceSerialPort.Close();
                    MessageBox.Show("Port Closed Successfully !");
                    button5.Text = "open COM Port";
//display changed according to button state of action
                }


            }
            catch
//exception
            {
                MessageBox.Show("Could Not Open/Close Specified Port !");
//display
            }
        }
        private void OpenOmegaTPort()
        {
            try
```

```
            {
                if ((OmegaTSerialPort.IsOpen) == false)
                {
                    OmegaTSerialPort.PortName = COMportOmega.Text;
//initializing serial port with proper argument from text box
                    OmegaTSerialPort.BaudRate = 19200;
                    OmegaTSerialPort.Parity = Parity.None;
                    OmegaTSerialPort.StopBits = StopBits.One;
                    OmegaTSerialPort.Handshake = Handshake.None;
                    OmegaTSerialPort.DataBits = 8;
                    OmegaTSerialPort.DataReceived += new
SerialDataReceivedEventHandler(OmegaTSerialPort_DataReceived);

                    OmegaTSerialPort.Open();  //open serial port
                    MessageBox.Show("Port Opened Successfully !");
                    button8.Text = "Close";
                }
                else if ((OmegaTSerialPort.IsOpen) == true)
                {
                    OmegaTSerialPort.Close();  //open serial port
                    MessageBox.Show("Port Closed Successfully !");
                    button8.Text = "Open";
                }
            }
            catch
            {
                MessageBox.Show("Could Not Open Specified Port !");
            }
        }
        private void CheckMassNow_Click(object sender, EventArgs e)
        {
            this.balanceReading.Text = "";
            TextToWriteBalance = "49 50 0d 0a"; // weight command P or IP
            msgBalance = DateTime.Now.ToString();
            arBalance =
WriteToComPortDelegateBalance1.BeginInvoke(TextToWriteBalance, new
AsyncCallback(WriteCompletedBalance), msgBalance);
        }

        private void TotalRunTimeText_TextChanged(object sender, EventArgs e)
        {
            TotalTimeHours.Text =
Convert.ToString(Convert.ToDouble(TotalRunTimeText.Text) / 60);
        }

        private void button6_Click(object sender, EventArgs e)
        {
            this.TareWeight();
        }

        private void saveAsToolStripMenuItem_Click(object sender, EventArgs e) //
Save input info as
        {
            try
            {
                SaveFileDialog saveFileDialog1 = new SaveFileDialog();
                saveFileDialog1.Filter = "Text Files | *.txt";
                saveFileDialog1.Title = "Save text File";
                saveFileDialog1.ShowDialog();
                FolderName =
System.IO.Path.GetDirectoryName(saveFileDialog1.FileName);

                // If file name is not an empty string open it for saving.
                if (saveFileDialog1.FileName != "")
                {
                    System.IO.FileStream fs =
                       (System.IO.FileStream)saveFileDialog1.OpenFile();
                    fs.Close();
                }

                using (System.IO.StreamWriter file = new
System.IO.StreamWriter(saveFileDialog1.FileName, true))
                {
                    file.WriteLine("NUMBER OF PUMPS: ");
                    file.WriteLine(Convert.ToString(nBombas));
                    file.WriteLine("NUMBER OF CYCLES: ");
                    file.WriteLine(textBoxNcycles.Text);

                    for (int j = 1; j <= nBombas; j++)
                    {
                        string WritePumpLabel = CtlArray[j - 1].ReadPumpLabel();
                        file.WriteLine("PUMP LABEL: ");
                        file.WriteLine(WritePumpLabel);

                        file.WriteLine("PUMP NUMBER: ");
                        file.WriteLine(Convert.ToString(j));

                        string WritePumpingDirection = CtlArray[j -
1].ReadPumpingDirection();
                        file.WriteLine("PUMPING DIRECTION: ");
                        file.WriteLine(WritePumpingDirection);

                        double WriteVolume = CtlArray[j - 1].ReadVolume();
                        file.WriteLine("VOLUME (ml): ");
                        file.WriteLine(Convert.ToString(WriteVolume));

                        double FlowRateToWrite = CtlArray[j - 1].ReadFlowRate();
                        file.WriteLine("FLOW RATE (ml/min): ");
                        file.WriteLine(Convert.ToString(FlowRateToWrite));

                        int WriteDelayTime = CtlArray[j - 1].ReadDelayTime(); //
seconds
                        file.WriteLine("DELAY (min): ");

file.WriteLine(Convert.ToString(Convert.ToDouble(WriteDelayTime / 60)));

                        int WriteRPM = CtlArray[j - 1].ReadRPMInput();
                        file.WriteLine("INPUT RPM CALIBRATION (rpm): ");
                        file.WriteLine(Convert.ToString(WriteRPM));
                        double ActualFlowRateToWrite = CtlArray[j -
1].ReadActualFlowRate();
                        file.WriteLine("ACTUAL FLOW RATE CALIBRATION (ml/min):
");
                        file.WriteLine(Convert.ToString(ActualFlowRateToWrite));
                    }
                }
            }
            catch (Exception)
            {
                StatusExperiment.Text = "Error while saving file";
            }
        }

        private void openToolStripMenuItem_Click(object sender, EventArgs e)
        {

            try
            {
                OpenFileDialog openFileDialog1 = new OpenFileDialog();
                openFileDialog1.Filter = "Text Files | *.txt";
                openFileDialog1.ShowDialog();

                if (openFileDialog1.FileName != "")
                {
                    System.IO.FileStream fs =
                       (System.IO.FileStream)openFileDialog1.OpenFile();
                    fs.Close();
                }
                using (System.IO.StreamReader file = new
System.IO.StreamReader(openFileDialog1.FileName, true))
                {
                    string[] lines =
System.IO.File.ReadAllLines(openFileDialog1.FileName);
                    int countLines = lines.Length;
                    StringReadLoadData = new string[countLines];
                    contar = 0;
                    foreach (string line in lines)
                    {
                        contar = contar + 1;
                        try
                        {
                            string valueLine = line;
                            StringReadLoadData[contar - 1] = valueLine;
                        }
                        catch (Exception ePeak)
                        {
                            MessageBox.Show("Error while loading data",
ePeak.Message);
                            return;
                        }
                    }

                    for (int j = 1; j <= nBombas; j++)
                    {
                        CtlArray[j - 1].WriteToTextBox(CtlArray[j - 1],
StringReadLoadData);
                    }
                }
            }
            catch (Exception)
            {
                StatusExperiment.Text = "Error while opening file";
            }
        }

        private void button7_Click(object sender, EventArgs e)
        {
            try
            {
                SmtpClient mySmtpClient = new SmtpClient("xxx.yyy.com");
                mySmtpClient.UseDefaultCredentials = false;
                mySmtpClient.Port = 25;

                //mySmtpClient.EnableSsl = true;
                mySmtpClient.Timeout = 100000000;
                mySmtpClient.DeliveryMethod = SmtpDeliveryMethod.Network;

                // add from,to mailaddresses
                MailAddress from = new MailAddress("fill_empty@email.com", "Fill
Empty Experiment");
                MailAddress to = new MailAddress(textBox2.Text, "TestToName");
                MailMessage myMail = new System.Net.Mail.MailMessage(from, to);

                // set subject and encoding
                myMail.Subject = "Message from Fill Empty";
                myMail.SubjectEncoding = System.Text.Encoding.UTF8;

                // set body-message and encoding
                myMail.Body = "Input Parameters for Fill-Empty Experiment";
                myMail.BodyEncoding = System.Text.Encoding.UTF8;

                //Attachments
                OpenFileDialog sendFileDialog = new OpenFileDialog();
                sendFileDialog.Filter = "Text Files | *.txt";
                sendFileDialog.Title = "Save text File";
                sendFileDialog.ShowDialog();

                Attachment atch = new Attachment(sendFileDialog.FileName,
MediaTypeNames.Application.Octet);
                myMail.Attachments.Add(atch);

                // text or html
                myMail.IsBodyHtml = true;
                mySmtpClient.Send(myMail);
                myMail.Dispose();
            }

            catch (Exception ex)
            {
```

```
                if (ex.InnerException != null)
                {
                    StatusExperiment.Text = Convert.ToString(ex.InnerException);
                }
            }

        }


        private void sendInputParametersToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            try
            {
                SmtpClient mySmtpClient = new SmtpClient("xxx.yyy.com");
                mySmtpClient.UseDefaultCredentials = false;
                mySmtpClient.Port = 25;

                //mySmtpClient.EnableSsl = true;
                mySmtpClient.Timeout = 100000000;
                mySmtpClient.DeliveryMethod = SmtpDeliveryMethod.Network;

                // add from,to mailaddresses
                MailAddress from = new MailAddress("fill_empty@email.com", "Fill
Empty Experiment");
                MailAddress to = new MailAddress(textBox2.Text, "TestToName");
                MailMessage myMail = new System.Net.Mail.MailMessage(from, to);

                // set subject and encoding
                myMail.Subject = "Message from Fill Empty";
                myMail.SubjectEncoding = System.Text.Encoding.UTF8;

                // set body-message and encoding
                myMail.Body = "Input Parameters for Fill-Empty Experiment";
                myMail.BodyEncoding = System.Text.Encoding.UTF8;

                //Attachments
                OpenFileDialog sendFileDialog = new OpenFileDialog();
                sendFileDialog.Filter = "Text Files | *.txt";
                sendFileDialog.Title = "Save text File";
                sendFileDialog.ShowDialog();

                Attachment atch = new Attachment(sendFileDialog.FileName,
MediaTypeNames.Application.Octet);
                myMail.Attachments.Add(atch);

                // text or html
                myMail.IsBodyHtml = true;
                mySmtpClient.Send(myMail);
                myMail.Dispose();
            }

            catch (Exception ex)
            {
                if (ex.InnerException != null)
                {
                    StatusExperiment.Text = Convert.ToString(ex.InnerException);
                }
            }

        }

        private void saveScreenshotToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            try
            {
                SaveFileDialog saveFileDialog2 = new SaveFileDialog();
                saveFileDialog2.Filter = "Image Files | *.jpg";
                saveFileDialog2.Title = "Save File";
                saveFileDialog2.ShowDialog();

                // If the file name is not an empty string open it for saving.
                if (saveFileDialog2.FileName != "")
                {
                    System.IO.FileStream fs =
                       (System.IO.FileStream)saveFileDialog2.OpenFile();
                    fs.Close();
                }
                Rectangle bounds = this.Bounds;
                using (Bitmap bitmap = new Bitmap(bounds.Width, bounds.Height))
                {
                    using (Graphics g = Graphics.FromImage(bitmap))
                    {
                        g.CopyFromScreen(new Point(bounds.Left, bounds.Top),
Point.Empty, bounds.Size);
                    }

                    bitmap.Save(saveFileDialog2.FileName, ImageFormat.Jpeg);
                }
            }
            catch (Exception)
            {
            }
        }

        private void button8_Click_1(object sender, EventArgs e)
        {
            this.OpenOmegaTPort();
        }

        private void TempCheckOption_SelectedIndexChanged(object sender,
EventArgs e)
        {
            if (TempCheckOption.Text == "YES")
            {
                OmegaTSerialPort = new SerialPort();
                getAvailablePorts(COMportOmega);
            }
        }
```

```
        private void button4_Click(object sender, EventArgs e)
        {
            this.OmegaTreading.Text = "";
            TextToWriteOmegaT = "2a 47 31 31 30 0d";
            // measure temp.
            msgOmegaT = DateTime.Now.ToString();
            arOmegaT =
WriteToComPortDelegateOmegaT1.BeginInvoke(TextToWriteOmegaT, new
AsyncCallback(WriteCompletedOmegaT), msgOmegaT);

        }

        private void currentCycleToolStripMenuItem_Click(object sender, EventArgs
e)
        {
            ChartTprofileFormCurrentCycle = new GraphForm("Current Cycle");
            ChartTprofileFormCurrentCycle.Name = "MyForm1";
            ChartTprofileFormCurrentCycle.Text = "Current cycle";
            ChartTprofileFormCurrentCycle.Show();
        }


        private void combinedCyclesToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            ChartTprofileForm = new GraphForm("Combined Cycles");
            ChartTprofileForm.Name = "MyForm2";
            ChartTprofileForm.Text = "Combined cycles";
            ChartTprofileForm.Show();
        }

        private void consecutiveCyclesToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            ChartTprofileFormAll = new GraphForm("All consecutive cycles");
            ChartTprofileFormAll.Name = "MyForm3";
            ChartTprofileFormAll.Text = "Combined cycles";
            ChartTprofileFormAll.Show();
        }

        private void allGraphsToolStripMenuItem_Click(object sender, EventArgs e)
        {
            ChartTprofileFormCurrentCycle = new GraphForm("Current Cycle");
            ChartTprofileFormCurrentCycle.Name = "MyForm1";
            ChartTprofileFormCurrentCycle.Text = "Current cycle";
            ChartTprofileFormCurrentCycle.Show();

            ChartTprofileForm = new GraphForm("Combined Cycles");
            ChartTprofileForm.Name = "MyForm2";
            ChartTprofileForm.Text = "Combined cycles";
            ChartTprofileForm.Show();

            ChartTprofileFormAll = new GraphForm("All consecutive cycles");
            ChartTprofileFormAll.Name = "MyForm3";
            ChartTprofileFormAll.Text = "Combined cycles";
            ChartTprofileFormAll.Show();
        }

        private void massBalanceToolStripMenuItem_Click(object sender, EventArgs
e)
        {
            ChartBalance = new GraphForm("Mass Balance");
            ChartBalance.Name = "MyForm4";
            ChartBalance.Text = "Mass Balance";
            ChartBalance.Show();
        }


        public void button9_Click(object sender, EventArgs e)
        {
            for (int i = 1; i <= nBombas; i++)
            {
                CtlArray[i - 1].openPumpPort();
            }
        }


        public void button10_Click(object sender, EventArgs e)
        {
            try
            {
                for (int i = 1; i <= nBombas; i++)
                {
                    CtlArray[i - 1].ConnectPump(CtlArray[i - 1]); // Send command
05 to pumps   ---- P?0
                }

                for (int j = 1; j <= nBombas; j++)
                {
                    CtlArray[j - 1].SetPumpNumber(CtlArray[j - 1]);
                }

                for (int j = 1; j <= nBombas; j++)
                {
                    CtlArray[j - 1].SetPumpToRemote(CtlArray[j - 1]);
                }

                MessageBox.Show("Established connection with PC");
            }
            catch (Exception)
            {
                MessageBox.Show("Error while connecting to PC.");
            }
        }

        private void OmegaTreading_TextChanged(object sender, EventArgs e)
        {
            try
            {
```

```
                string TempString = OmegaTreading.Text.Trim();

                if (TempString.Contains("G110+") == true)
                {
                    string separator1 = "G110+";
                    string[] separators0 = { separator1 };
                    string[] StringTrim0 = TempString.Split(separators0,
System.StringSplitOptions.RemoveEmptyEntries);

                    Tmeasured = Convert.ToDouble(StringTrim0[0]) / 10;
                    Tnow.Text = Convert.ToString(Tmeasured);
                    Tnow.Text = string.Format("{0:F1}", Tmeasured);
                }

                else if (TempString.Contains("G110-") == true)
                {
                    string separator1 = "G110-";
                    string[] separators0 = { separator1 };
                    string[] StringTrim0 = TempString.Split(separators0,
System.StringSplitOptions.RemoveEmptyEntries);

                    Tmeasured = -Convert.ToDouble(StringTrim0[0]) / 10;
                    Tnow.Text = Convert.ToString(Tmeasured);
                    Tnow.Text = string.Format("{0:F1}", Tmeasured);
                }
            }
            catch (Exception) { }
        }

        private void balanceReading_TextChanged(object sender, EventArgs e)
        {
            try
            {
                BalanceReadingTrim = (balanceReading.Text).Trim();

                if ((balanceReading.Text != "") &
((BalanceReadingTrim.StartsWith("Sample Name:")) == true))
                {
                    string inputline = "";
                    int countChar = 0;
                    string outputline = "";
                    MassStringTrim = "";

                    foreach (char line in BalanceReadingTrim)
                    {
                        countChar = countChar + 1;
                        if (countChar <= 12)
                        {
                            inputline = inputline + Convert.ToString(line);
                        }  // Sample Name:
                        if ((countChar > 12) & (Convert.ToString(line) != " ") &
(Convert.ToString(line) != "g"))
                        { outputline = outputline + Convert.ToString(line); }
                        if ((countChar > 12) & (Convert.ToString(line) == "g"))
                        { break; }
                    }

                    string separator3 = " ";
                    string[] separators0 = { separator3, "[0]", "[1]", "[2]",
"[3]", "[4]", "[5]", "[6]", "[7]", "[8]", "[9]" };
                    string[] StringTrim0 = (outputline.Trim()).Split(separators0,
System.StringSplitOptions.RemoveEmptyEntries);
                    string separator4 = ".";
                    string[] separators1 = { separator4 };
                    string[] StringTrim1 = StringTrim0[0].Split(separators1,
System.StringSplitOptions.RemoveEmptyEntries);

                    if (StringTrim1.Length == 2)
                    {
                        MassStringTrim = StringTrim1[0] + "," + StringTrim1[1];
                        MassNow = Convert.ToDouble(MassStringTrim);
                        ActualMass.Text = Convert.ToString(MassNow);
                        ActualMass.Text = string.Format("{0:F2}", MassNow);
                    }
                }
            }
            catch (Exception) { }
        }
    }
}
```

Peristaltic Pump Class (User Control)

Methods	**Description**	**Output**
OpenPumpPort	Open/close port	void
GetAvailablePorts	Detects and displays COM ports for pump connection	void
SetText	Displays serial port read response	void
WriteToComPort	Sends bytes to pump serial port to execute command	void
WriteCompleted	Obtains message (command) sent and if the operation succedded returns true. Otherwise, returns false.	bool
ConfigPorts	Configures serial port (4800-7-1-odd-none)	void
RetrievePumpStatus	Returns number status according to pump manual from read serial port response	int
EvaluateStatus	Returns true if pump is waiting for instructions	bool
RunPump	Starts pump with parameters as input (flow direction, revolutions, speed)	void
AskForPumpStatus	Sends command to pump to get pump status	void
ReadPumpStatus	Gets pump status as string	string
StopPump	Stops pump	void
ConvertFlowRateTo-RotorSpeed	Converts flow rate to rotor speed based on desired flowrate and calibration information (rpm-actual flowrate)	string
DoCalculate	Calculates pumping time and displays it in textbox	void
ReadDelayTime	Calculates in seconds time for pump delay from input in minutes given as input in Delay.Text textbox	void
ReadRPMInput	Returns as integer the number of revolutions per minute from input RPM.Text for calibration purposes	int
ReadPumpingDirection	Returns pump direction from input Direction.Text	string
ReadVolume	Returns volume to pump from Volume.Text	double
ReadFlowRate	Returns flowrate in ml/min from input in FlowRate.Text	double

ReadActualFlowRate	Returns actual flowrate in ml/min from input in ActualFlowRate.Text for calibration purposes	double
ConnectPump	Sends command 05 to establish communication with pump	void
SetPumpNumber	Sends command 02 50 30 3n 0d to set identifier number to pump	void
SetPumpToRemote	Sends command 02 50 3n 52 0d to set pump to remote mode	void
SetDefaultValues	Initializes values for pump parameters in usercontrol textboxes	void
WriteToTextBox	Writes parameters into user control textboxes from string array	void

Events	**Description**	**Output**
myport_dataReceived	Reads serial port and displays it in pump status textBox	void
Volume_TextChanged	Reads volume and executes DoCalculate()	void
FlowRate_TextChanged	Reads flowrate and executes DoCalculate()	void
Delay_TextChanged	Reads delay time	void
RPM_TextChanged	Reads RPM	void
ActualFlowRate_Text-Changed	Reads actual flow rate	void
Direction_Selected-IndexChanged	Reads pumping direction	void
runCCW_click	Starts pump with specified parameters in textboxes in counterclockwise direction	void
runCC_click	Starts pump with specified parameters in textboxes in clockwise direction	void
stopPump_click	Stops pump	void

Buttons	**Description**	Input
runCCW	Starts pump with specified parameters in textboxes in counterclockwise direction	click
runCC	Starts pump with specified parameters in textboxes in clockwise direction	click
stopPump	Stops pump	click

```
using System;
using System.Drawing;
using System.Data;
using System.Linq;
using System.Windows.Forms;
using System.IO.Ports;
using System.Runtime.Remoting.Messaging;

namespace Peristaltic_General.Controls
{
    public partial class ctlPump : UserControl
    {
        public string StringRotorSpeed;
        public int nBombas = 1;
        public int DelayTimeSec;
        public int PumpingTimeSec;
        public double PumpingTimeMin;

        public static double test;
        public double VolumeInput;
        public double FlowRateInput;
        public double PumpingTimeCalc;
        public string Pumpingdirection;

        public int RPMInput;
        public double ActualFlowRateInput;
        public double DelayTimeInputMin;
        public string responsePump;
        public string stringResponse;

        public SerialPort myport;
        public string respuesta1;
        public string CutStringStatus;
        public string outputline;
        public string wordReceived;
```

```
        internal delegate void
SerialDataReceivedEventHandlerDelegate(object sender,
SerialDataReceivedEventArgs e);
        delegate void SetTextCallback(string text);
        string InputData = String.Empty;

        internal delegate Boolean WriteToComPortDelegate(string
textToWrite,SerialPort myport);
        internal static WriteToComPortDelegate WriteToComPortDelegate1 =
new WriteToComPortDelegate(WriteToComPort);

        public IAsyncResult ar;
        public static string msg;
        public int StatusChanged = 0;
        public bool TemperatureOutsideOfRange = false;

        public string outputlineEvaluate;
        public bool TrueFalseEvaluate;
        public string EvaluatePumpString;

        public ctlPump()
        {
            InitializeComponent();
            myport = new SerialPort();
            myport.DataReceived += new
System.IO.Ports.SerialDataReceivedEventHandler(myport_DataReceived);

            this.getAvailablePorts();
            this.SetDefaultValues();
            this.DoCalculate();
        }
        public void openPumpPort()
        {
            if ((myport.IsOpen) == false)
            {
                ConfigPorts();
```

```
                myport.Open();
                MessageBox.Show("Port Opened Successfully !");
            }
            else if ((myport.IsOpen) == true)
            {
                myport.Close();
                MessageBox.Show("Port Closed Successfully !");
            }
        }
        void getAvailablePorts()
        {
            string[] ports = SerialPort.GetPortNames();
            comboBox1.Items.AddRange(ports);
        }
        private void myport_DataReceived(object sender,
System.IO.Ports.SerialDataReceivedEventArgs e)
        {
            InputData = myport.ReadExisting();
            if (InputData != String.Empty)
            {
                this.BeginInvoke(new SetTextCallback(SetText), new
object[] { InputData });
            }
        }
        private void SetText(string text)
        {
            wordReceived += text;
            this.rtbIncoming.Text = wordReceived;
        }

        internal static void WriteCompleted(IAsyncResult ar)
        {
            WriteToComPortDelegate deleg;
            Boolean success;

            deleg =
(WriteToComPortDelegate)((AsyncResult)ar).AsyncDelegate;
            msg = (string)ar.AsyncState;
            success = WriteToComPortDelegate1.EndInvoke(ar);
        }

        internal static Boolean WriteToComPort(string TextToWrite,
SerialPort myport)
        {
            Boolean success = false;
            if (myport.IsOpen)
            {
                byte[] bytes = TextToWrite.Split(' ').Select(s =>
Convert.ToByte(s, 16)).ToArray();
                myport.Write(bytes, 0, bytes.Length);
                success = true;
            }
            return success;
        }

        public void ConfigPorts()
        {
            myport.PortName = comboBox1.Text;
            myport.Parity = Parity.Odd;
            myport.StopBits = StopBits.One;
            myport.Handshake = Handshake.None;
            myport.DataBits = 7;
            myport.BaudRate = Convert.ToInt32("4800");
        }

        private void runCCW_Click(object sender, EventArgs e)
        {
            this.RunPump(this, "Counterclockwise", 99999,
ConvertFlowRateToRotorSpeed(this));
        }

        private void stopPump_Click(object sender, EventArgs e)
        {
            this.StopPump(this);
        }

        private void runCC_Click(object sender, EventArgs e)
        {
            this.RunPump(this, "Clockwise", 99999,
ConvertFlowRateToRotorSpeed(this));
        }

        private int RetrievePumpStatus(ctlPump bomba, string Status)
        {
            try
            {
                string Message = "";
                int numberStatus = 0;
                string trimString = "";
                string trimString2 = "";
                string charString = "";
                string stringStatus = "";
                string CutStringStatus = "";
                int endString = Status.Length;
                charString = Status;


                for (int k = 0; k <= endString - 1; k++)
                {
                    if (k == 0)
                    {
                        trimString = "";
                        charString = Status;
                        trimString = charString.Remove(k+1, endString-1);

                        if (trimString == "P")
                        {
                            charString = Status;
                            stringStatus=charString.Remove(9,endString-9);
                            CutStringStatus = stringStatus.Remove(0, 7);
                            k = endString - 1;
                        }
                    }
                    if (k != 0)
                    {
                        trimString = "";
                        charString = Status;
                        trimString = charString.Remove(0, k);
                        trimString2 = trimString.Remove(1, endString-k-1);

                        if (trimString2 == "P")
                        {
                            charString = Status;
                            trimString = charString.Remove(0, k);
                            CutStringStatus = trimString.Remove(0, 7);

                            if (trimString.Length > 9)
                            {
                                trimString2 =
trimString.Remove(9,endString-9-k);
                                CutStringStatus = trimString2.Remove(0,7);
                            }
                            k = endString - 1;
                        }
                    }
                }


                if ((Convert.ToInt32(CutStringStatus) == 10) ||
(Convert.ToInt32(CutStringStatus) == 12) ||
(Convert.ToInt32(CutStringStatus) == 15))
                {
                    Message = "Pump waiting for instructions.
Communication OK.";
                    numberStatus = 1;
                }
                if ((Convert.ToInt32(CutStringStatus) == 20) ||
(Convert.ToInt32(CutStringStatus) == 22) ||
(Convert.ToInt32(CutStringStatus) == 25))
                {
                    Message = "Pump instructed, waiting to go.
Communication OK.";
                    numberStatus = 2;
                }
                if ((Convert.ToInt32(CutStringStatus) == 30) ||
(Convert.ToInt32(CutStringStatus) == 32) ||
(Convert.ToInt32(CutStringStatus) == 35))
                {
                    Message = "Pump running. Communication OK.";
                    numberStatus = 3;
                }
                if (CutStringStatus == "40")
                {
                    Message = "Pump stopped by local switch. Communication
OK.";
                }
                if (CutStringStatus == "50")
                {
                    Message = "No motor feedback. Communication OK.";
                }
                if (CutStringStatus == "60")
                {
                    Message = "Overload. Communication OK.";
                }
                if (CutStringStatus == "70")
                {
                    Message = "Excessive motor feedback. Communication
OK.";
                }

                else
                {
                    Message = "Error occurred while retrieving pump status
" + Message;
                }
                return numberStatus;
            }
            catch (Exception)
            {
                return 0;
            }
        }
        public bool EvaluateStatus(ctlPump bomba)
        {
            try
            {
                int countChar = 0;
                outputlineEvaluate = "";
                rtbIncoming.Text= "";

                this.AskForPumpStatus(this);

                foreach (char line in (rtbIncoming.Text).Trim())
                {
                    if (Convert.ToString(line) == "P")
                    {
                        countChar = 1;
                    }
                    if ((countChar <= 9) & (countChar >= 1))
                    {
                        countChar = countChar + 1;
                        outputlineEvaluate = outputlineEvaluate +
Convert.ToString(line);
                    }
                    if (countChar > 9)
                    {
                        EvaluatePumpString = (outputlineEvaluate).Trim();
                        break;
                    }
                }
```

```
                string trimString = "";
                string trimString2 = "";
                string charString = "";
                string stringStatus = "";
                string CutStringStatus = "";
                int endString = EvaluatePumpString.Length;
                charString = EvaluatePumpString;

                for (int k = 0; k <= endString - 1; k++)
                {
                    if (k == 0)
                    {
                        trimString = "";
                        charString = EvaluatePumpString;
                        trimString = charString.Remove(k+1,endString - 1);

                        if (trimString == "P")
                        {
                            charString = EvaluatePumpString;
                            stringStatus=charString.Remove(9,endString-9);
                            CutStringStatus = stringStatus.Remove(0, 7);
                            k = endString - 1;
                        }
                    }
                    if (k != 0)
                    {
                        trimString = "";
                        charString = EvaluatePumpString;
                        trimString = charString.Remove(0, k);
                        trimString2 = trimString.Remove(1,endString-k-1);

                        if (trimString2 == "P")
                        {
                            charString = EvaluatePumpString;
                            trimString = charString.Remove(0, k);
                            CutStringStatus = trimString.Remove(0, 7);

                            if (trimString.Length > 9)
                            {
                                trimString2 =
trimString.Remove(9,endString - 9 - k);
                                CutStringStatus = trimString2.Remove(0,7);
                            }
                            k = endString - 1;
                        }
                    }
                }


                if ((Convert.ToInt32(CutStringStatus) == 10) ||
(Convert.ToInt32(CutStringStatus) == 12) ||
(Convert.ToInt32(CutStringStatus) == 15))
                {
                    Message = "Pump waiting for instructions.
Communication OK.";
                    numberStatus = 1;
                    return true;
                }
                if ((Convert.ToInt32(CutStringStatus) == 20) ||
(Convert.ToInt32(CutStringStatus) == 22) ||
(Convert.ToInt32(CutStringStatus) == 25))
                {
                    Message = "Pump instructed, waiting to go.
Communication OK.";
                    numberStatus = 2;
                    return false;
                }
                if ((Convert.ToInt32(CutStringStatus) == 30) ||
(Convert.ToInt32(CutStringStatus) == 32) ||
(Convert.ToInt32(CutStringStatus) == 35))
                {
                    Message = "Pump running. Communication OK.";
                    numberStatus = 3;
                    return false;
                }
                if (CutStringStatus == "40")
                {
                    Message = "Pump stopped by local switch. Communication
OK.";
                    return false;
                }
                if (CutStringStatus == "50")
                {
                    Message = "No motor feedback. Communication OK.";
                    return false;
                }
                if (CutStringStatus == "60")
                {
                    Message = "Overload. Communication OK.";
                    return false;
                }
                if (CutStringStatus == "70")
                {
                    Message = "Excessive motor feedback. Communication
OK.";
                    return false;
                }
                else { return false; }
            }
            catch (Exception)
            {
                return false;
            }
        }

        public void RunPump(ctlPump bomba, string flowDirection, int
inputRev, string SpeedAsInput)
        {
            try
            {
                if (TemperatureOutsideOfRange == false)
                {

                    if ((LabelText.Text != "OFF") || (LabelText.Text !=
"off") || (LabelText.Text != "Off"))
                    {
                        string numberOfRevolutions =
inputRev.ToString("D5");
                        char[] characters =
numberOfRevolutions.ToCharArray();
                        string numberOfRevolutionsHex =
Convert.ToInt16(characters[0]).ToString("x") + " " +
Convert.ToInt16(characters[1]).ToString("x") + " " +
Convert.ToInt16(characters[2]).ToString("x") + " " +
Convert.ToInt16(characters[3]).ToString("x") + " " +
Convert.ToInt16(characters[4]).ToString("x");
                        string pumpDirection = "2b";

                        if (flowDirection == "Clockwise")
                        {
                            pumpDirection = "2b";
                        }
                        if (flowDirection == "Counterclockwise")
                        {
                            pumpDirection = "2d";
                        }

                        string str = "02 50 30 3" + bomba.Name + " 53 " +
pumpDirection + " " + SpeedAsInput + " 56 " + numberOfRevolutionsHex + "
2e 30 30 47 0d";

                       wordReceived = "";
                        msg = DateTime.Now.ToString();
                        ar = WriteToComPortDelegate1.BeginInvoke(str,
myport, new AsyncCallback(WriteCompleted), msg);
                        System.Threading.Thread.Sleep(150);
                        pumpNumber.BackColor = Color.Green;


                        if ((this.rtbIncoming.Text).Trim() == "")
                        {
                            pumpNumber.BackColor = Color.Orange;
                            System.Threading.Thread.Sleep(50);

                            if ((this.rtbIncoming.Text).Trim() == "")
                            {
                                this.ConnectPump(this); // Send command 05
to pump   ---- P?0
                                System.Threading.Thread.Sleep(50);

                                this.SetPumpNumber(this);
                                System.Threading.Thread.Sleep(50);

                                this.SetPumpToRemote(this);
                                System.Threading.Thread.Sleep(50);

                                pumpNumber.BackColor = Color.Orange;

                                msg = DateTime.Now.ToString();
                                ar =
WriteToComPortDelegate1.BeginInvoke(str, myport, new
AsyncCallback(WriteCompleted), msg);
                                System.Threading.Thread.Sleep(150);
                                pumpNumber.BackColor = Color.Green;

                                if ((this.rtbIncoming.Text).Trim() == "")
                                {
                                    this.ConnectPump(this); // Send
command 05 to pump   ---- P?0
                                    System.Threading.Thread.Sleep(50);

                                    this.SetPumpNumber(this);
                                    System.Threading.Thread.Sleep(50);

                                    this.SetPumpToRemote(this);
                                    System.Threading.Thread.Sleep(50);
                                    pumpNumber.BackColor = Color.Orange;

                                    msg = DateTime.Now.ToString();
                                    ar =
WriteToComPortDelegate1.BeginInvoke(str, myport, new
AsyncCallback(WriteCompleted), msg);
                                    System.Threading.Thread.Sleep(150);
                                    pumpNumber.BackColor = Color.Green;
                                }
                            }
                            else
                            {
                                wordReceived = "";
                                msg = DateTime.Now.ToString();
                                ar =
WriteToComPortDelegate1.BeginInvoke(str, myport, new
AsyncCallback(WriteCompleted), msg);
                                System.Threading.Thread.Sleep(150);
                                pumpNumber.BackColor = Color.Green;
                            }
                        }
                    }
                }
            }
            catch (Exception)
            {
                pumpNumber.BackColor = Color.Orange;
            }
        }

        public void AskForPumpStatus(ctlPump bomba)
        {
            try
            {
```

```
                string str = "02 50 30 3" + bomba.Name + " 49 0d";
                wordReceived = "";
                msg = DateTime.Now.ToString();
                ar = WriteToComPortDelegate1.BeginInvoke(str, myport, new
AsyncCallback(WriteCompleted), msg);
                System.Threading.Thread.Sleep(100);
            }
            catch (Exception)
            {
                pumpNumber.BackColor = Color.Orange;
            }
        }

        public string ReadPumpStatus(ctlPump bomba)
        {
            try
            {
                return CutStringStatus = PumpStatus.Text.Trim();
            }
            catch (Exception)
            {
                PumpStatus.Text = "N/A";
                return "N/A";
            }
        }

        public void StopPump(ctlPump bomba)
        {
            try
            {
                string str = "02 50 30 3" + bomba.Name + " 5a 0d";
                wordReceived = "";
                msg = DateTime.Now.ToString();
                ar = WriteToComPortDelegate1.BeginInvoke(str, myport, new
AsyncCallback(WriteCompleted), msg);
                System.Threading.Thread.Sleep(150);
                pumpNumber.BackColor = Color.Red;

                if ((this.rtbIncoming.Text).Trim() == "")
                {
                    pumpNumber.BackColor = Color.Orange;

                    System.Threading.Thread.Sleep(50);

                    if ((this.rtbIncoming.Text).Trim() == "")
                    {
                        this.ConnectPump(this); // Send command 05 to pump
                        this.SetPumpNumber(this);
                        this.SetPumpToRemote(this);
                        pumpNumber.BackColor = Color.Orange;

                        msg = DateTime.Now.ToString();
                        ar = WriteToComPortDelegate1.BeginInvoke(str,
myport, new AsyncCallback(WriteCompleted), msg);
                        System.Threading.Thread.Sleep(100);
                        pumpNumber.BackColor = Color.Red;

                        if ((this.rtbIncoming.Text).Trim() == "")
                        {
                            this.ConnectPump(this); // Send command 05 to
pump   ---- P?0
                            this.SetPumpNumber(this);
                            this.SetPumpToRemote(this);
                            pumpNumber.BackColor = Color.Orange;

                            msg = DateTime.Now.ToString();
                            ar = WriteToComPortDelegate1.BeginInvoke(str,
myport, new AsyncCallback(WriteCompleted), msg);
                            System.Threading.Thread.Sleep(100);
                            pumpNumber.BackColor = Color.Red;
                        }
                    }
                    else
                    {
                        wordReceived = "";
                        msg = DateTime.Now.ToString();
                        ar = WriteToComPortDelegate1.BeginInvoke(str,
myport, new AsyncCallback(WriteCompleted), msg);
                        System.Threading.Thread.Sleep(100);
                        pumpNumber.BackColor = Color.Red;
                    }
                }
            }
            catch (Exception)
            {
                pumpNumber.BackColor = Color.Orange;
            }
        }
        public string ConvertFlowRateToRotorSpeed(ctlPump bomba)
        {
            try
            {
                if ((LabelText.Text != "OFF") || (LabelText.Text != "off")
|| (LabelText.Text != "Off"))
                {
                    string StringRotorSpeedHex;
                    double rotorSpeed1 = Convert.ToDouble(FlowRate.Text) *
Convert.ToDouble(RPM.Text) / Convert.ToDouble(ActualFlowRate.Text);
                    int value1 = Convert.ToInt16(rotorSpeed1);
                    StringRotorSpeed = value1.ToString("D4");

                    char[] characters = StringRotorSpeed.ToCharArray();
                    StringRotorSpeedHex =
Convert.ToInt16(characters[0]).ToString("x") + " " +
Convert.ToInt16(characters[1]).ToString("x") + " " +
Convert.ToInt16(characters[2]).ToString("x") + " " +
Convert.ToInt16(characters[3]).ToString("x");
                    StringRotorSpeed = StringRotorSpeedHex + " 2e 30";

                    return StringRotorSpeed;
                }
                else
                { return "30 30 30 30 2e 30"; }
            }
            catch (Exception err)
            {
                pumpNumber.BackColor = Color.Orange;
                MessageBox.Show("Error when trying to convert flow rate to
rotor speed");
                return "30 30 30 30 2e 30";
            }
        }

        public void SetZeroByte()
        {            string str = " 00 00 00";        }

        public void DoCalculate()
        {
            Pumpingdirection = ReadPumpingDirection();
            DelayTimeSec = ReadDelayTime();

            VolumeInput = ReadVolume();
            FlowRateInput = ReadFlowRate();
            PumpingTimeCalc = 0;
            PumpingTimeMin = 0;
            PumpingTimeSec = 0;

            // calculating filling time
            if ((VolumeInput == 0.0) || (FlowRateInput == 0.0))
            {
                PumpingTimeCalc = 0;
                PumpingTimeMin = 0;
                PumpingTimeSec = 0;
            }
            else
            {
                PumpingTimeMin = VolumeInput / FlowRateInput; // min
                PumpingTimeSec = Convert.ToInt32(VolumeInput /
FlowRateInput*60); // ml/ (ml/min) * 60 ==  seconds
            }
            PumpingTime.Text = PumpingTimeMin.ToString();
            PumpingTime.Text = string.Format("{0:F3}", PumpingTimeMin);
        }

        public int ReadDelayTime()
        {
            DelayTimeInputMin = Convert.ToDouble(Delay.Text);
            DelayTimeInputMin = double.Parse(Delay.Text);
            int DelayTimeInputSec = Convert.ToInt32(DelayTimeInputMin *
60);
            return DelayTimeInputSec;
        }
        public int ReadDelayTime(ctlPump bomba)
        {
            double DelayTimeInputMin = Convert.ToDouble(Delay.Text);
            DelayTimeInputMin = double.Parse(Delay.Text);
            int DelayTimeInputSec = Convert.ToInt32(DelayTimeInputMin *
60);
            return DelayTimeInputSec;
        }

        public int ReadRPMInput()
        {
            int RPMInput = Convert.ToInt32(RPM.Text);
            return RPMInput;
        }
        public string ReadPumpingDirection()
        {
            string PumpingDirectionInput = Direction.Text;
            return PumpingDirectionInput;
        }
        public string ReadPumpLabel()
        {
            string PumpLabelInput = LabelText.Text;
            return PumpLabelInput;
        }
        public double ReadVolume()
        {
            double VolumeInputToPump = Convert.ToDouble(Volume.Text);
            return VolumeInputToPump;
        }
        public double ReadFlowRate() // in ml/min
        {
            double FlowRateInputPump = Convert.ToDouble(FlowRate.Text);
            return FlowRateInputPump;
        }
        public double ReadActualFlowRate()
        {
            double ActualFlowRateInputPump =
Convert.ToDouble(ActualFlowRate.Text);
            return ActualFlowRateInputPump;
        }
        public void ConnectPump(ctlPump bomba)
        {
            try
            {
                if ((LabelText.Text != "OFF") || (LabelText.Text != "off")
|| (LabelText.Text != "Off"))
                {
                    SetZeroByte();
                    string str = "05";
                    wordReceived = "";
                    msg = DateTime.Now.ToString();
                    ar = WriteToComPortDelegate1.BeginInvoke(str, myport,
new AsyncCallback(WriteCompleted), msg);
                    System.Threading.Thread.Sleep(100);
                }
            }
            catch (Exception)
            {
```

```
                pumpNumber.BackColor = Color.Orange;
            }
        }

        public void SetPumpNumber(ctlPump bomba)
        {
            try
            {
                if ((LabelText.Text != "OFF") || (LabelText.Text != "off")
|| (LabelText.Text != "Off"))
                {
                    SetZeroByte(); string str = "02 50 30 3" +
Convert.ToString(pumpNumber.Text) + " 0d";
                    wordReceived = "";
                    msg = DateTime.Now.ToString();
                    ar = WriteToComPortDelegate1.BeginInvoke(str, myport,
new AsyncCallback(WriteCompleted), msg);
                    System.Threading.Thread.Sleep(100);
                }
            }
            catch (Exception)
            {
                pumpNumber.BackColor = Color.Orange;
            }
        }

        public void SetPumpToRemote(ctlPump bomba)
        {
            try
            {
                if ((LabelText.Text != "OFF") || (LabelText.Text != "off")
|| (LabelText.Text != "Off"))
                {
                    SetZeroByte();
                    string str = "02 50 30 3" +
Convert.ToString(bomba.Name) + " 52 0d";

                    wordReceived = "";
                    msg = DateTime.Now.ToString();
                    ar = WriteToComPortDelegate1.BeginInvoke(str,
myport,new AsyncCallback(WriteCompleted), msg);
                    System.Threading.Thread.Sleep(100);
                }
            }
            catch (Exception)
            {
                pumpNumber.BackColor = Color.Orange;
            }
        }

        public void SetDefaultValues()
        {
            Direction.Text = "Clockwise";
            Volume.Text = "10";
            FlowRate.Text = "50";
            Delay.Text = "0";
            RPM.Text = "50";
            ActualFlowRate.Text = "50";
            PumpingTime.Text = "0";
        }

        private void Volume_TextChanged(object sender, EventArgs e)
        {
                try
                {
                    VolumeInput = ReadVolume();
                    this.DoCalculate();
                }
                catch (Exception)
                {
                    Volume.Text = "10";
                }
        }

        private void FlowRate_TextChanged(object sender, EventArgs e)
        {
                try
                {
                    FlowRateInput = ReadFlowRate();
                    this.DoCalculate();
                }
                catch (Exception)
                {
                    FlowRate.Text = "50";
                }
        }

        private void Delay_TextChanged(object sender, EventArgs e)
        {
            try
            {
                DelayTimeSec = ReadDelayTime();

                if ((Delay.Text == "") || (Delay.Text == " "))
                {
                    Delay.Text = "0";
                }
            }
            catch (Exception)
            {
                Delay.Text = "0";
            }
        }

        private void RPM_TextChanged(object sender, EventArgs e)
        {
            {
                try
                {
                    RPMInput = ReadRPMInput();
                    if ((RPM.Text == "") || (RPM.Text == " "))
                    {
                        RPM.Text = "0";
                    }
                }
                catch (Exception)
                {
                    RPM.Text = "0";
                }
            }
        }

        private void ActualFlowRate_TextChanged(object sender, EventArgs
e)
        {
            {
                try
                {
                    ActualFlowRateInput = ReadActualFlowRate();

                    if ((ActualFlowRate.Text == "") ||
(ActualFlowRate.Text == " "))
                    {
                        ActualFlowRate.Text = "0";
                    }
                }
                catch (Exception)
                {
                    ActualFlowRate.Text = "0";
                }
            }
        }

        private void Direction_SelectedIndexChanged(object sender,
EventArgs e)
        {
            try { this.DoCalculate(); }
            catch (Exception) {  }
        }
        public void WriteToTextBox(ctlPump bomba, string[] InputText)
        {
            int numb = Convert.ToInt16(bomba.Name);
            int jump = 16;

            LabelText.Text = InputText[5 + jump * (numb - 1)];
            pumpNumber.Text = InputText[7 + jump * (numb - 1)];
            Direction.Text = InputText[9 + jump * (numb - 1)];
            Volume.Text = InputText[11 + jump * (numb - 1)];
            FlowRate.Text = InputText[13 + jump * (numb - 1)];
            Delay.Text = InputText[15 + jump * (numb - 1)];
            RPM.Text = InputText[17 + jump * (numb - 1)];
            ActualFlowRate.Text = InputText[19 + jump * (numb - 1)];
        }

        public void textBox1_TextChanged(object sender, EventArgs e)
        {
            try
            {
                if (((LabelText.Text != "OFF") || (LabelText.Text !=
"off") || (LabelText.Text != "Off")) &
((rtbIncoming.Text).Trim().Contains("P") == true))
                {
                    int countChar = 0;
                    outputline = "";
                    foreach (char line in (rtbIncoming.Text).Trim())
                    {
                        if (Convert.ToString(line) == "P")
                        {
                            countChar = 1;
                        }
                        if ((countChar <= 9) & (countChar >= 1))
                        {
                            countChar = countChar + 1;
                            outputline = outputline +
Convert.ToString(line);
                        }
                        if (countChar > 9)
                        {
                            PumpStatus.Text = (outputline).Trim(); // OK
                            break;
                        }
                    }

                    if ((rtbIncoming.Text).Trim().Contains("??") == true)
                    {
                        this.ConnectPump(this); // Send command 05 to pump
                        System.Threading.Thread.Sleep(50);

                        this.SetPumpNumber(this);
                        System.Threading.Thread.Sleep(50);

                        this.SetPumpToRemote(this);
                        System.Threading.Thread.Sleep(50);

                        pumpNumber.BackColor = Color.Orange;
                        this.AskForPumpStatus(this);
                    }
                }
            }
            catch (Exception)
            { }
        }

        private void PumpStatus_TextChanged(object sender, EventArgs e)
        {
            StatusChanged = StatusChanged + 1;
            textBoxStatusChanged.Text = Convert.ToString(StatusChanged);
        }
    }
}
```