TECHNEAU Water Treatment Simulator:

Modelling Framework (Version 1.0)

TECHNEAU
September 2008

TECHNEAU Water Treatment Simulator:

Modelling Framework (Version 1.0)

Colofon

Title: TECHNEAU Water Treatment Simulator:
Modelling Framework (Version 1.0)

Author(s)
Luuk Rietveld, Petra Ross (TUD);
Jeremy Dudley (WRc)

Quality Assurance
By Glenn Dillon (WRc)

Deliverable number
D 5.4.4.

This report is: PU = Public

Contents

1 Introduction

The objective of TECHNEAU Work Package 5.4 “Development of a water treatment plant simulator” is to produce a European platform for modelling of drinking water treatment processes. This document - the fourth deliverable of the project (D5.4.4) - describes the first version of the modelling platform, also known as ‘SimEau’.

The state-of-the-art of existing water treatment simulators was reviewed in the first deliverable (D5.4.1) of this project. The review concluded that OTTER, developed by WRc, Stimela, developed by TU-Delft/DHV and Metrex, developed by the University of Duisburg/IWW were the most appropriate existing platforms to act as a foundation for future development and integration.

The second deliverable (D5.4.2) described and discussed the methodology for integration. It was concluded that a prototype of the software would be developed, with WRc taking the lead in the development and technical descriptions of the user interface and numerical solvers, and TU-Delft working on the technical descriptions of process models to be incorporated in the prototype.

The third deliverable (D5.4.3) gave a description of the conceptual design of the new modelling platform and an overview of the selected treatment processes and determinands. It was concluded that the key features were that the treatment simulator would be free to use, easy to handle and that the simulator could be readily extended with new processes and determinands. Individual process models would be developed including their relevant water quality parameters. By calibrating and validating the models for different raw water qualities and with data from different water treatment plants (e.g. Waternet located in the Netherlands and Riga located in Latvia), the simulator should be robust and widely applicable. In addition, within the TECHNEAU programme, new models will be developed by Sintef, WRc, Delft University of Technology and the University of Riga, tackling the issues of NOM removal, biological treatment and membrane filtration.

The present report gives a description of the first version of the modelling framework. It includes a description of the use of the modelling framework itself, descriptions of basic models for influent, flow control, flow division and mixing, and descriptions of the first two process models: balancing tank and pellet softening. These process models are used to test the framework for different applications and degrees of complexity.

2 Using the modelling platform

2.1 Installation

Distribution of the program will be either through the Internet, downloading a single install file, or through a CD, where the install will comprise several files.

With the Internet version all that is required is to run the program, which will initiate the setup program. The CD version requires that the file SETUP.EXE be run.

Once the install program is running the user will have two choices – where to install the program, and what to call the program group on the Windows [Start] menu. At the end of the install the program is ready to be run.

2.2 Running the program

On starting the program the user sees a blank drawing board, with a list of available processes on the left-hand side (Figure 2.1). Note that the screen shots used represent a development version and the list of processes will change in content and appearance during the development of the program.

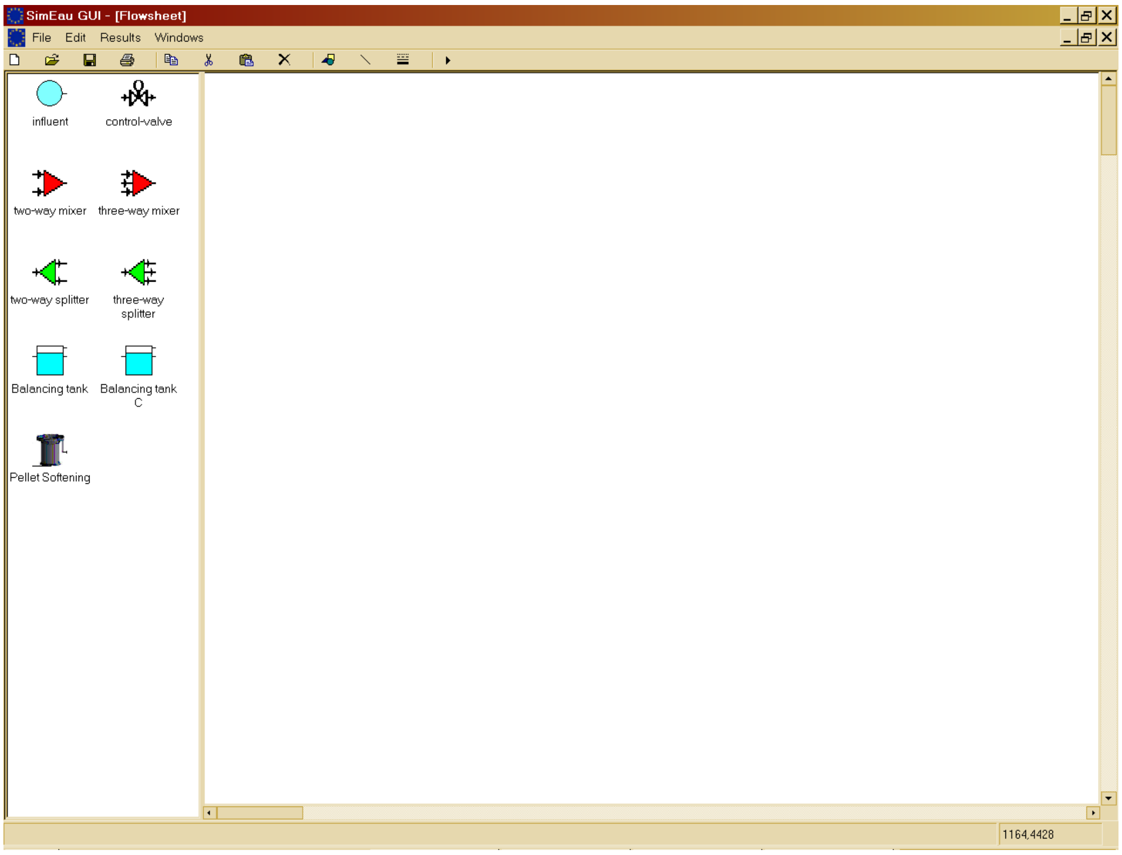


Figure 2.1 SimEau starting screen

The user can then select processes and place them on the drawing board, to construct a flowsheet of the intended water treatment works. Each process icon is clicked on and then, with the left mouse button held down, the mouse is moved to the drawing board and the mouse button released when the icon is placed on the drawing board (Figure 2.2).

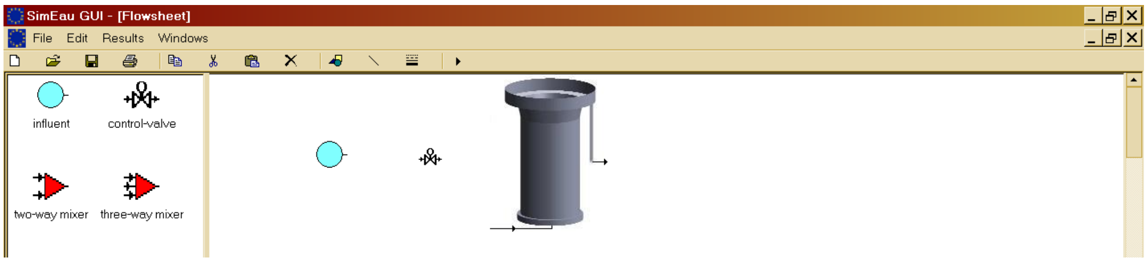


Figure 2.2 Drawing board with process models in position

The processes are connected by clicking on an outlet of a process, holding the left mouse button down, and moving the mouse to an inlet of another process, then releasing the button. The cursor will change from the default (usually, an arrow pointer) to a cross-hair when the mouse is over an outlet, and will change again (to a linked-chain) when the mouse is over an unconnected inlet.

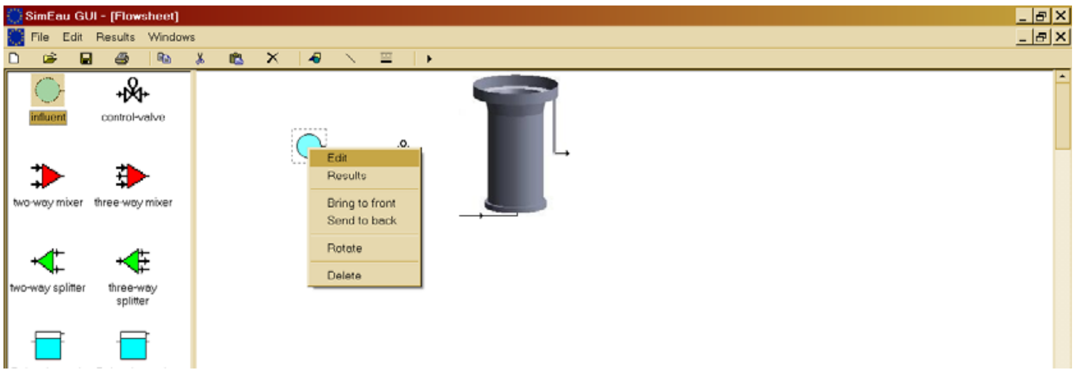
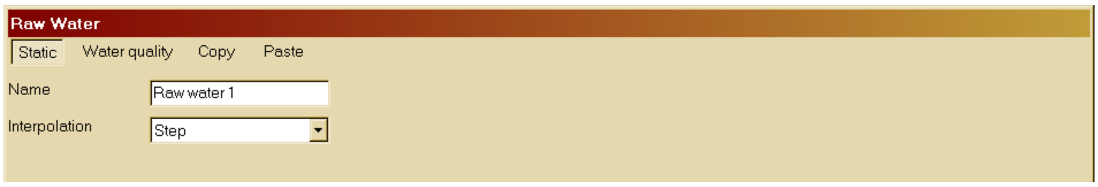


Figure 2.3 Selecting the properties of a process

At any time a right-mouse button click over a process will allow the user to edit the properties for that process (Figure 2.3). The properties will vary from process to process. For example, the raw water stream provides access to static data (Figure 2.4; a descriptive name, and whether the time-series is to be interpolated between measurements linearly or using a step function).


Raw Water

Static Water quality Copy Paste

Name: Raw water 1

Interpolation: Step

Figure 2.4 Static properties for raw water

As well as the static properties there are (for raw water) the water quality parameters (Figure 2.5). Temperature and pH are always defined. Flow is not specified, as flow is always set by using the control valve.

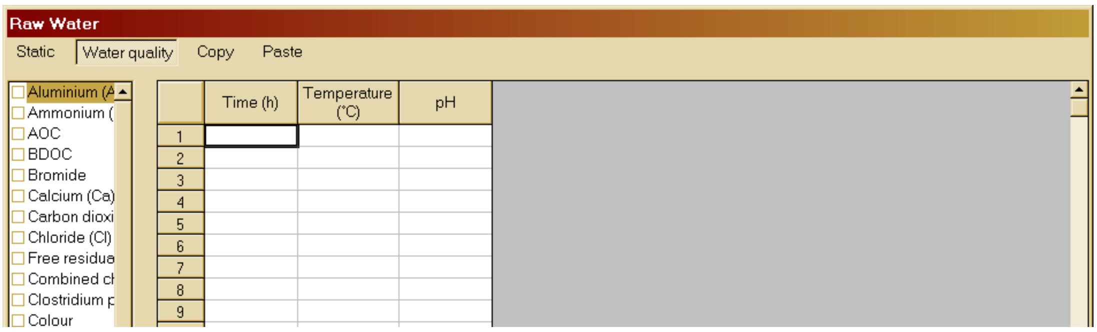


Figure 2.5 **Raw water quality – no determinands selected**

Determinands can be added or removed by selecting them from the check-list on the left-hand side of the raw water quality property table (Figure 2.6).

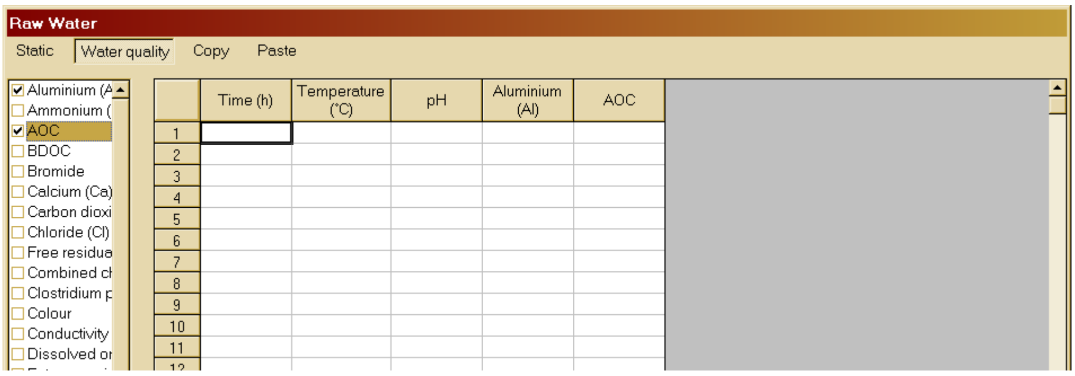
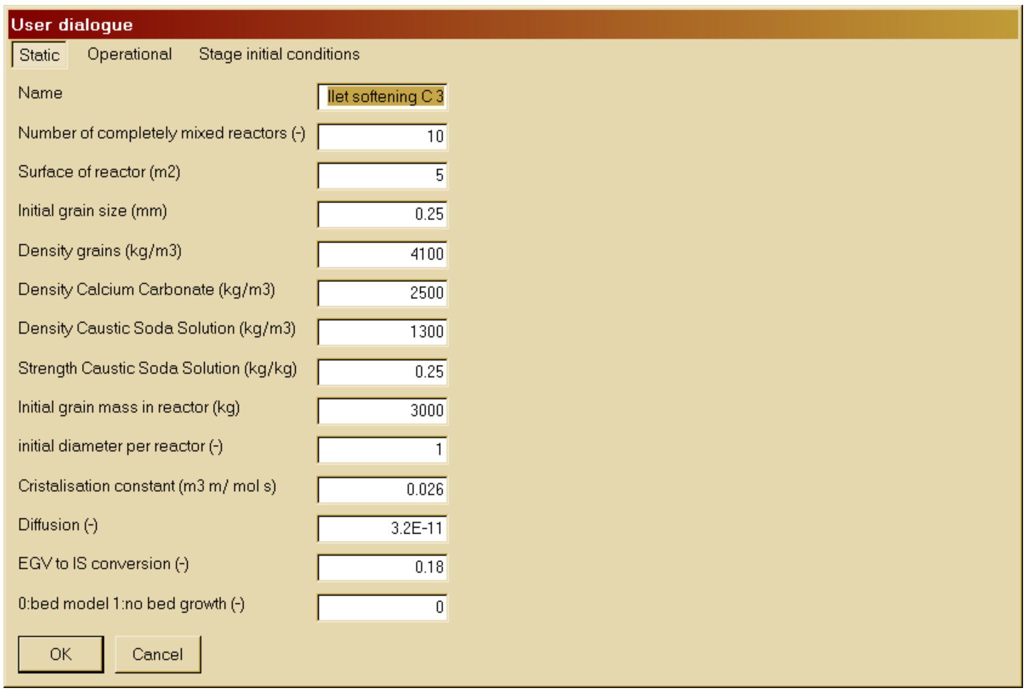


Figure 2.6 **Adding determinands to the raw water table**

Other processes have other properties – for example, the pellet softening model includes operational parameters and initial conditions (Figure 2.7). Other processes may include calibration parameters. Calibration parameters may be specified separately, or as part of the general static data.



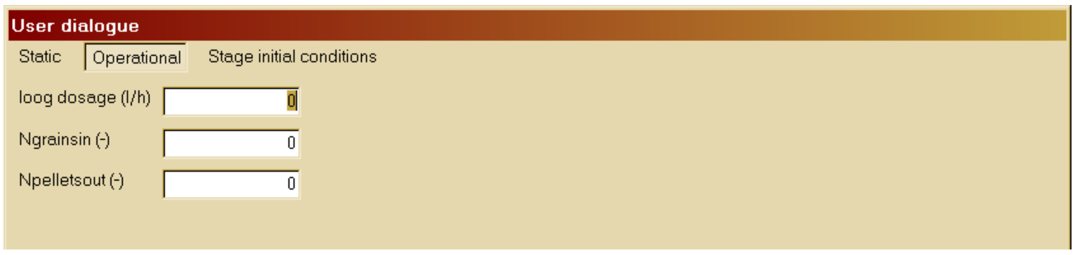
User dialogue

Static | Operational | Stage initial conditions

Name	llet softening C 3
Number of completely mixed reactors (-)	10
Surface of reactor (m2)	5
Initial grain size (mm)	0.25
Density grains (kg/m3)	4100
Density Calcium Carbonate (kg/m3)	2500
Density Caustic Soda Solution (kg/m3)	1300
Strength Caustic Soda Solution (kg/kg)	0.25
Initial grain mass in reactor (kg)	3000
initial diameter per reactor (-)	1
Cristalisation constant (m3 m/ mol s)	0.026
Diffusion (-)	3.2E-11
EGV to IS conversion (-)	0.18
0:bed model 1:no bed growth (-)	0

Figure 2.7 Pellet softener static properties

Operational properties (Figure 2.8)may be changed at any time during a simulation, by pausing the simulation and making the required changes. Some process models may allow a time profile to be specified for the operational parameters, so that there would be no need to pause a simulation to make such changes.



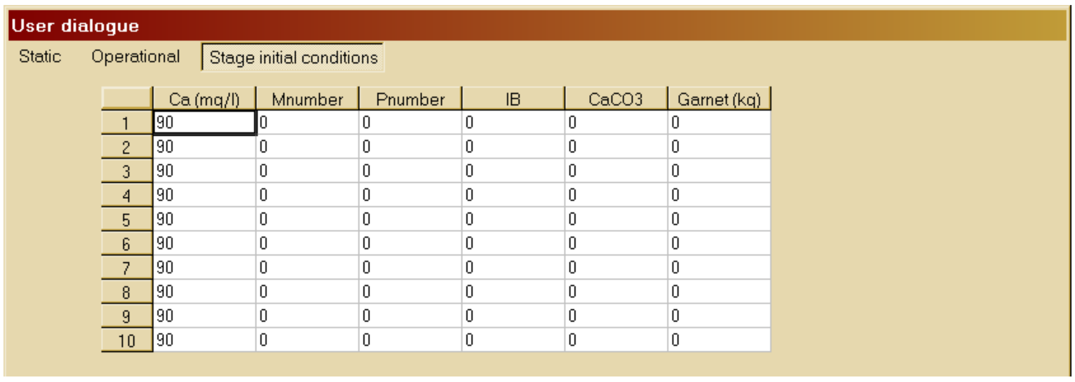
User dialogue

Static | Operational | Stage initial conditions

loog dosage (l/h)	0
Ngrainsin (-)	0
Npelletsout (-)	0

Figure 2.8 Pellet softener operational properties

The processes have default initial conditions specified (Figure 2.9), but these can be changed if necessary.



User dialogue

Static | Operational | Stage initial conditions

	Ca (mg/l)	Mnumber	Pnumber	IB	CaCO3	Garnet (kg)
1	90	0	0	0	0	0
2	90	0	0	0	0	0
3	90	0	0	0	0	0
4	90	0	0	0	0	0
5	90	0	0	0	0	0
6	90	0	0	0	0	0
7	90	0	0	0	0	0
8	90	0	0	0	0	0
9	90	0	0	0	0	0
10	90	0	0	0	0	0

Figure 2.9 Pellet softener initial conditions

Once a water works has been set up, and, ideally, saved, then the simulation can be run, using the 'play' button on the toolbar. A progress bar is displayed during the simulation. Once the simulation has been completed then results may be viewed, by right-clicking over the relevant process (Figure 2.10).

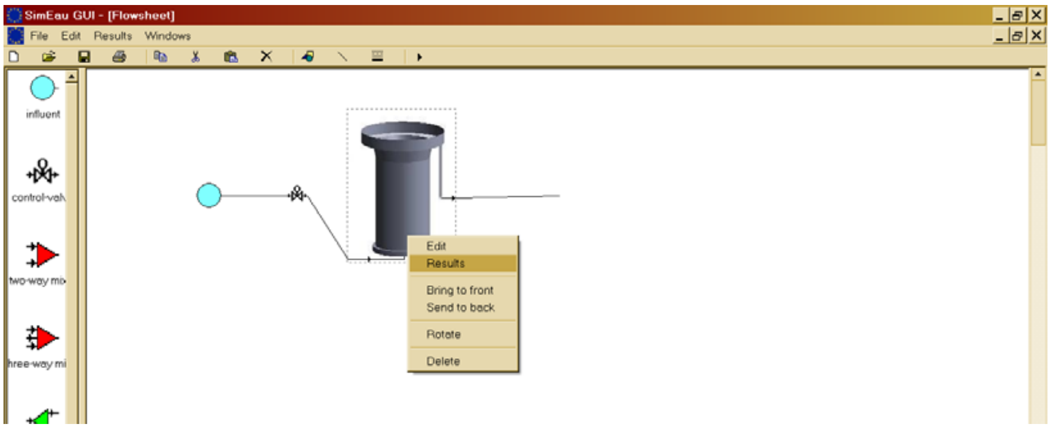
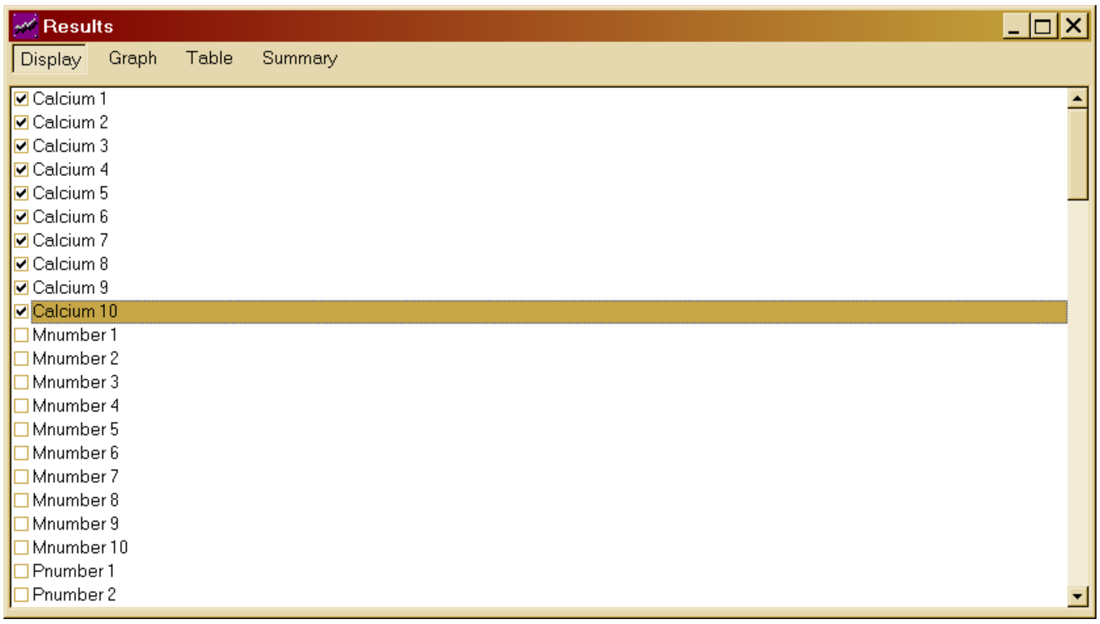


Figure 2.10 Viewing results

A check-list is displayed, where results of interest may be selected (Figure 2.11). At any time the user can switch between the check-list of what to display and the result views – graphical, tabular, or summary statistics, using the top menu bar.



☑ Calcium 1
☑ Calcium 2
☑ Calcium 3
☑ Calcium 4
☑ Calcium 5
☑ Calcium 6
☑ Calcium 7
☑ Calcium 8
☑ Calcium 9
☑ Calcium 10
☐ Mnumber 1
☐ Mnumber 2
☐ Mnumber 3
☐ Mnumber 4
☐ Mnumber 5
☐ Mnumber 6
☐ Mnumber 7
☐ Mnumber 8
☐ Mnumber 9
☐ Mnumber 10
☐ Pnumber 1
☐ Pnumber 2

Figure 2.11 Selecting results to display

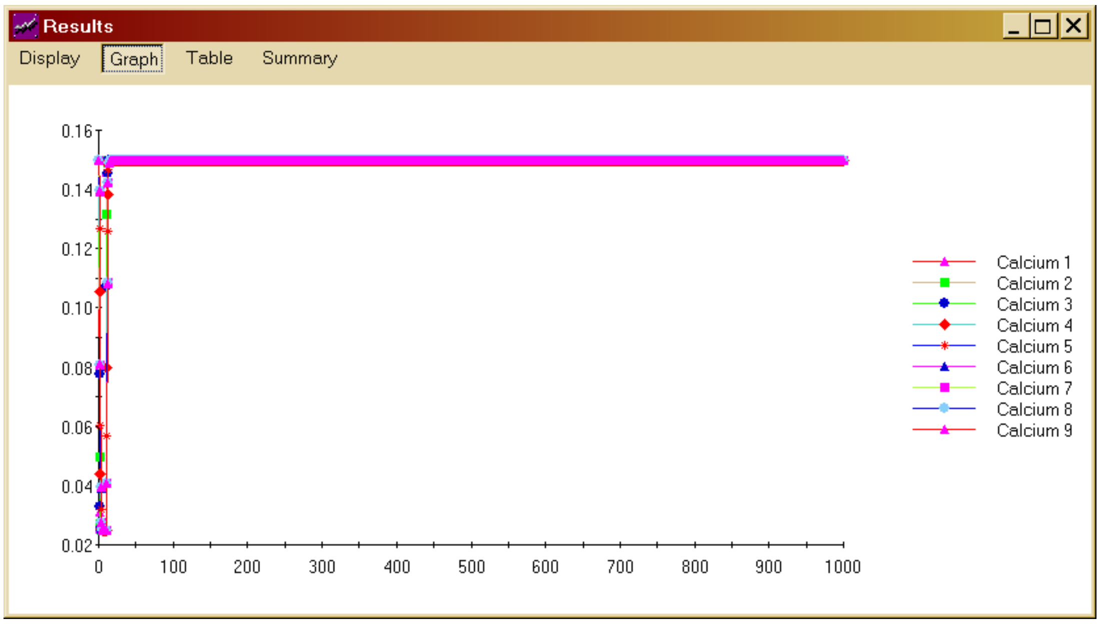
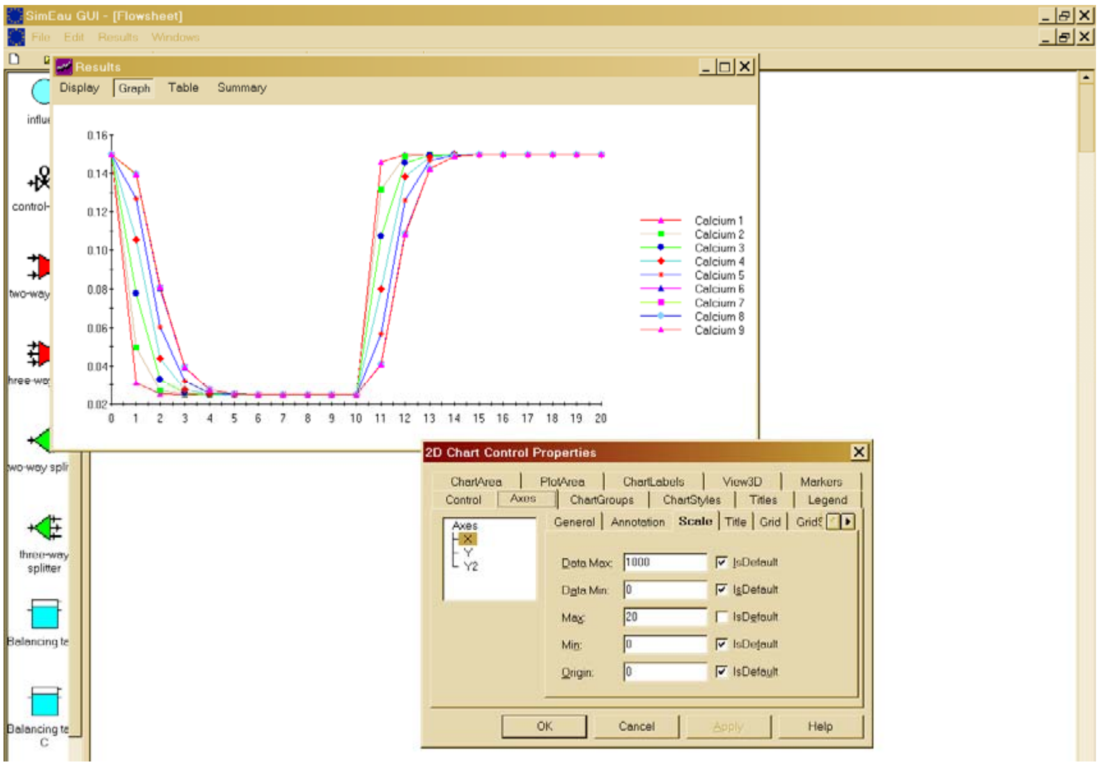


Figure 2.12 Graphical results view

The graphical view (Figure 2.12) can be modified by the user, right-clicking over the graph, when a graph properties control will be displayed (Figure 2.13). From here the X and Y axes scales, grid markings, series line colour, thickness, style, data points, all can be altered.

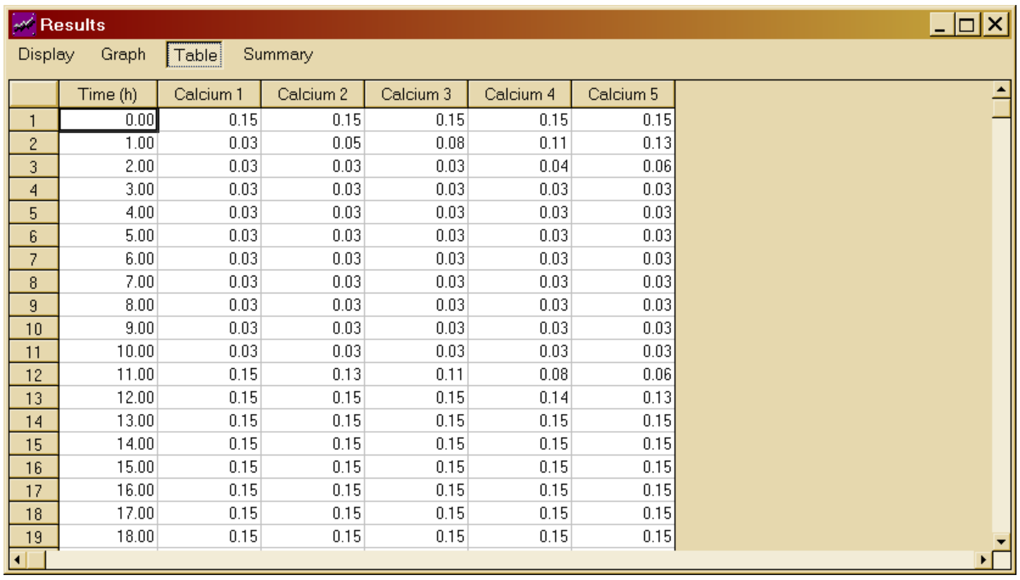


Data Max: 1000 ☑ IsDefault
Data Min: 0 ☑ IsDefault
Max: 20 ☐ IsDefault
Min: 0 ☑ IsDefault
Origin: 0 ☑ IsDefault

Figure 2.13 Example of zooming in on a graph

As well as using the graph properties control to zoom in on the graph this can be done by holding down the shift key, then using the mouse to mark out the area for zooming in. At any point the original graph can be returned to by pressing the [F3] key.

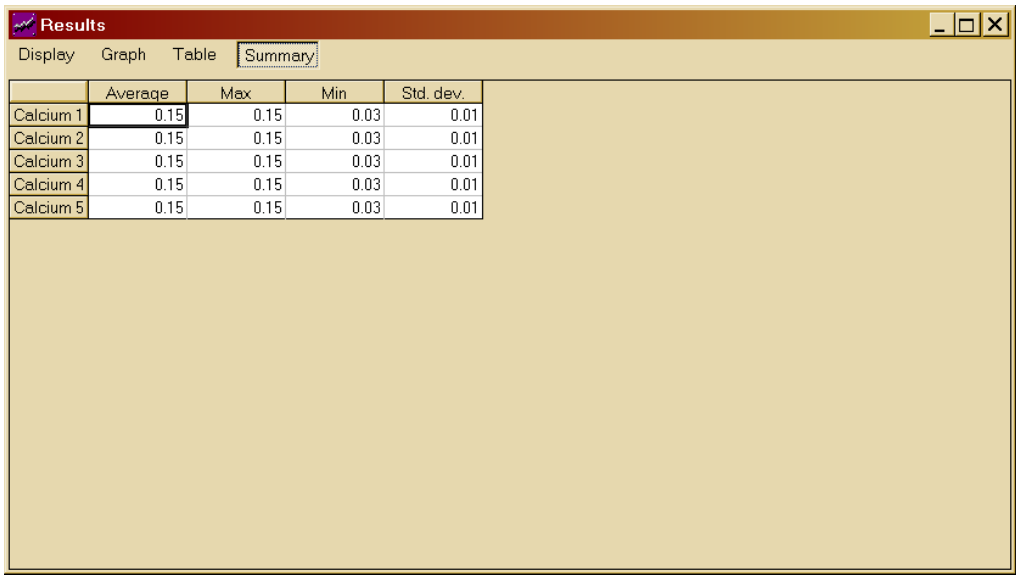
Tabular results are available, always displayed with the elapsed time in the first column (Figure 2.14).



	Time (h)	Calcium 1	Calcium 2	Calcium 3	Calcium 4	Calcium 5
1	0.00	0.15	0.15	0.15	0.15	0.15
2	1.00	0.03	0.05	0.08	0.11	0.13
3	2.00	0.03	0.03	0.03	0.04	0.06
4	3.00	0.03	0.03	0.03	0.03	0.03
5	4.00	0.03	0.03	0.03	0.03	0.03
6	5.00	0.03	0.03	0.03	0.03	0.03
7	6.00	0.03	0.03	0.03	0.03	0.03
8	7.00	0.03	0.03	0.03	0.03	0.03
9	8.00	0.03	0.03	0.03	0.03	0.03
10	9.00	0.03	0.03	0.03	0.03	0.03
11	10.00	0.03	0.03	0.03	0.03	0.03
12	11.00	0.15	0.13	0.11	0.08	0.06
13	12.00	0.15	0.15	0.15	0.14	0.13
14	13.00	0.15	0.15	0.15	0.15	0.15
15	14.00	0.15	0.15	0.15	0.15	0.15
16	15.00	0.15	0.15	0.15	0.15	0.15
17	16.00	0.15	0.15	0.15	0.15	0.15
18	17.00	0.15	0.15	0.15	0.15	0.15
19	18.00	0.15	0.15	0.15	0.15	0.15

Figure 2.14 Tabular view of results

Summary statistics are also available, returning the average, maximum and minimum values, and the standard deviation (Figure 2.15).



	Average	Max	Min	Std. dev.
Calcium 1	0.15	0.15	0.03	0.01
Calcium 2	0.15	0.15	0.03	0.01
Calcium 3	0.15	0.15	0.03	0.01
Calcium 4	0.15	0.15	0.03	0.01
Calcium 5	0.15	0.15	0.03	0.01

Figure 2.15 Summary statistics for results

3 Descriptions of basic models

This section provides a pseudo-code description of the following models:

- Raw water influent
- Flow control
- Flow division
- Flow mixing

The pseudo-code should enable most readers to readily understand what the process models are doing.

3.1 Raw water influent

The influent model has the raw water quality defined at two times, T1 and T2. In addition, there is a flag indicating if the water quality should be linearly interpolated or treated as step function. The algorithm is:

```
if interpolate = step then
  if time < T2 then
    stream(out) = stream(in)@T1
  else
    stream(out) = stream(in)@T2
  end if
else
  factor = (time – T1) / (T2 – T1)
  stream(out) = (1 – factor) * stream(in)@T1
              +      factor  * stream(in)@T2
end if
```

3.2 Flow control

The flow control is straightforward. It is passed the raw water quality, two values for the required flow at T1 and T2, and a flag indicating if the flow and water quality should be linearly interpolated or treated as a step function. The code is:

```
if interpolation = step then
  if time < T2 then
    stream(out).flow = flow1
  else
    stream(out).flow = flow2
  end if
else
  factor = (time – T1) / (T2 – T1)
  stream(out).flow = factor * flow2 + (1 – factor) * flow1
end if
```

3.3 Flow division

Flow dividers are the simplest possible class of model. There is no change in water quality, so all that is required is the specification of the relative flow split between the various outlets. The algorithm is:

```
for i = 1 to NumberOfOutlets
  stream(i).flow       := InletStream.flow x FlowFractionOut(i)
  stream(i).pH         := InletStream.pH
  stream(i).temperature: = InletStream.temperature
  for j = 1 to NumberofWaterQualityParameters
    stream(i).WaterQuality(j) := InletStream.WaterQuality(j)
  next j
next i
```

3.4 Flow mixing

Flow mixers are relatively simple, as all that is needed is a mass balance on the streams being mixed. The one complexity is pH, as this is nonlinear. The approach used here is to check to see if enough parameters are defined to use carbonate chemistry calculations. If basic carbonate information is available, but not ionic strength, then a simplified approach is used. If there is no carbonate information, then a simple unbuffered pH method is used. The algorithm is:

```
stream(out).flow = Σ stream(in).flow

                   Σ stream(in).flow x stream(in).WQ[1]
stream(out).WQ  = ----------------------------------
                        stream(out).flow

                  Σ stream(in).flow x stream(in).T[2]
stream(out).T   = --------------------------------
                        stream(out).flow
```

pH calculation:

```
if carbonate values then
[1] M         = 2*CO3 + HCO3 + OH - H3O
[2] P         = CO3 - CO2 + OH - H3O
[3] K1 * CO2  = HCO3 * H3O equilibrium [CO2]+[H2O]->[H+][HCO3-]
[4] K2 * HCO3 = CO3 * H3O equilibrium [HCO3-] -> [H+] [CO3--]
[5] Kw        = H3O * OH  equilibrium [H2O] -> [H+] [OH-]
Charge balance: 2 [CO3--] + [HCO3-] + [OH-] - [H+] = [Cations]
  This can be solved as
  Guess H
    OH = H / Kw
    HCO3 = (Cations + H - OH) / (1 + 2d0 * K2 / H)
    CO3  = K2 * HCO3 / H
    CO2  = HCO3 * H / K1
    H    = C - (CO2 + CO3 + HCO3)
        where C = the sum of CO2 + HCO3 + CO3 in the inlet
  Repeat until guessed and calculated H values agree

 The implementation uses a bisection search procedure to find the value of H

if ionic strength then
  Calculate Kw, K1, K2 with an activity coefficient correction
else
  Calculate Kw, K1, K2 without an activity correction
end if
else
  [1]  [H] [OH] = Kw                            -- equilibrium
  [2]  [H]in - [H] = [OH]in - [OH]              -- charge conservation
  [H]**2 - A [H] - Kw = 0 Solve for [H], then calculate pH
    Where   A = [H]in - [OH]in
end if
```

[1] WQ here is used as shorthand for Water Quality.
[2] T here is used as shorthand for temperature.

4 Description of the balancing tank model

The balancing tank is one of the simplest dynamic models, and was chosen for this reason as one of the exemplar models.

The basics of a balancing tank are that it has a time-varying influent, and usually has a constant outflow, with the volume within the tank varying depending upon whether the inflow is greater or less than the outflow.

The balancing tank is illustrated in Figure 4.1.

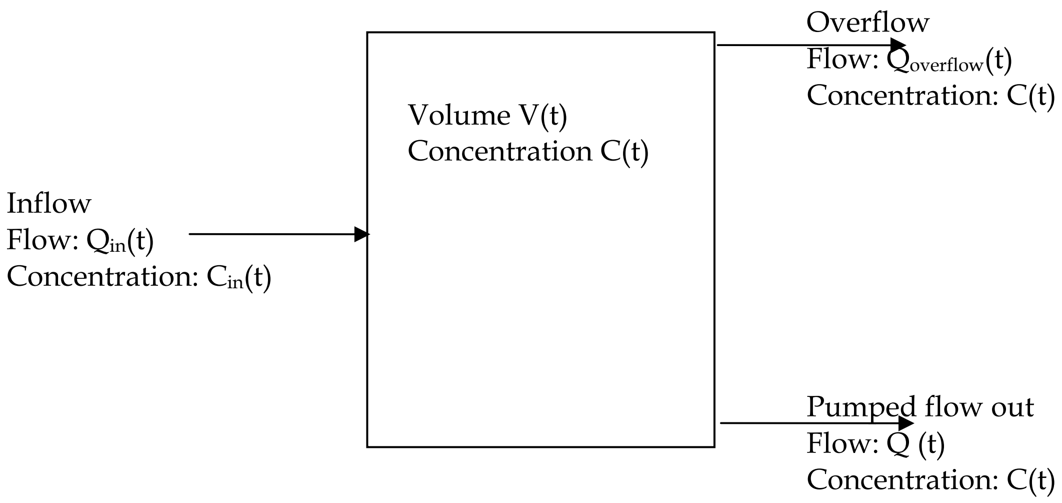


Figure 4.1 Balancing tank model

The balance around volume can be written as:

$$\frac{dV}{dt} = Q_{in} - Q - Q_{overflow}$$
$$if\ V > V_{\max}\ Q_{overflow} = Q_{in} - Q\ else\ Q_{overflow} = 0$$
$$if\ V < V_{\min}\ Q_{pump} = 0$$

The corresponding balance around the concentrations of determinands is:

$$\begin{aligned}\frac{dV \cdot C}{dt} &= V\frac{dC}{dt} + C\frac{dV}{dt} \\ &= V\frac{dC}{dt} + C\left(Q_{in} - Q - Q_{overflow}\right) \\ &= Q_{in}C_{in} - \left(Q + Q_{overflow}\right)C\end{aligned}$$

Or

$$V\frac{dC}{dt} = Q_{in}\left(C_{in} - C\right)$$

A pH model is provided using a simple unbuffered pH model:

$$\text{Charge balance}: [Cation] + [H] = [Anion] + [OH]$$
$$\text{Equilibrium}: [H][OH] = K_W$$
Solving these two equations gives
$$[Cation][H] + [H]^2 = [Anion][H] + K_W$$
[H] is then solved from the quadratic. pH is calculated as pH = -$\log_{10}$[H].

The code for this model is given in Appendix I, using both C and Fortran 90, to show how models can be implemented in a variety of languages.

5 Description of the softening process model

5.1 Model description

The softening process consists of a number of fluidised bed reactors with one bypass. The chemical reactions in the water take place in the reactor. The mixing process of reactor effluents and bypass water is modelled as instantaneous mixing (taking the calcium carbonic equilibrium into account) without any reaction kinetics.

Various models have been developed (Dijk & Wilms 1991, Bogt *et al.* 1992). These models are steady-state, used for design purposes. Van Schagen *et al.* (2008) developed a new, dynamic, model of the pellet softening process, which has been implemented first in Stimela and now in SimEau. This model is described in this section, with the detailed code listed in Appendix II. This model takes diffusion in the fluidised bed and new insights in the fluidised bed behaviour of pellet softening reactors into account.

The model is defined in the following three sections. In the first section, the calcium carbonic equilibrium, which determines the crystallisation in the reactor, is explained. The second section describes the fluidisation of the bed, which determines the available crystallisation surface in the reactor. Finally the crystallisation rate is modelled based on the crystallisation surface and the calcium carbonic equilibrium.

5.1.1 *Modelling the Calcium Carbonic Equilibrium*

The crystallisation of calcium carbonate is a shift in the equilibrium between the solid and soluble state of calcium carbonate (Wiechers *et al.* 1975):

$$\mathrm{Ca}^{2+} + \mathrm{CO}_3^{2-} \overset{K_s}{\leftrightarrows} \mathrm{CaCO}_3 \tag{1}$$

$$K_s = f^8[\mathrm{Ca}^{2+}][\mathrm{CO}_3^{2-}] \tag{2}$$

where the equilibrium constant K_s is an experimentally determined constant depending on the water temperature. The activity factor f is based on the ionic strength (IS) of the water and is given by (Schock 1984):

$$\log(f) = \frac{-0.5\sqrt{\mathrm{IS}}}{\sqrt{1000}+\sqrt{\mathrm{IS}}} + 0.00015\mathrm{IS} \tag{3}$$

To determine the carbonate concentration, the carbonic equilibrium must be taken into account. This is the balance between three carbonic fractions (CO_2, HCO_3^- and CO_3^{2-}). The ratio between the concentrations of these fractions has a strong relation with the pH. The following reactions describe the equilibrium:

$$\begin{aligned} CO_2 + 2H_2O &\overset{K_1}{\leftrightarrows} H_3O^+ + HCO_3^- \\ HCO_3^- + H_2O &\overset{K_2}{\leftrightarrows} H_3O^+ + CO_3^{2-} \\ H_3O^+ + OH^- &\overset{K_w}{\leftrightarrows} 2H_2O \end{aligned} \tag{4}$$

The reaction rates of these equilibria are high, and it is therefore assumed that the carbonic fractions are always in equilibrium. The conservative parameters m-alkalinity (*M*) and the p-alkalinity (*P*) are used to describe the equilibrium. The actual concentrations of the equilibrium can now be found by solving the following set of algebraic equations:

$$\begin{aligned} M &= 2[CO_3^{2-}] + [HCO_3^-] + [OH^-] - [H_3O^+] \\ P &= [CO_3^{2-}] - [CO_2] + [OH^-] - [H_3O^+] \\ K_1 &= f^2[HCO_3^-][H_3O^+][CO_2]^{-1} \\ K_2 &= f^4[CO_3^{2-}][H_3O^+][HCO_3^-]^{-1} \\ K_w &= f^2[H_3O^+][OH^-] \end{aligned} \tag{5}$$

where K_1, K_2 and K_w are experimentally well determined constants depending on the water temperature (Jacobsen & Langmuir 1974, Plummer & Busenberg 1982). Equation (5) is a set of five equations with seven unknown concentrations (M, P, CO_2, HCO_3^-, CO_3^{2-}, H_3O^+ and OH^-). Two concentrations must be known to determine the remaining ones. The H_3O^+ and the HCO_3^- are known concentrations in the raw water of the softening reactors. The H_3O^+ concentration is normally measured as pH:

$$pH = -\log(f[H_3O^+]) \tag{6}$$

It is now possible to determine the carbonate concentration throughout the crystallisation process. Based on the measured *pH* and HCO_3^- concentration, the m-alkalinity and the p-alkalinity are determined using Equation (5) and (6). The dosing of caustic soda causes an increase of the m-alkalinity and p-alkalinity due to the feed of OH^-. With the new m-alkalinity and p-alkalinity, the CO_3^{2-} concentration is determined. As soon as crystallisation takes place, carbonate (CO_3^{2-}) is removed from the water and m-alkalinity and p-alkalinity are accordingly lowered. Based on the lowered m-alkalinity and p-alkalinity the new carbonate concentration is determined.

Two parameters describe the super-saturation of calcium carbonate in water. The Saturation Index (*SI*) is defined as the pH offset at which the actual calcium concentration is in equilibrium with the carbonate:

$$\mathrm{SI} = \log\left(\frac{f^8[\mathrm{Ca}^{2+}][\mathrm{CO}_3^{2-}]}{K_s}\right) \tag{7}$$

and is a measure for the driving force in the crystallisation process. The TCCP (theoretical calcium carbonate crystallisation potential) is the amount of calcium in (mol/m^3) that should crystallise to obtain water in chemical equilibrium. The TCCP is a measure for the amount of calcium carbonate that can be formed in consecutive process steps. The two indices are strongly related, but are both used separately to quantify the performance of the crystallisation process.

5.1.2 *Modelling the fluidised bed*

The aim of the fluidised bed model is to describe the fluidisation of the bed and the transportation of the pellets through the bed. The fluidisation model is necessary to determine bed properties such as bed height, pressure drop and porosity depending on pellet size, water flow, pellet discharge speed and temperature. The porosity is used to model the crystallisation rate in the next paragraph.

The model is deduced by dividing the reactor in layers as shown in Figure 5.1. The water flow is schematised as a one-dimensional flow in upward direction, keeping the bed of pellets fluidised. In the case of a pellet discharge the pellets are transported in downward direction. The state variables of the fluidised bed model are the mass of the calcium carbonate m_c and the mass of the grains m_g.

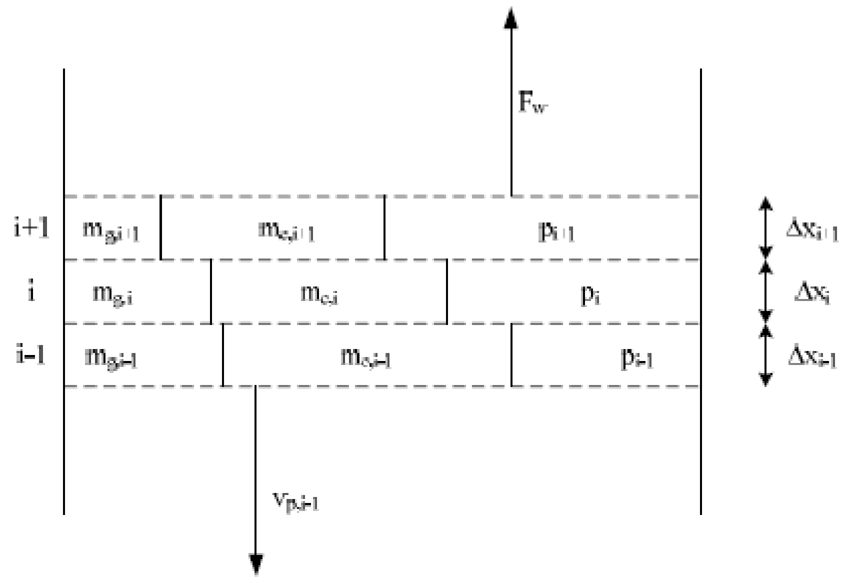


Figure 5.1 Modelled layers in the reactor

Each layer is divided into 3 sections: the volume of grains, the volume of calcium carbonate and the water volume determined by the porosity. The height of each layer is given by the porosity of the bed and the mass of pellets, consisting of grains and calcium carbonate:

$$\Delta x_i = \left(\frac{m_{g,i}}{\rho_g} + \frac{m_{c,i}}{\rho_c} \right) (1 - p_i)^{-1} A^{-1} \tag{8}$$

The porosity p_i in the ith layer is based on temperature, pellet diameter and water velocity and is accurately described by the Richardson-Zaki expansion formula (Richardson & Zaki 1954) (the indices i are dropped for readability):

$$p = \left(\frac{F_w}{A\ v_0} \right)^{\frac{1}{n}} \tag{9}$$

The terminal settling velocity v_0 and the exponent n are experimentally determined properties of a single particle. Richardson and Zaki found the following empirical relationship for the exponent n:

$$n = \begin{cases} 4.6 & \text{for } Re_0 < 0.2 \\ 4.4 Re_0^{-0.03} & \text{for } 0.2 \geq Re_0 < 1 \\ 4.4 Re_0^{-0.1} & \text{for } 1 \geq Re_0 < 500 \\ 2.4 & \text{for } Re_0 \geq 500 \end{cases} \tag{10}$$

For perfectly round, smooth and uniform particles, v_0 can be determined using the Newton-Stokes equation (Bird *et al.* 1960):

$$v_0^2 = \frac{4}{3} \frac{d_p (\rho_p - \rho_w) g}{C_{w2} \rho_w} \tag{11}$$

The drag coefficient C_{w2} is experimentally determined. The following drag coefficient for calcium carbonate pellets with a garnet seeding material was found (Schagen *et al.* 2006):

$$C_{w2} = \frac{24}{Re_0} (1 + 0.79 Re_0^{0.87}) \tag{12}$$

where the terminal settling Reynolds number is given by:

$$Re_0 = \frac{v_0 d_p}{\nu} \tag{13}$$

The average pellet diameter is determined using the mass of crystallised material. Assuming an even distribution of the mass over the grains, in the layer under consideration, the pellet diameter is calculated as follows:

$$d_p = d_g \sqrt[3]{1 + \frac{m_c}{m_g}\frac{\rho_g}{\rho_c}} \tag{14}$$

The pressure drop over each layer is given by the submerged weight of the fluidised pellets:

$$\Delta P = (\rho_p - \rho_w)(1-p)g\Delta x \tag{15}$$

where the density of the pellets is a function of the accumulated mass of the crystallised material and the mass of the grains:

$$\rho_p = (m_c + m_g)\left(\frac{m_c}{\rho_c} + \frac{m_g}{\rho_g}\right)^{-1} \tag{16}$$

The transportation of pellets is modelled as a transportation of grain material with the calcium carbonate attached. The velocity of transportation $v_{p,i}$ is given in kilograms of grain material per second. The transportation of the calcium carbonate part of the pellet is given by the ratio of calcium carbonate mass and grain mass. Since grain material can be accumulated in the reactor, the transportation of grains into the layer is different from the transportation of grains out of the layer. The accumulation of calcium carbonate caused by pellet transportation through the reactor is given by:

$$\frac{dm_{c,i}}{dt} = v_{p,i+1}\frac{m_{c,i+1}}{m_{g,i+1}} - v_{p,i}\frac{m_{c,i}}{m_{g,i}} \tag{17}$$

The increase of calcium carbonate, due to the crystallisation process, is determined by the crystallisation reaction and is given at the end of the next paragraph.

5.1.3 Modelling crystallisation

The aim of the crystallisation model is to describe the crystallisation of calcium carbonate in the bed and the transportation of the dissolved components through the bed.

The crystallisation model is deduced using layers of the reactor as given in Figure 5.1. The m-alkalinity, p-alkalinity and ionic strength in a layer determine carbonic equilibrium as described by Equation (5) and (3). The crystallisation rate of the equilibrium in Equation (1) is determined by the crystallisation kinetics *K*, the available crystallisation surface *S* and the super-saturation of calcium carbonate (Wiechers *et al.* 1975):

$$\mathcal{C} = \mathcal{K} \cdot \mathcal{S} \cdot \left([\mathrm{Ca}^{2+}][\mathrm{CO}_3^{2-}] - \frac{K_s}{f^8}\right) \tag{18}$$

The specific surface of the pellets is determined by the porosity *p* of the layer from Equation (9) and the diameter of the pellet in the layer from Equation (14).

$$S = \frac{6(1-p)}{d_p} \tag{19}$$

The crystallisation kinetics is modelled as a two-stage crystallisation process (Karpinski 1980). The first stage is the transportation of supersaturated water to the pellet surface (k_f), which depends on water flow and temperature. The second stage is the crystallisation of the supersaturated water on the pellet (k_T), which only depends on temperature.

$$\mathcal{K} = \frac{k_T \cdot k_f}{k_T + k_f} \tag{20}$$

The transportation of the supersaturated water to the surface of the pellets depends on the flow pattern of the water between the pellets (Budz *et al.* 1984). Based on the Reynolds number of the water flow in the bed Re_h and the Schmidt number Sc the Sherwood number Sh is given by the Froessling equation:

$$Re_h = \frac{2}{3}\frac{vd_p}{(1-p)\nu} \tag{21}$$

$$Sc = \frac{D_f}{\nu} \tag{22}$$

$$Sh = 0.66 Re_h^{0.5} Sc^{0.33} \tag{23}$$

The transportation coefficient in Equation (20) is given by:

$$k_f = \frac{Sh \cdot D_f}{d_p} \tag{24}$$

The temperature dependency of k_T is found by Wiechers *et al.* (1975) as:

$$k_T = 1.053^{(T-20)} \cdot k_{T20} \tag{25}$$

The change of m-alkalinity, p-alkalinity and ionic strength and calcium over time in one layer is now given by the combination of water flow through the reactor and crystallisation of calcium carbonate. Based on the mass balance over the layer, this is given by:

$$p_i A \Delta x_i \frac{d[\mathrm{Ca}^{2+}]_i}{dt} = F_w([\mathrm{Ca}^{2+}]_{i-1} - [\mathrm{Ca}^{2+}]_i) - A\Delta x_i \mathcal{C} \quad (26)$$

$$p_i A \Delta x_i \frac{dM_i}{dt} = F_w(M_{i-1} - M_i) - 2A\Delta x_i \mathcal{C} \quad (27)$$

$$p_i A \Delta x_i \frac{dP_i}{dt} = F_w(P_{i-1} - P_i) - A\Delta x_i \mathcal{C} \quad (28)$$

$$p_i A \Delta x_i \frac{dIS_i}{dt} = F_w(IS_{i-1} - IS_i) - 2A\Delta x_i \mathcal{C} \quad (29)$$

At the bottom of the reactor, where caustic soda is dosed, the concentration of the water flowing into the first section is given as:

$$M_0 = \frac{(M_{raw}F_w + [\mathrm{OH}^-]F_l)}{F_w + F_l} \quad (30)$$

$$P_0 = \frac{(P_{raw}F_w + [\mathrm{OH}^-]F_l)}{F_w + F_l} \quad (31)$$

$$IS_0 = \frac{(IS_{raw}F_w + 0.5[\mathrm{OH}^-]F_l)}{F_w + F_l} \quad (32)$$

Finally the increase of crystallised material in the layer is equal to the crystallised mass of calcium in mol times the molecular weight of calcium carbonate (M_c):

$$\frac{dm_{c,i}}{dt} = A\ dx_i\ M_c\ \mathcal{C} \quad (33)$$

5.2 Modelling example

To determine the parameters of the model, the model was calibrated at the pilot plant installation of the Weesperkarspel Treatment Plant of Waternet, the water cycle company of Amsterdam and surroundings. The calibrated model was validated using data from the full-scale installation of Weesperkarspel.

5.2.1 *Calibration at Weesperkarspel pilot plant*

The aim was to calibrate the crystallisation constant k_T and diffusion constant D_f in the model. The model with the calibrated constants minimise the Mean Squared Error (MSE) based on the measurements of total hardness, pH and m-alkalinity.

The model was calibrated with data from the Weesperkarspel pilot plant. The softening process in the pilot plant consists of two columns with a diameter of 31 cm and a height of 4.5 m. A regulated valve controls the flow between 4 m^3/h and 7 m^3/h. Caustic soda dosage is controlled between 0 and 2 l/h. To determine the fluidised bed status, the reactors are equipped with on-line measurements of water flow, water temperature, bed height, pressure drop over the total fluidised bed and pressure drop between 20 and 60 cm from the bottom of the reactor. To follow the crystallisation process, the turbidity, pH, total hardness, m-alkalinity and the conductivity are automatically measured using an online titration unit (Applikon ADI 2040) every 15 minutes.

Before calibration one reactor was operated at constant flow (6 m^3/h) and caustic soda dosage (1 l/h) for one month (February 2005). In this period the discharge of pellets was controlled using the pressure drop measurement at the bottom of the reactor (with a set point of 3.5 kPa), resulting in a constant pellet size at the bottom of the reactor. The bed height was kept constant at a height of 4 m by dosing garnet sand as seeding material.

The composition of the bed was constant after this run-in period. The state of the bed (described by $m_{c,i}$ and $m_{g,i}$) is identified using manual pressure drop measurements at 5 heights in the bed and the online bed height measurement at 3 different flows. The identification was performed with a different number of layers, to determine the influence on the prediction of the pressure drop and level measurements. The best estimate minimises the MSE using a nonlinear optimisation technique. The MSE is used for all calibration and validation experiments and is generically given by:

$$MSE = \frac{1}{N}\sum_{j=1}^{N}\sqrt{\frac{1}{N_j}\sum_{i=1}^{N_j}\left(\frac{y_{model,j,i}-y_{data,j,i}}{y_{data,j,i}}\right)^2} \qquad (34)$$

where y are the N outputs from the model and the measurement data and N_j are the number of samples for the jth output.

In the calibration experiment the water flow through the reactor and caustic soda flow were changed every 20 minutes. The water flow was varied between 4 m^3/h and 7 m^3/h and the caustic soda was varied between 0.5 and 1.5 l/h. After 20 minutes the water quality parameters were measured automatically. In this manner 40 different combinations of water flow and caustic soda dosage settings were performed in a random order. This procedure was repeated three times.

5.2.2 *Validation at Weesperkarspel full-scale plant*

The model for the pellet softening process was first validated with data from the eight softening reactors of WTP Weesperkarspel of Waternet. The Weesperkarspel treatment plant uses lake water with relatively high organic concentrations as source water. Before softening the water is treated with ozone.

The reactors operate at a variable flow velocity of 60-100 m/h to keep the ratio between bypass and reactor flow constant for different total flows. The reactor height is 4.5 meter, the seeding material is garnet sand and the dosage is caustic soda. The pellet discharge is based on the total pressure drop and the garnet sand dosage is based on the amount of discharged pellets.

The aim was to validate the dynamic output of the model. Therefore data from the full-scale plant was selected with relatively large variations in flow and caustic soda. For a five day period (15-20 October 2005) the dynamic simulation was performed for all 8 reactors. The inputs for the simulation model were the on-line measured temperature, water flow and caustic soda flow. The quality data (pH, bicarbonate and conductivity) were assumed constant based on laboratory values from that particular week. The state of the bed was deduced from the sieve analyses from the corresponding day.

The outputs of the model were compared to the on-line measured pH of the full-scale plant. The laboratory measurement of calcium was only performed once a day. This measurement was also compared to the simulated value. The on-line pH measurement suffered from static offset due to drift of the measurement device. For Reactor 6, the pH measurement was recalibrated on October 18th, after which the pH measurement was close to the simulated value (see Figure 5.2)

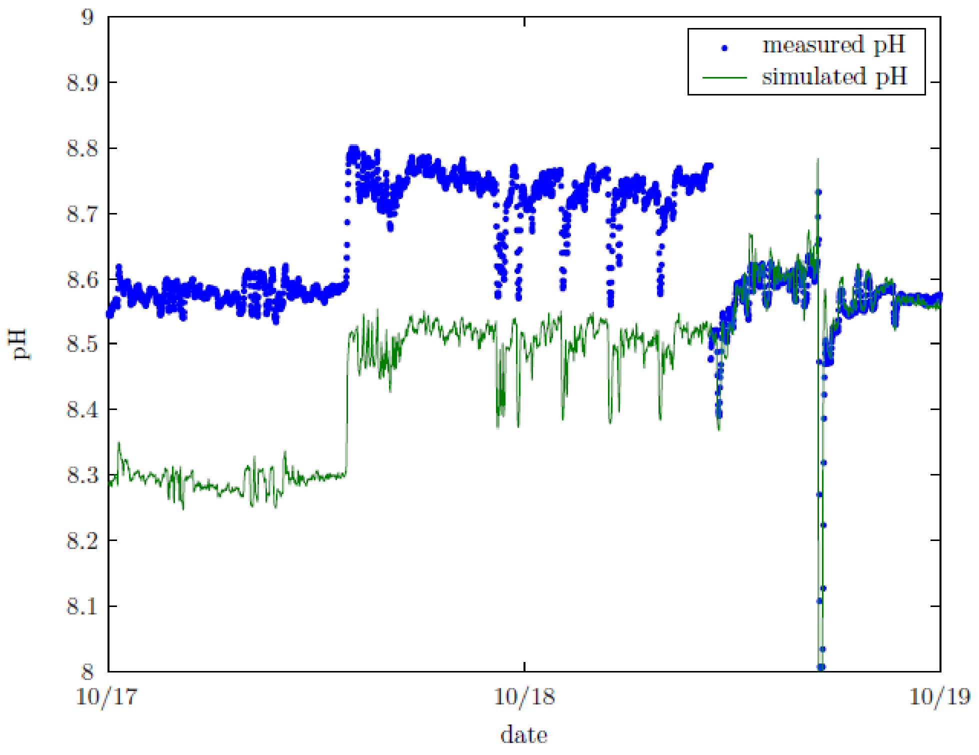


Figure 5.2 Measured and simulated pH for Reactor 6 of Weesperkarspel full-scale plant

6 Conclusions

A first version of the European Water Treatment Simulator ('SimEau') has been developed. The first version includes basic models for influent and flow and two process models: balancing tank and pellet softening. The framework was developed by WRc and tested by TU-Delft. Descriptions of the models are given in the present deliverable (D5.4.4).

In the coming period, more process models will be incorporated in the modelling framework in order to be able to test the models in TECHNEAU case studies such as planned for Riga, Lisbon and Amsterdam.

The performance of the modelling framework will continue to be optimised, leading to a robust tool for use in the water industry.

7 References

D5.4.1, L Rietveld and J Dudley, 2006. "Models for drinking water: State of the art", available at www.techneau.eu.

D5.4.2, L Rietveld and J Dudley, 2006. "Models for drinking water: Methodology for integration", available at www.techneau.eu.

D5.4.3, L Rietveld, P Ross, G Dillon and J Dudley, 2007. "Conceptual design modelling framework", available at www.techneau.eu.

Bird, R., Stewart, W. & Lightfoot, E., 1960. Transport Phenomena. Wiley, New York.

ter Bogt, L., Roos, P., Rosens, E., van Veen, A. & Veenendaal, M., 1992. Modellering onthard-ing in korrelreactoren. Technical report, TU Twente, Enschede, The Netherlands. CR II-verslag.

Budz, J., Karpanski, P. & Nuruc, Z., 1984. Influence of hydrodynamics on crystal growth and dissolution in a fluidized bed. AIChE J. 30(5), 710–717.

van Dijk, J. & Wilms, D., 1991. Water treatment without waste material - fundamentals and state of the art of pellet softening. J. Wat. Suppl.: Res. Technol.-AQUA 40(5), 263–280.

Jacobsen, R. & Langmuir, D., 1974. Dissociation constants of calcite and $CaHCO_3$ from 0 to 50 °C. Geochim. Cosmochim. Acta 38, 301–308.

Karpinski, P., 1980. Crystallization as a mass transfer phenomenon. Chem. Eng. Sci. 35, 2321–2324.

Plummer, L. & Busenberg, E., 1982. The solubilities of calcite, aragonite and vaterite in CO_2-H_2O solutions between 0 and 90 °C, and an evaluation of the aqueous model for the system $CaCO_3$-CO_2-H_2O. Geochim. Cosmochim. Acta 46, 1011–1040.

Richardson, J. F. & Zaki, W. N., 1954. Sedimentation and fluidisation: Part I. Trans. Inst. Chem. Eng. 32, 35–53.

van Schagen, K., Rietveld, L., Babuska, R. & Kramer, O., 2007. Model-based operational constraints for fluidised bed crystallisation. Wat. Res. doi:10.1016/j.watres.2007.07.019

Schock, M., 1984. Temperature and ionic strength correction to the Langelier Index-revisited. J. Am. Wat. Wks. Assoc. 76, 72–76.

Wiechers, H., Sturrock, P. & Marais, G., 1975. Calcium carbonate crystallization kinetics. Wat. Res. 9, 835–845.

APPENDIX I: Adding New Process Models

Extending SimEau

SimEau has been developed with the intention of being open framework software in the manner of Stimela. New process models can readily be added to SimEau. Each new process model requires a minimum of three files, and possibly more. These files are:

- A bitmap to represent the process. If the process image can be rotated – for example, to support inlets on the left or right – then one bitmap per rotated image is required.
- A DLL (dynamic link library – a means of providing libraries that can be accessed by other programs) containing the model. Each DLL must contain at least two subroutines, one that calculates the differential equations for the process, and the other then sets the output streams for the process.
- A configuration file, providing the following information:

- The name of the DLL file
- The name of the entry in the DLL for the differential equations
- The name of the entry in the DLL for the stream assignment equations
- The name and location of the image bitmaps
- Information for each bitmap to locate the inlets and outlets
- Information for the data required by the model – static, operational, etc.

This section provides three examples. The first two use the same process model, a balancing tank, and show the use of C++ and Fortran. (The Fortran compiler uses some specific extensions provided in the Digital/Compaq/Intel Visual Fortran range; all Windows-based compilers will offer an equivalent method, but the syntax may vary.) In each case there are two main files, a configuration file, and the program source. In addition, the program source may use some subsidiary files.

More details of the configuration file and coding requirements are given in Appendix IV.

Simple example #1: Balancing tank model using C

<u>Configuration file</u>

```
<!-- Balancing tank data file -->    Comments are inside <!—and --> markers
<model>                              <model> marks the start of the file
  <name = "Balancing tank C" />      A descriptive name
  <dll = "BalancingTankC.dll" />     The name of the DLL that has the model
  <diff-entry = "BalancingTank" />   The entry point for differential equations
  <alg-entry = "BalancingTankS" />   The entry point for stream assignment
```

```
<ImageData>                        Define the icons to be used
  <Symbol-Count = 2 />             Two icons; left-right flow, and right-left
  <Inlet-Name = ["Inlet"] />       One inlet
  <Outlet-Name = ["Outlet", "Overflow"] />Two outlets

  <Icon>                                  First icon
    <Filename = "models\balancing tank\bal.bmp" />       File to be used
    <Inlet-Loc = [0, 15] />                              Inlet location
    <Outlet-Loc = [40, 15| 40, 5] />                     Outlet locations
  </Icon>

  <Icon>                                  Second icon
    <Filename = "models\balancing tank\bal-left.bmp" />
    <Inlet-Loc = [40, 15] />
    <Outlet-Loc = [0, 15| 0, 5] />
  </Icon>
</ImageData>

<Static-Data>                      Static data
  <Parameter>                      <Parameter> marks the start of a data item
    <Name = "Name" />              Internal name
    <Legend = "Name" />            Display name
    <Type = STRING />              Type[3]
    <Default = "Balancing tank C" />
  </Parameter>

  <Parameter>
    <Name = MAXVOL />                             Internal name
    <Legend = "Maximum volume (m3)" />            Display name
    <Type = REAL />
    <Range>0; 1E10</range>  For REAL and INTEGER a range can be specified
    <Default = 100 />
  </Parameter>

  <Parameter>
    <NAME = MINVOL />
    <Legend = "Minimum volume (m3)" />
    <Type = REAL />
    <Range>0; 1E10</range>
    <Constraint = LESS_THAN(MAXVOL) />    Introduce a constraint
    <Default = 0 />
  </Parameter>
</Static-Data>

<Operational-Data>
  <Parameter>
    <Name = OUTFLOW />
    <Legend = "Outflow (m3/h)" />
    <Type = REAL />
    <Range>0; 1E10</range>
    <Default = 10 />
  </Parameter>
</Operational-Data>

<Initial-Conditions>               Specify initial conditions. This will
  <Parameter>                      also be the order expected of the
    <Name = VOLUME />              differential equations
    <Legend = "Volume (m3)" />
    <Type = REAL />
    <Default = 50 />
    <SF = 3 />                     Number of significant figures when
  </Parameter>                     displayed

  <Parameter>
    <Name = AGE />
    <Legend = "Hydraulic age (h)" />
    <Type = REAL />
    <Default = 0 />
    <SF = 3 />
  </Parameter>
```

[3] Permitted types are **STRING, REAL, INTEGER, LIST, DATE, CHECK-BOX,** and **GRID**

```
    <Parameter>
      <Name = T />
      <Legend = "Temperature (deg C)" />
      <Type = REAL />
      <Default = 25 />
      <SF = 3 />
    </Parameter>

    <Parameter>
      <Name = CATION />
      <Legend = "Cations (M * 1E8)" />
      <Type = REAL />
      <Default = 0 />
      <SF = 3 />
    </Parameter>

    <Parameter>
      <Name = ANION />
      <Legend = "Anions (M * 1E8)" />
      <Type = REAL />
      <Default = 0 />
      <SF = 3 />
    </Parameter>

    <Parameter>
      <Name = "H" />
      <Legend = "Hydrogen (Mx1E8)" />
      <Type = REAL />
      <Default = 0 />
      <SF = 3 />
    </Parameter>

    <Parameter>
      <USE OTHER_WATER_QUALITY />    Specify that other WQ determinands are
    </Parameter>                     to be added to the model, as well as those
  </Initial-Conditions>              known about & declared above

  <WaterQualityParameters>           Specify what determinands are expected to
    <determinand>                    be defined in the raw water quality.
      <wq-code = "CATION" />                 Internal name
      <wq-legend = "Cations (M)" />          Display name
      <wq-units = "M" />
    </determinand>

    <determinand>
      <wq-code = "ANION" />
      <wq-legend = "Anions (M)" />
      <wq-units = "M" />
    </determinand>
  </WaterQualityParameters>

</model>
```

<u>*Source code*</u>

BalancingTank.h

This is a support file that contains information used in the file BalancingTank.cpp.

```
#include "SimdeauTypes.h"     Include a header file, Simdeau.h. Defined below
#include "math.h"             Include math.h – a C++ standard file
```

To simplify coding we define EXPORT as a synonym for extern "C" __declspec(dllexport), and similarly for CORE_API.

extern "C" ensures that the subroutine names are kept in a simple format – C++ compilers would otherwise add additional characters to the subroutine names to indicate the number and type of input and output parameters.

__declspec(dllexport) marks that the routine will be compiled to a DLL and made publically visible.

```
#define EXPORT     extern "C" __declspec(dllexport)
#define CORE_API   extern "C" __declspec(dllimport)
```

The following two routines are provided by an external DLL.
The DLL is the core DLL provided by SimEau, and a library file is included with the DLL. That library file should be used when linking the model DLL, to ensure that the library routines can be found.

```
CORE_API long NumberOfDeterminands();
CORE_API long _stdcall GetIndex(char * Code, long LenCode);
```

The two routines that must be made available, one for differential equations, the other for stream assignment.

```
EXPORT void BalancingTank(double * Time, long * Stages,
                          SDProcess *Descriptor, SDStream *S,
                          double *y, double **yStage,
                          double *x, double **xStage,
                          double *dy, double **dyStage,
                          double *Results, double **StageResults,
                          double *Operation, double **StageOperation,
                          double *ModelData);

EXPORT void BalancingTankS(double * Time, long * Stages,
                           SDProcess *Descriptor, SDStream *S,
                           double *y, double **yStage,
                           double *x, double **xStage,
                           double *Results, double **StageResults,
                           double *Operation, double **StageOperation,
                           double *ModelData);
```

SimEau.h

This file defines various data structures used by the program.

```
/* Simdeau Type definition file */

#ifndef SIMDEAUTYPE_DEFINED
#define SIMDEAUTYPE_DEFINED          This is a common C/C++ method to ensure
                                     that DEFINE statements are not duplicated.
#define MAX_DETERMINANDS 500
#define MAX_INLETS 3
#define MAX_OUTLETS 3

struct SDStream {                            The definition of a STREAM
 double Flow;                   // Water flow  [m3/h]
 double Temperature;            // Temperature [C]
 double pH;                     // pH          [-]
 double Value[MAX_DETERMINANDS]; // other determinands
};

struct SDProcess {                   The definition of a PROCESS descriptor
  long ModelID;
  long ModelIndex;
  long InStream[MAX_INLETS];         Necessary to locate inlet streams
  long OutStream[MAX_OUTLETS];       Necessary to locate outlet streams
  long Stages;                       The number of stages defined
  long y;                            This and all below is used by the core
  long yStage;                       program, but access should not normally be
  long x;                            needed by the user
  long xStage;
  long Results;
  long StageResults;
  long Operation;
  long StageOperation;
  long ModelData;
};
#endif
```

BalancingTank.cpp

This is the model source code.

```
// BalancingTank.cpp : Defines the entry point for the DLL application.
//

#include "stdafx.h"
#include "BalancingTank.h"

double Kw(double T) {return 1e-14;}         Define water equilibrium constant

long ANION, CATION;   Will be set to the location in the STREAM water
                              quality array for anions & cations
int *Loc;                     Will be set to the location of water quality
                              determinands that are NOT anion/cation
bool FirstCall = true;

This routine defines differential equations
EXPORT void BalancingTank(double * Time, long * Stages,
                                SDProcess *Descriptor, SDStream *S,
                                double *y,         double **yStage,
                                double *x,         double **xStage,
                                double *dy,        double **dyStage,
                                double *Results,   double **StageResults,
                                double *Operation, double **StageOperation,
                                double *ModelData)
{
  double MinVol, MaxVol, PumpFlow;
  double Volume, Qoverflow, Qpump, Qin;
  double H, OH, An, Cat, T, K;
  int i, in;
  int n;

  // if n == 2 then 'CATION' and 'ANION' should have been defined
  n = (int) NumberOfDeterminands();

  MaxVol = ModelData[0];
  MinVol = ModelData[1];
  PumpFlow = Operation[0];

Fortran and Visual Basic start arrays at 1. C/C++ start arrays at 0. Therefore,
for many array index calculations C/C++ need to subtract 1 from the array
locations as stored in the data structures.
  in = (Descriptor->InStream[0]) - 1;
  Qin = S[in].Flow;

  Volume = y[0];
  Qoverflow = 0.0;
  Qpump = PumpFlow;
No outflow permitted if the volume is below the minimum.
  if (Volume < MinVol) Qpump     = 0.0;
Overflow only if the volume would exceeds the maximum permitted. The overflow
rate is then the difference between the flow in and the flow being pumped out.
  if (Volume >= MaxVol) Qoverflow = max(0.0, Qin - Qpump);

Identified differential equations
// Y( 0) = Volume
//    1  = Hydraulic age
//    2  = Temperature
//    3  = Cations
//    4  = Anions
//    5  = [H+]
Possible additional equations from other water quality determinands
//    6 .. NumberOfComponents - 2 = Water quality parameters
  dy[0] = Qin - Qpump - Qoverflow;
  if (Volume > 0.0)
  {
    dy[1] = 1.0 - Qin * y[1] / Volume;
    dy[2] = Qin * (S[in].Temperature - y[2]) / Volume;
    T   = y[2];
    K   = Kw(T);
    H   = pow(10.0 , (-S[in].pH));
    OH  = K / H;
    An  = S[in].Value[ANION];
    Cat = S[in].Value[CATION];
// Typical concentrations are of the order 1E-6 - 1E-8
// Multiply by 1E10 to attain some scaling
    dy[3] = Qin * (1e10* Cat - y[3]) / Volume;
    dy[4] = Qin * (1e10* An  - y[4]) / Volume;
    H     = y[5];
    dy[5] = (dy[4] - dy[5]) * pow(H,2) / (pow(H,2) + K*1e20);

// Now do other components
    for (i=2;i<n;i++)
         dy[4+i] = Qin * (S[in].Value[Loc[i]] - y[4+i]) / Volume;
  }
  else
  {
    for (i=1;i<n+6;i++)
      dy[i] = 0.0;
  }
}
```

This routine assigns the contents of the streams leaving the balancing tank

```
EXPORT void BalancingTankS(double * Time, long * Stages,
                           SDProcess *Descriptor, SDStream *S,
                                  double *y,         double **yStage,
                                  double *x,         double **xStage,
                                  double *Results,   double **StageResults,
                                  double *Operation, double **StageOperation,
                                  double *ModelData)
{
int Pumpedflow, Overflow, i, In;
double Volume, MinVol, MaxVol;
double PumpFlow, Qin;
int j, n;
long Len;

  n = (int) NumberOfDeterminands();
```

Aim to save the locations of the water quality determinands

```
  if (FirstCall || (*Time == 0.)) {
    if (FirstCall == false) delete [] Loc;
    FirstCall = false;
    Loc = new int [n];Allocate the memory required
    if (n != 0) {
      Len = 7;
       CATION = GetIndex("CATION", Len);            Get the location of CATION
       Subtract 1: the difference between Fortran and C/C++ indexing
       CATION = CATION - 1;  // Len("CATION") = 6; + 1 for CHAR(0) at end
       Len = 6;
       ANION  = GetIndex("ANION",  Len);            Get the location of ANION
       ANION = ANION - 1;            // Len("ANION") = 5
       Create an array that contains all other water quality indices, except
       for the values for anion & cation.
       j = - 1
       for (i = 0; i < n; i++) {
         if ((i != CATION) && (i != ANION)) {j = j + 1; Loc[j] = i;}
       }
    }
  }
  In         = (Descriptor->InStream[0]) - 1;
  Pumpedflow = (Descriptor->OutStream[0]) - 1;
  Overflow   = (Descriptor->OutStream[1]) - 1;
  Volume     = y[0];

  MaxVol = ModelData[0];
  MinVol = ModelData[1];
  PumpFlow = Operation[0];
  Qin = S[In].Flow;

  if (Volume < MinVol)
  {
    S[Pumpedflow].Flow = 0;
    S[Overflow].Flow = 0;
  }
  else
  {
    S[Pumpedflow].Flow = PumpFlow;
    S[Pumpedflow].Temperature = y[2];
    S[Pumpedflow].pH   = -log10(y[5] / 1.0e10);
    S[Pumpedflow].Value[CATION] = y[3] / 1.0e10;
    S[Pumpedflow].Value[ANION]  = y[4] / 1.0e10;
    for (i=2;i<n;i++) S[Pumpedflow].Value[Loc[i]] = y[4+i];

    if (Volume < MaxVol)
      S[Overflow].Flow = 0;
    else
       {
      S[Overflow] = S[Pumpedflow];
      S[Overflow].Flow = Qin - PumpFlow;
    }
  }
}
```

Simple example #2: Balancing tank model using Fortran

Here is the same balancing tank model, using Fortran.

Define data of interest. The SAVE attribute ensures that the values will be retained from one call to the next.

```
module SharedData
  integer, save                :: n, nBound
  integer, allocatable, save    :: ComponentLoc(:)
end module

subroutine Setup
USE SharedData
implicit none
```

The !DEC$ statements are compiler directives for the Digital/Compaq/Intel Visual Fortran range of compilers. They may be understood by other compilers.

DLLIMPORT specifies that the routine will be provided by another DLL.
STDCALL specifies how the routines will interface in calling mechanism with the other DLL.
REFERENCE specifies that all parameters will be passed by reference, not by value.
ALIAS indicates the exact name that we wish to use, and overrides any compiler settings on how names should be manipulated before being exported/imported from DLLs.

```
  !DEC$ ATTRIBUTES DLLIMPORT, STDCALL, REFERENCE:: NumberOfDeterminands
  !DEC$ ATTRIBUTES ALIAS: '_NumberOfDeterminands':: NumberOfDeterminands
  integer, external            :: NumberOfDeterminands

  !DEC$ ATTRIBUTES DLLIMPORT,  STDCALL, REFERENCE:: GetIndexF
  !DEC$ ATTRIBUTES ALIAS: '_GetIndexF':: GetIndexF
  integer, external            :: GetIndexF

  logical, save                :: Init = .TRUE.

  integer i, j, k
```

If the first call, set up values for the location of CATION, ANION, and all other water quality determinands.

```
  if (Init) then
    Init = .FALSE.
    n = NumberOfDeterminands()        ! How many components are used?
    nBound = n
    if (n .gt. 0) then
      allocate (ComponentLoc(n))        ! Dimension a work array
      ComponentLoc(1) = GetIndexF('CATION')
      ComponentLoc(2) = GetIndexF('ANION')
      j = 2
      do i = 1, n
        if (i .ne. ComponentLoc(1) .and. i .ne. ComponentLoc(2)) then
         j = j + 1
         ComponentLoc(j) = i
        end if
      end do
    end if
    if (n .gt. 0) then            ! Cations/anions found in WQ stream
      n = n + 4                   ! Add to count for V, T, [H] and Hydraulic
age
    else                          ! Cations/anions NOT found in WQ stream
      n = n + 6
    end if
  end if
end subroutine Setup

!  Balancing Tank.f90
!
!  FUNCTIONS/SUBROUTINES exported from Balancing Tank.dll:
!  Balancing Tank      - subroutine
!
! pH model
!   Work with excess of anion/cation
!   Given pH -> [H+], [OH-]
!   Then assign [Cat+], [An-] to close electroneutrality
!  Then we have
!   d [Cat] = ... Standard equation
!   d [An ] = ... Standard equation
!
!  Electroneutrality
!     [Cat+] + [H+] = [An-] + [OH-]
!  Equilibrium
!     [H+] [OH-] = Kw
!     [Cat+][H+] + [H+]**2 = [H+][An-] + Kw
!
! Solution options:
!     [1] Solve for Cat', An'
!           Solve for [H+] explicitly
!           This is an ODE system with an explicit algebraic
!           embedded equation.
!     [2] Solve for Cat', An', [H+]
!           This is a DAE system, of index-1
!           Usually easy to solve, providing
!           consistent initial conditions are used
!     [3] Solve for Cat', An', H' -- see below
!           This has turned the DAE system into an
!           ODE system. The DAE system was index-1 because
!           only one level of differentiation of [H+] was needed
!           to remove the algebraic constraints.
!
!   Taking time derivative
!     H Cat' + Cat H' + 2 H H' = H An' + An H'
!     H' (2 H + Cat - An) = H (An' - Cat')
!     H' (2 H + Kw/H - H) = H (An' - Cat')
!     H'  = (An' - Cat') H / (H + Kw/H)
!         = (An' - Cat') H**2 / (H**2 + Kw)
!
subroutine BalancingTank(Time, Stages, Descriptor, S,         &
                         y,  yStage, x, xStage,               &
                         dy, dyStage, Results, StageResults,  &
                         Operation, StageOperation, ModelData)
use CoreTypes
USE SharedData
implicit none
double precision:: Time
integer         :: Stages
type(PROCESS)   :: Descriptor
type(STREAM)    :: S(*)
double precision:: y(*),         yStage(Stages, *)
double precision:: x(*),         xStage(Stages, *)
double precision:: dy(*),        dyStage(Stages, *)
double precision:: Results(*),   StageResults(Stages, *)
double precision:: Operation(*), StageOperation(Stages, *)
double precision:: ModelData(*)

  ! Expose subroutine Balancing Tank to users of this DLL
  !
  ! Export routine from DLL and ensure that it has
  ! the name 'BalancingTank'
  !DEC$ ATTRIBUTES DLLEXPORT::Balancing Tank
  !DEC$ ATTRIBUTES ALIAS: 'BalancingTank':: BalancingTank

  ! Variables
  double precision:: MinVol, MaxVol, PumpFlow
  double precision:: Volume, Qoverflow, Qpump, Qin
  double precision:: H, OH, An, Cat, T, K
  integer         :: i, j, in

  ! Body of Balancing Tank
  MaxVol = ModelData(1)
  MinVol = ModelData(2)
  PumpFlow = Operation(1)

  in = Descriptor%InStream(1)
  Qin = s(in)%Flow

  !n = NumberOfDeterminands() + 6
  Volume = y(1)
  Qoverflow = 0d0
  Qpump = PumpFlow
  if (Volume .le. MinVol) then
    Qpump = min(Qpump, Qin)
  elseif (Volume .ge. MaxVol) then
    Qoverflow = max(0d0, Qin - Qpump)
  end if

 ! Y( 1) = Volume
 !    2  = Hydraulic age
 !    3 = Temperature
 !    4 = Cations
 !    5 = Anions
 !    6 = [H+]
 !    7 .. NumberOfComponents = Water quality parameters
  dy(1) = Qin - Qpump - Qoverflow
  if (Volume .gt. 0d0) then
    dy(2) = 1d0 - Qin * y(2) / Volume
    dy(3) = Qin * (s(in)%T - y(3)) / Volume
    T   = y(3)
    K   = Kw(T)
    H   = 10d0 ** (-s(in).pH)
    OH  = K / H
!
    if (nBound .gt. 0) then
      Cat = s(in)%Value(ComponentLoc(1))
      An  = s(in)%Value(ComponentLoc(2))
    else
      Cat = max(0d0, H - OH)
      An  = max(0d0, OH - H)
    end if

! Typical concentrations are of the order 1E-6 - 1E-8
! Multiply by 1E10 to attain some scaling
    dy(4) = Qin * (1d10 * Cat - y(4)) / Volume
    dy(5) = Qin * (1d10 * An  - y(5)) / Volume
    H     = y(6)
    dy(6) = (dy(5) - dy(6)) * H**2 / (H**2 + K*1d20)
    if (n .gt. 6) then
      dy(7:n) = Qin * (s(in)%Value(ComponentLoc(3:)) - y(7:n)) / Volume
    end if
  else
    dy(2)  = 0d0
    dy(3:n) = 0d0
  end if

contains
  function Kw(T) result(K)
  double precision:: T, K
    K = 1d-14
  end function
end subroutine BalancingTank

!DEC$ ATTRIBUTES DLLEXPORT::Balancing TankS
!DEC$ ATTRIBUTES ALIAS: 'BalancingTankS':: BalancingTankS
```

```
subroutine BalancingTankS(t, Stages, Descriptor, S,            &
                          y,  yStage, x, xStage,               &
                          Results, StageResults,               &
                          Operation, StageOperation, ModelData)
use CoreTypes
USE SharedData
implicit none
double precision:: t
integer         :: Stages
type(PROCESS)   :: Descriptor
type(STREAM)    :: S(*)
double precision:: y(*),         yStage(Stages, *)
double precision:: x(*),         xStage(Stages, *)
double precision:: Results(*),   StageResults(Stages, *)
double precision:: Operation(*), StageOperation(Stages, *)
double precision:: ModelData(*)
!
! Local variables
!
integer:: PumpedFlow, Overflow, In
double precision:: Volume, MinVol, MaxVol
double precision:: PumpFlow, Qin

  Call SetUp

  in         = Descriptor%InStream(1)
  pumpedflow = Descriptor%OutStream(1)
  overflow   = Descriptor%OutStream(2)
  volume     = y(1)

  MaxVol = ModelData(1)
  MinVol = ModelData(2)
  PumpFlow = Operation(1)
  Qin = s(in)%Flow

  if (volume .le. MinVol) then
    s(pumpedflow) = 0d0
    s(overflow)   = 0d0
  else
    s(pumpedflow)%flow = PumpFlow
    s(pumpedflow)%t    = y(3)
    if (y(6) .gt. 0d0) then
      s(pumpedflow)%ph   = -log10(y(6) / 1d10)
    else
      s(pumpedflow)%ph   = 7d0
    endif
    if (nBound .gt. 0) then
      s(pumpedflow)%Value(ComponentLoc(:)) = [y(4) / 1d10, y(5) / 1d10, y(7:n)]
    end if
    if (volume .lt. MaxVol) then
      s(overflow) = 0d0
    else
      s(overflow) = s(pumpedflow)
      s(overflow)%flow = Qin - PumpFlow
    end if
  endif
end subroutine BalancingTankS



module ModelTypes
USE StreamMatch
  integer, parameter:: iLen   = 255
  integer, parameter:: iLenp1 = iLen + 1
  type ProcessModel
    integer:: ModelID
    integer:: ModelPointerDiff
    integer:: ModelPointerAlloc
  end type

  integer, parameter:: MAX_INLETS = 3
  integer, parameter:: MAX_OUTLETS = MAX_INLETS
  type Process
    integer:: ModelID
    integer:: ModelIndex
    integer:: InStream(MAX_INLETS)
    integer:: OutStream(MAX_OUTLETS)
    integer:: Stages
    integer:: y
    integer:: yStage
    integer:: x
    integer:: xStage
    integer:: Results
    integer:: StageResults
    integer:: Operation
    integer:: StageOperation
    integer:: ModelData
  end type

  type Stream
    double precision:: flow
    double precision:: t        ! Temperature
    double precision:: pH
    double precision:: Value(MAX_DETERMINANDS)
  end type

 integer, parameter:: LIBRARY_MODELS = 0
end module
```

APPENDIX II: Pellet Softener

Configuration file

```
<?xml version="1.0"?>
<!-- Pellet softening data file -->
<model>
  <name = "Pellet Softening" />            Descriptive name
  <dll = "PelletSoftener.dll" />           DLL file name
  <diff-entry = "Pelsof_derivative" />      Entry point for differential
equations
  <alg-entry = "Pelsof_outputs" />         Entry point for stream assignment

  <ImageData>                              Set up icon data.
    <Symbol-Count = 1 />                   Only one icon.
    <Inlet-Name = ["Inlet"] />
    <Outlet-Name = ["Outlet"] />
    <Icon>
      <Filename = "models\Pellet softening\pels25_f.jpg" />
      <Inlet-Loc = [0, 183] />
      <Outlet-Loc = [131, 108] />
    </Icon>
  </ImageData>

  <Static-Data>
    <Parameter>
      <Name = "Name" />
      <Legend = "Name" />
      <Type = STRING />
      <Default = "Pellet softening C" />
    </Parameter>

<!-- Stage, if used, MUST be the second static-data parameter, and MUST be
called STAGES -->
    <Parameter>
      <NAME = STAGES />
      <Legend = "Number of completely mixed reactors (-)" />
      <Type = REAL />
      <Range>1; 20</range>    Limits the number of stages to the range 1 .. 20
      <Default = 10 />
    </Parameter>
    <Parameter>
      <Name = A />
      <Legend = "Surface of reactor (m2)" />
      <!--<ID = 100 /-->
      <Type = REAL />
      <Range>0; 1E10</range>
      <Default = 5 />
    </Parameter>

    <Parameter>
      <NAME = d0 />
      <Legend = "Initial grain size (mm)" />
      <Type = REAL />
      <Range>0; 1E10</range>
      <Default = 0.25 />
    </Parameter>

    <Parameter>
      <NAME = rhog />
      <Legend = "Density grains (kg/m3)" />
      <Type = REAL />
      <Range>0; 1E10</range>
      <Default = 4100 />
    </Parameter>

    <Parameter>
      <NAME = rhos />
      <Legend = "Density Calcium Carbonate (kg/m3)" />
      <Type = REAL />
      <Range>0; 1E10</range>
      <Default = 2500 />
    </Parameter>

    <Parameter>
      <NAME = rhol />
      <Legend = "Density Caustic Soda Solution (kg/m3)" />
      <Type = REAL />
      <Range>0; 1E10</range>
      <Default = 1270 />
    </Parameter>

    <Parameter>
      <NAME = sl />
      <Legend = "Strength Caustic Soda Solution (kg/kg)" />
      <Type = REAL />
      <Range>0; 1E10</range>
      <Default = 0.25 />
    </Parameter>

    <Parameter>
      <NAME = kg0 />
      <Legend = "Initial grain mass in reactor (kg)" />
      <Type = REAL />
      <Range>0; 1E10</range>
      <Default = 3000 />
    </Parameter>

    <Parameter>
      <NAME = d />
      <Legend = "initial diameter per reactor (-)" />
      <Type = REAL />
      <Range>0; 1E10</range>
      <Default = 1 />
    </Parameter>

    <Parameter>
      <NAME = KT20 />
      <Legend = "Crystallisation constant (m3 m/ mol s)" />
      <Type = REAL />
      <Range>0; 1E10</range>
      <Default = 0.026 />
    </Parameter>

    <Parameter>
      <NAME = Df />
      <Legend = "Diffusion (-)" />
      <Type = REAL />
      <Range>0; 1E10</range>
      <Default = 3.18e-011 />
    </Parameter>

    <Parameter>
      <NAME = IoEc />
      <Legend = "EGV to IS conversion (-)" />
      <Type = REAL />
      <Range>0; 1E10</range>
      <Default = 0.183 />
    </Parameter>

    <Parameter>
      <NAME = FB />
      <Legend = "0:bed model 1:no bed growth (-)" />
      <Type = REAL />
      <Range>0; 1E10</range>
      <Default = 0 />
  </Static-Data>
```

We implement a non-standard feature of XML. Comments should be enclosed between <!–and -->. We allow a C++-style one-line comment, where the comment starts with //.
// Specify if we will want the operational settings to be scheduled with time
// Default is FALSE; use TRUE if wanted

```
  <time-varying-operation = FALSE />
  <Operational-Data>
    <Parameter>
      <Name = NaOH />
      <Legend = "loog dosage (l/h)" />
      <Type = REAL />                       If <time-varying-operation> was
      <Range>0; 1E10</range>                set to TRUE then the type should
      <Default = 0 />                       be set to GRID.
    </Parameter>

    <Parameter>
      <Name = Ngrainsin />
      <Legend = "Ngrainsin (-)" />
      <Type = REAL />
      <Range>0; 1E10</range>
      <Default = 0 />
    </Parameter>

    <Parameter>
      <Name = Npelletsout />
      <Legend = "Npelletsout (-)" />
      <Type = REAL />
      <Range>0; 1E10</range>
      <Default = 0 />
    </Parameter>
  </Operational-Data>

  <Stage-initial-conditions>
    <Parameter>
      <Name = Calcium/>
      <Legend = "Ca (mg/l)" />
      <Type = GRID />               Selecting the type as GRID is essential
      <Default = 90 />                      for stage-wise entries.
    </Parameter>

     <Parameter>
      <Name = Mnumber />
      <Legend = "Mnumber (mmol/l)" />
      <Type = GRID />
      <Default = 0 />
    </Parameter>

    <Parameter>
      <Name = Pnumber />
      <Legend = "Pnumber (mmol/l)" />
      <Type = GRID />
      <Default = 0 />
    </Parameter>

    <Parameter>
      <Name = "Basis ionstrength" />
      <Legend = "IB" />
      <Type = GRID />
      <Default = 0 />
    </Parameter>

    <Parameter>
      <Name = "Calcium carbonate" />
      <Legend = "CaCO3" />
      <Type = GRID />
      <Default = 0 />
    </Parameter>

    <Parameter>
      <Name = "Mass Garnet" />
      <Legend = "Garnet (kg)" />
      <Type = GRID />
      <Default = 0 />
    </Parameter>
  </Stage-initial-conditions>
```

<Results> section indicates results that will be returned by the model. State variables (initial conditions) are always returned, but additional, computed, results may be wanted.

```
  <Results>
```

```
    <Parameter>
      <Name = "Total Pressure drop" />
      <Legend = "Total Pressure drop (Bar)" />
      <Type = REAL />
    </Parameter>

    <Parameter>
      <Name = "Total Bedheight" />
      <Legend = "Total Bedheight (m)" />
      <Type = REAL />
    </Parameter>

    <Parameter>
      <Name = "Released pellet diameter" />
      <Legend = "Bedheight (m)" />
      <Type = REAL />
    </Parameter>

    <Parameter>
      <Name = "Saturation Index" />
      <Legend = "SI (-)" />
      <Type = REAL />
    </Parameter>
  </Results>

  <Stage-Results>
    <Parameter>
      <Name = "Diameter pellets" />
      <Legend = "dp (mm)" />
      <Type = GRID />
    </Parameter>

    <Parameter>
      <Name = "Porosity" />
      <Legend = "Pe (-)" />
      <Type = GRID />
    </Parameter>

    <Parameter>
      <Name = "Bedheight" />
      <Legend = "dX (m)" />
      <Type = GRID />
    </Parameter>

    <Parameter>
      <Name = "pH" />
      <Legend = "pH (-)" />
      <Type = GRID />
    </Parameter>

    <Parameter>
      <Name = "Pressure drop" />
      <Legend = "dP (kPa)" />
      <Type = GRID />
    </Parameter>

  </Stage-Results>

<!-- Ensure that the following are defined in streams connecting processes -->
<WaterQualityParameters>
    <determinand>
      <wq-code = "CA" />
      <wq-legend = "Calcium" />
      <wq-units = "(mg/L)" />
    </determinand>

    <determinand>
      <wq-code = "HCO3" />
      <wq-legend = "Hydrogen carbonate" />
      <wq-units = "(mg/L)" />
    </determinand>

    <determinand>
      <wq-code = EC />
      <wq-legend = "Electrical conductivity" />
      <wq-units =  "(mS/m)" />
    </determinand>

    <determinand>
      <wq-code = "Mnumber" />
      <wq-legend = "Mnumber" />
      <wq-units = "mmol" />
    </determinand>

    <determinand>
      <wq-code = "Pnumber" />
      <wq-legend = "Pnumber" />
      <wq-units = "mmol" />
    </determinand>
  </WaterQualityParameters>
</model>
```

Source code

```
// PelletsofteningC.cpp : Defines the entry point for the DLL application.
//

#include <windows.h>
#include <math.h>
#include "SimdEauTypes.h"     Definition of structures
#include "SimEauCore.h"               Definition of DLL import/export features
#include "CarbonEq.c"         Subsidiary file for carbonate chemistry.
                                      Not included in this example.

/*molecular masses */
const double mCa = 40;
const double mC  = 12;
const double mO2 = 16;
const double mH  = 1;
const double mNa = 23;
const double mCaCO3 = mCa + mC + 3*mO2;
const double mHCO3 = mH + mC + 3*mO2;
const double mNaOH = mNa + mO2 + mH;

int POS_CA      = 0;
int POS_HCO3    = 0;
int POS_EC      = 0;
int POS_Mnumber = 0;
int POS_Pnumber = 0;

double Kw(double T) {return 1e-14;}

double pi=3.14159265359;

double functiep_Ergun(double nu,double rhow,double rhop,double v,double d);
double functiep_RZ   (double nu,double rhow,double rhop,double v,double d,
                      int flag);

double MAX(double x, double y)
{
   if (x > y) return x;
    else return y;
}

Mandatory subroutine header – the names can be chosen by the user, but the
parameter list is simposed by SimEau.
EXPORT void Pelsof_derivative(double * Time, long * Stages,
                                     SDProcess *Descriptor, SDStream *S,
                                     double *y,         double **yStage,
                                     double *x,         double **xStage,
                                     double *dy,        double **dyStage,
                                     double *Results,   double **StageResults,
                                     double *Operation, double
**StageOperation,
                                     double *ModelData)
{
/*parameters */
double A,d0,rhog,rhos,rhow,rhol,KT,KT20;
```

```
long NumCel;
double Q,Temp,pH0,HCO30,Ca0,EC,NaOH,NGrains,NPellets,sl;
double Mgetal0,Pgetal0,IB0,Uionsterkteco,Factiefco;
double TempK,nu;

double d[20],rhop[20],pe[20],dX[20],Sp[20];
double Mgetal[20],Pgetal[20],pH[20],CO3[20],HCO3[20];
double CaCO3[20], mg[20], Ca[20], dP[20];
double DF[20], IB[20], Factief[20];
double Nc[20], Nt[20];
double Ntot;
double KS, IoEc;
double K1,K2,Kw;
double FB, Sa;
double Df,Reh,Sc,Sh,kf;


int i;
int in;

NumCel = * Stages;

A      = ModelData[ 1];          /* surface area reactor */
d0     = ModelData[ 2]/1000.0;   /* initial grain diameter */
rhog= ModelData[ 3];            /* density grains */
rhos= ModelData[ 4];            /* density caco3 */
rhol= ModelData[ 5];            /* density caustic soda solution */
sl  = ModelData[ 6];            /* strength caustic soda solution */
KT20= ModelData[ 9]/1000.0;     /* KT Invoer naar mmol*/
Df  = ModelData[10];
IoEc= ModelData[11];
FB  = ModelData[12];

if (NumCel > 20) NumCel=20;   /* Limit number of cells, because of limited
                                         memory allocation. */

in = Descriptor->InStream[0] - 1;
Q    = S[in].Flow/3600.0;            /* water flow, limited to 200 m3/h */
Temp = S[in].Temperature;            /* Temperature in oC, limited to 1 oC */
pH0  = S[in].pH;                     /* influent pH between 6 and 9*/
Ca0  = S[in].Value[POS_CA]/mCa;     /* influent calcium conc. (mmol/l) */
HCO30= S[in].Value[POS_HCO3]/mHCO3; /* influent HCO3 concentration (mmol/l) */
EC   = S[in].Value[POS_EC];         /* influent electrical conductivity mS/m */

NaOH = 1E-3*Operation[0]/(S[in].Flow+1E-3*Operation[0])*sl*rhol/mNaOH*1E3;
//1e3 voor mol/l -> mmol/l

NGrains = Operation[1]*1000.0;          // number of grains in
NPellets = MAX(Operation[2]*1000.0,0);  // number of pellets out

// change to  kg/s
NGrains = NGrains * rhog * (pi/6*d0*d0*d0);
NPellets = NPellets * rhog * (pi/6*d0*d0*d0);

TempK = Temp + 273.15; /* Temperature in Kelvin */
nu = 4.97e-4 / (pow(Temp+42.5,3.0/2.0)); /* viscosity */
rhow = ( 0.2198670356299949E16/2199023255552.0
        +0.4769643019957967E16/281474976710656.0*Temp
        -0.4604215144723611E16/0.5764607523034235E18*Temp*Temp
        -212923669458047.0/0.4611686018427388E19*Temp*Temp*Temp
        +249253633739249.0/0.2361183241434823E22*Temp*Temp*Temp*Temp
        -0.5426481728272187E16/0.1934281311383407E26*Temp*Temp*Temp*Temp*Temp
       )
        /(1.0+0.4865285514884471E16/0.2882303761517117E18*Temp);

KT=pow(1.053,(Temp-20.0))*KT20;

Uionsterkteco=IoEc*EC; /* mmol/l benaderingsformule */
Factiefco = CE_Activity(Uionsterkteco);

KValues(&K1,&K2,&Kw,&KS,TempK);
Mgetal0 = CE_pHHCO3_M(pH0,HCO30,K1,K2,Kw,Factiefco);
Pgetal0 = CE_pHHCO3_P(pH0,HCO30,K1,K2,Kw,Factiefco);
//IS = IB + (2*CO3+HCO3/2 +H3O/2 + OH/2);
```

```
IB0 = Uionsterkteco - (HCO30/2) + NaOH/2 ; // aanname CO3 =0, H3O =0 OH =0 en
Na+ bij basis sterkte op.


/*mixing */
Pgetal0 = Pgetal0+NaOH;
Mgetal0 = Mgetal0+NaOH;

/* reactor */
for (i=0; i < NumCel; i++)
{
  Ca[i]     = y[i];
  Mgetal[i] = y[NumCel+i];
  Pgetal[i] = y[2*NumCel+i];
  IB[i]     = y[3*NumCel+i];
  CaCO3[i]  = y[4*NumCel+i];
  mg[i]     = y[5*NumCel+i];

  if (CaCO3[i]<0) CaCO3[i]=0.0;
  if (IB[i]<0)    IB[i]=0.0;
  if (mg[i]<0.1)  mg[i]=0.1;

  //pellet diameter
  //d[i]=pow( pow(d0,3) + 6.0/pi*CaCO3[i]/(rhos*Ng[i]),(1.0/3.0));
  d[i] = d0*pow((1+CaCO3[i]/mg[i]*rhog/rhos),(1.0/3.0));

  //pellet density
  //rhop[i]=(rhog*pow(d0,3)+rhos*(pow(d[i],3)-pow(d0,3)))/pow(d[i],3);
  rhop[i]=(CaCO3[i]+mg[i])/(CaCO3[i]/rhos + mg[i]/rhog);

  // porosity
  //pe[i]=functiep(nu,rhow,rhop[i],Q/A,d[i],Cd,nd);
  pe[i]=functiep_RZ(nu,rhow,rhop[i],Q/A,d[i],3);


  // bedheight
  //dX[i]=Ng[i]*pi/6.0*pow(d[i],3)/((1-pe[i])*A);
  dX[i] = (CaCO3[i]+mg[i])/(rhop[i]*(1.0-pe[i])*A);

  //specific surface
  Sp[i]=6.0*(1-pe[i])/(d[i]);
  Sp[i] = Sp[i]/pe[i];

  //pressuredrop
  dP[i]=((1-pe[i])*dX[i])*(rhop[i]-rhow)/rhow;


  /* schatting met Factiefco */
  if (i==0)
    pH[i]=CE_MP_pH(Mgetal[i],Pgetal[i],K1,K2,Kw,Factiefco,9.0);
  else
    pH[i]=CE_MP_pH(Mgetal[i],Pgetal[i],K1,K2,Kw,Factiefco,pH[i-1]);

  CO3[i]=CE_pHM_CO3(pH[i],Mgetal[i],K1,K2,Kw,Factiefco);
  HCO3[i]=CE_pHM_HCO3(pH[i],Mgetal[i],K1,K2,Kw,Factiefco);

  Factief[i]=CE_Activity(IB[i] + HCO3[i]/2 + 2*CO3[i]);

  /* berekening Factief */
  pH[i]=CE_MP_pH(Mgetal[i],Pgetal[i],K1,K2,Kw,Factief[i],pH[i]);

  CO3[i]=CE_pHM_CO3(pH[i],Mgetal[i],K1,K2,Kw,Factief[i]);
  HCO3[i]=CE_pHM_HCO3(pH[i],Mgetal[i],K1,K2,Kw,Factief[i]);

  Sa = (Ca[i]*CO3[i]-KS/(pow(Factief[i],8.0)));

  //diffusie
  Reh = 2.0/3.0*Q/A*d[i]/(1.0-pe[i])/nu;
  Sc = nu/Df;
  Sh = 0.66*pow(Reh,0.5)*pow(Sc,0.33);
  kf = Sh*Df/d[i];

  DF[i] = (KT*kf/(KT+kf))*Sp[i]*Sa;
        if (DF[i]<0)
```

```
              DF[i]=0;

}
/* doorschuiven pellets */
Ntot=mg[0];
Nc[0]=mg[0];
for (i=1; i < NumCel; i++)
{
  Nc[i] = Nc[i-1]+mg[i];
  Ntot += mg[i];
}
for (i=0; i < NumCel; i++)
{
  Nc[i] = Nc[i]/Ntot;
}
Nt[0] = NPellets;
for (i=1; i < NumCel; i++)
{
  Nt[i] = (1-Nc[i-1])*NPellets + Nc[i-1]*NGrains;
}

/* End general calculations */

/* mengcompartiment */
dy[0]         = Q/(A*pe[0]*dX[0])*(Ca0-Ca[0])         -   DF[0];
dy[NumCel]    = Q/(A*pe[0]*dX[0])*(Mgetal0-Mgetal[0]) - 2.0*DF[0];
dy[2*NumCel] = Q/(A*pe[0]*dX[0])*(Pgetal0-Pgetal[0]) -   DF[0];
dy[3*NumCel] = Q/(A*pe[0]*dX[0])*(IB0-IB[0])         - 2.0*DF[0];
if (FB==0)
{
   // /1000 voor omzetten naar kg
  dy[4*NumCel] = (DF[0]*A*pe[0]*dX[0]*mCaCO3)/1000.0;
  dy[4*NumCel] -= CaCO3[0]/mg[0]*Nt[0];
  dy[4*NumCel] += CaCO3[1]/mg[1]*Nt[1];
  dy[5*NumCel] = Nt[1]-Nt[0];
}
else
{
  dy[4*NumCel]=0;
  dy[5*NumCel]=0;
}


/* overige comparimenten */

for (i=1; i<NumCel-1; i++)
{
    dy[i]          = Q/(A*pe[i]*dX[i])*(Ca[i-1]-Ca[i])         -   DF[i];
    dy[NumCel+i]   = Q/(A*pe[i]*dX[i])*(Mgetal[i-1]-Mgetal[i]) - 2*DF[i];
    dy[2*NumCel+i] = Q/(A*pe[i]*dX[i])*(Pgetal[i-1]-Pgetal[i]) -   DF[i];
    dy[3*NumCel+i] = Q/(A*pe[i]*dX[i])*(IB[i-1]-IB[i])         - 2*DF[i];
    if (FB==0)
    {
    // /1000 voor omzetten naar kg
    dy[4*NumCel+i] = (DF[i]*A*pe[i]*dX[i]*mCaCO3)/1000.0;
    dy[4*NumCel+i] -= CaCO3[i]/mg[i]*Nt[i];
    dy[4*NumCel+i] += CaCO3[i+1]/mg[i+1]*Nt[i+1];
    dy[5*NumCel+i] = Nt[i+1]-Nt[i];
    }
    else
{
  dy[4*NumCel+i]=0;
  dy[5*NumCel+i]=0;
}

}

i = NumCel-1;
    dy[i]          = Q/(A*pe[i]*dX[i])*(Ca[i-1]-Ca[i])         -   DF[i];
    dy[NumCel+i]   = Q/(A*pe[i]*dX[i])*(Mgetal[i-1]-Mgetal[i]) - 2*DF[i];
    dy[2*NumCel+i] = Q/(A*pe[i]*dX[i])*(Pgetal[i-1]-Pgetal[i]) -   DF[i];
    dy[3*NumCel+i] = Q/(A*pe[i]*dX[i])*(IB[i-1]-IB[i])         - 2*DF[i];
    if (FB==0)
    {
```

```
    // /1000 voor omzetten naar kg
    dy[4*NumCel+i] = (DF[i]*A*pe[i]*dX[i]*mCaCO3)/1000.0;
    dy[4*NumCel+i] -= CaCO3[i]/mg[i]*Nt[i];
    dy[4*NumCel+i] += 0;
    dy[5*NumCel+i] = NGrains - Nt[i];
    }
    else
    {
      dy[4*NumCel+i]=0;
      dy[5*NumCel+i]=0;
    }
}
```

The names can be chosen by the user, but the number of parameters and their types is imposed by the requirements of SimEau.

```
EXPORT void Pelsof_outputs(double *Time, long *Stages,
                                  SDProcess *Descriptor, SDStream  *S,
                                  double *y,         double **yStage,
                                  double *x,         double **xStage,
                                  double *Results,   double **StageResults,
                                  double *Operation, double **StageOperation,
                                  double *ModelData)
```

```
{
/*parameters */
double A,d0,rhog,rhos,rhow,rhol,KT,KT20;
long NumCel;
double Q,Temp,pH0,HCO30,Ca0,EC,NaOH,NGrains,NPellets,sl;
double Mgetal0,Pgetal0,IB0,Uionsterkteco,Factiefco;
double TempK,nu;
int Pumpedflow;

double d[20],rhop[20],pe[20],dX[20],Sp[20];
double Mgetal[20],Pgetal[20],pH[20],CO3[20],HCO3[20];
double CaCO3[20], mg[20], Ca[20], dP[20];
double DF[20], IB[20], Factief[20];
double Nc[20], Nt[20];
double deltaP, bedhoogte;
double Ntot;
double KS, IoEc;
double K1,K2,Kw;
double FB, Sa;
double Df,Reh,Sc,Sh,kf;


int i;
int in;

// Get stream parameters
if (*Time == 0.)
  {POS_CA      = GetIndex("CA", 2)      - 1;
   POS_HCO3    = GetIndex("HCO3", 4)    - 1;
   POS_EC      = GetIndex("EC", 2)      - 1;
   POS_Mnumber = GetIndex("MNUMBER", 7) - 1;
   POS_Pnumber = GetIndex("PNUMBER", 7) - 1;
  }

// NumCel is defined as the FIRST element of the ModelData array.
// It is ALSO passed as the value STAGES
//NumCel = (int) ModelData[0]; /* aantal cellen */
NumCel = * Stages;
A    = ModelData[ 1];                 // surface area reactor
d0   = ModelData[ 2]/1000.0;          // initial grain diameter
rhog = ModelData[ 3];                 // density grains
rhos = ModelData[ 4];                 // density caco3
rhol = ModelData[ 5];                 // density caustic soda solution
sl   = ModelData[ 6];                 // strength caustic soda solution
                                     // 7:: Initial mass of grains
                                     // 8: Initial diameter [of what?]
KT20= ModelData[ 9]/1000.0;          // KT Invoer naar mmol
Df  = ModelData[10];
IoEc= ModelData[11];
FB  = ModelData[12];
```

```
if (NumCel > 20) NumCel=20;  /* Limit number of cells, because of limited
memory
                                        allocation. */

in = Descriptor->InStream[0] - 1;
Q    =S[in].Flow/3600.0;          /* water flow, limited to 200 m3/h */
Temp =S[in].Temperature;           /* Temperature in oC, limited to 1 oC */
pH0  =S[in].pH;                    /* influent pH between 6 and 9*/
Ca0  =S[in].Value[POS_CA]/mCa;     /* influent calcium concentration (mmol/l)
*/
HCO30=S[in].Value[POS_HCO3]/mHCO3; /* influent HCO3 concentration (mmol/l) */
EC   =S[in].Value[POS_EC];         /* influent electrical conductivity mS/m */

//1e3 voor mol/l -> mmol/l
NaOH = 1E-3*Operation[0]/(S[in-1].Flow+1E-3*Operation[0])*sl*rhol/mNaOH*1E3;
NGrains = Operation[1]*1000.0;          // number of grains in
NPellets = MAX(Operation[2]*1000.0,0);  // number of pellets out

// change to  kg/s
NGrains = NGrains * rhog * (pi/6*d0*d0*d0);
NPellets = NPellets * rhog * (pi/6*d0*d0*d0);

TempK = Temp + 273.15; /* Temperature in Kelvin */
nu = 4.97e-4 / (pow(Temp+42.5,3.0/2.0)); /* viscosity */
rhow = ( 0.2198670356299949E16/2199023255552.0
        +0.4769643019957967E16/281474976710656.0*Temp
        -0.4604215144723611E16/0.5764607523034235E18*Temp*Temp
        -212923669458047.0/0.4611686018427388E19*Temp*Temp*Temp
        +249253633739249.0/0.2361183241434823E22*Temp*Temp*Temp*Temp
        -0.5426481728272187E16/0.1934281311383407E26*Temp*Temp*Temp*Temp*Temp
       )/(1.0+0.4865285514884471E16/0.2882303761517117E18*Temp);

KT=pow(1.053,(Temp-20.0))*KT20;

Uionsterkteco=IoEc*EC; /* mmol/l benaderingsformule */
Factiefco = CE_Activity(Uionsterkteco);

KValues(&K1,&K2,&Kw,&KS,TempK);
Mgetal0 = CE_pHHCO3_M(pH0,HCO30,K1,K2,Kw,Factiefco);
Pgetal0 = CE_pHHCO3_P(pH0,HCO30,K1,K2,Kw,Factiefco);
//IS = IB + (2*CO3+HCO3/2 +H3O/2 + OH/2);
IB0 = Uionsterkteco - (HCO30/2) + NaOH/2 ; // aanname CO3 =0, H3O =0 OH =0 en
Na+ bij basis sterkte op.

/*mixing */
Pgetal0 = Pgetal0+NaOH;
Mgetal0 = Mgetal0+NaOH;

/* reactor */
for (i=0; i < NumCel; i++)
{
  Ca[i]     = y[i];
  Mgetal[i] = y[NumCel+i];
  Pgetal[i] = y[2*NumCel+i];
  IB[i]     = y[3*NumCel+i];
  CaCO3[i]  = y[4*NumCel+i];
  mg[i]     = y[5*NumCel+i];

  if (CaCO3[i]<0) CaCO3[i]=0.0;
  if (IB[i]<0)    IB[i]=0.0;
  if (mg[i]<0.1)  mg[i]=0.1;

  //pellet diameter
  //d[i]=pow( pow(d0,3) + 6.0/pi*CaCO3[i]/(rhos*Ng[i]),(1.0/3.0));
  d[i] = d0*pow((1+CaCO3[i]/mg[i]*rhog/rhos),(1.0/3.0));

  //pellet density
  //rhop[i]=(rhog*pow(d0,3)+rhos*(pow(d[i],3)-pow(d0,3)))/pow(d[i],3);
  rhop[i]=(CaCO3[i]+mg[i])/(CaCO3[i]/rhos + mg[i]/rhog);

  // porosity
  pe[i]=functiep_RZ(nu,rhow,rhop[i],Q/A,d[i],3);
  // bedheight
  //dX[i]=Ng[i]*pi/6.0*pow(d[i],3)/((1-pe[i])*A);
  dX[i] = (CaCO3[i]+mg[i])/(rhop[i]*(1.0-pe[i])*A);

  //specific surface
  Sp[i]=6.0*(1-pe[i])/(d[i]);
  Sp[i] = Sp[i]/pe[i];

  //pressuredrop
  dP[i]=((1-pe[i])*dX[i])*(rhop[i]-rhow)/rhow;


  /* schatting met Factiefco */
  if (i==0)
    pH[i]=CE_MP_pH(Mgetal[i],Pgetal[i],K1,K2,Kw,Factiefco,9.0);
  else
    pH[i]=CE_MP_pH(Mgetal[i],Pgetal[i],K1,K2,Kw,Factiefco,pH[i-1]);

  CO3[i]=CE_pHM_CO3(pH[i],Mgetal[i],K1,K2,Kw,Factiefco);
  HCO3[i]=CE_pHM_HCO3(pH[i],Mgetal[i],K1,K2,Kw,Factiefco);

  Factief[i]=CE_Activity(IB[i] + HCO3[i]/2 + 2*CO3[i]);

  /* berekening Factief */
  pH[i]=CE_MP_pH(Mgetal[i],Pgetal[i],K1,K2,Kw,Factief[i],pH[i]);

  CO3[i]=CE_pHM_CO3(pH[i],Mgetal[i],K1,K2,Kw,Factief[i]);
  HCO3[i]=CE_pHM_HCO3(pH[i],Mgetal[i],K1,K2,Kw,Factief[i]);

  Sa = (Ca[i]*CO3[i]-KS/(pow(Factief[i],8.0)));

  //diffusie
  Reh = 2.0/3.0*Q/A*d[i]/(1.0-pe[i])/nu;
  Sc = nu/Df;
  Sh = 0.66*pow(Reh,0.5)*pow(Sc,0.33);
  kf = Sh*Df/d[i];

  DF[i] = (KT*kf/(KT+kf))*Sp[i]*Sa;
  if (DF[i]<0) DF[i]=0;

}

/* doorschuiven pellets */
Ntot=mg[0];
Nc[0]=mg[0];
for (i=1; i < NumCel; i++)
{
  Nc[i] = Nc[i-1]+mg[i];
  Ntot += mg[i];
}
for (i=0; i < NumCel; i++) Nc[i] = Nc[i]/Ntot;

Nt[0] = NPellets;
for (i=1; i < NumCel; i++) Nt[i] = (1-Nc[i-1])*NPellets + Nc[i-1]*NGrains;

/* End general calculations */

/* Stimela water quality outputs */

Pumpedflow = Descriptor->OutStream[0] - 1;
S[Pumpedflow].pH= pH[NumCel-1]; /* effluent pH */

/* effluent calcium concentration (mg/l) */
S[Pumpedflow].Value[POS_CA]=  Ca[NumCel-1]*mCa;

/* effluent HCO3 concentration (mg/l) */
S[Pumpedflow].Value[POS_HCO3]= HCO3[NumCel-1]*mHCO3;

/* effluent electrical conductivity */
S[Pumpedflow].Value[POS_EC] = (IB[NumCel-1] + 2*CO3[NumCel-1]
                                            + HCO3[NumCel-1]/2 )/IoEc;

S[Pumpedflow].Value[POS_Mnumber]= Mgetal[NumCel-1];
```

```
S[Pumpedflow].Value[POS_Pnumber]= Pgetal[NumCel-1];


/* Additional outputs */
bedhoogte = 0;
deltaP = 0;
for (i=0; i < NumCel; i++)
{
    deltaP=deltaP+dP[i];
    bedhoogte = bedhoogte + dX[i];
/* diameter van pellets over de hoogte in mm */
    Results[NumCel+i]=d[i]*1000.0;
    Results[2*NumCel+i]=pe[i];
    Results[3*NumCel+i]=dX[i];
    Results[4*NumCel+i]=pH[i];
    Results[5*NumCel+i]=dP[i];         /* drukval */
}

/* Important outputs */
Results[0] = deltaP; /* Total Pressure drop */
Results[1] = bedhoogte; /* Bedheight */
Results[2] = d[0]*1000.0; /* Released pellet diameter */
Results[3]= log10(Ca[NumCel-1]*CO3[NumCel-1]/(KS/pow(Factief[NumCel-1],8)));
}

double fnkv(double vb, double d, double nu, int flag)
{
  double Re;
  double Cw;
  double res;

  Re = vb*d/nu;
  switch (flag)
  {
   case 0: Cw = 24.0/Re*(1.0 + 0.15*pow(Re,0.687));
           break;
   case 1: Cw = 24.0/Re*(1.0 + 0.0752121354307409*pow(Re,0.880220950948111));
           break;
   case 2: Cw = 24.0/Re*(1.0 + 0.0703133122353068*pow(Re,0.89489568693569));
           break;
   case 3: Cw = 24.0/Re*(1.0 + 0.0790784448728646*pow(Re,0.869705267565341));
           break;
  }

  res = Cw*pow(vb,2.0);

  return res;
}

double functiep_RZ(double nu,double rhow,double rhop,double v,double d,int
flag)
{
  // function to determine p based on iteration
  // to find the solution to fcnp(p)=Var2

  double xa, fnka, df,dx;

  double tol = 1e-3;
  int iter = 0;
  int itermax = 50;
  double fnk ;
  double Var2;
  double g;

  double x, x0;
  double Re, n;
  double p;


  g=9.81;

  x0=300.0/3600.0;

  Var2 = 4.0/3.0*d*(rhop-rhow)*g/rhow;
  fnk = fnkv(x0,d,nu,flag)-Var2;

  x=x0;
  x0 = x + 1;

  // Main iteration loop
  while ((fabs(x - x0) > tol) && (iter <= itermax))
  {
   iter = iter + 1;
   x0 = x;

   // Set dx for differentiation
   if (x != 0)
   {
      dx = x*tol;
         }
   else
   {
      dx = tol;
         }


   // Differentiation
   xa = x - dx;
   fnka = fnkv(xa,d,nu,flag)-Var2;
   df = (fnk - fnka)/(x - xa);

   // Next approximation of the root
   if (df == 0)
   {
      x = x0 + 1.1*tol;
           p=iter;
   }
   else
   {
      x = x0 - fnk/df;
   }

   fnk = fnkv(x,d,nu,flag)-Var2;
  }

  Re = x*d/nu;
  if (Re<500)
  {
    n=(4.4)*pow(Re,-0.1);
  }
  else
  {
    n=2.4;
  }

  p = pow(v/x,1.0/n);
  if (p<0.1)  p=0.1;
  if (p>0.99) p=0.99;
  return p;
}
```

APPENDIX III: Core Simulator

The core simulator is written in Fortran, and has responsibility for solving the differential equations. The flow logic is:

```
for each time step
  do
    call all stream assignment routines
  loop until all stream recycles have converged
  call all differential equation routines
  integrate differential equations
next time step
```

The following support routines are available:

```
Return the value of component COMP_NAME in stream cStream.
C/C++ users should call this routine as
GetStreamValue(Stream cStream, *char COMP_NAME, int len(COMP_NAME))
!DEC$ ATTRIBUTES DLLEXPORT, STDCALL, REFERENCE:: GetStreamvalue
!DEC$ ATTRIBUTES ALIAS: '_GetStreamValue':: GetStreamValue
double precision function GetStreamValue(cStream, Comp_Name)
type (Stream)   :: cStream
character       :: Comp_Name*(*)

Set the value of component COMP_NAME.
C/C++ users should call this routine as
SetStreamValue(Stream cStream, *char COMP_NAME, double Value,
               int len(COMP_NAME))
!DEC$ ATTRIBUTES DLLEXPORT, STDCALL, REFERENCE:: SetStreamvalue
!DEC$ ATTRIBUTES ALIAS: '_SetStreamValue':: SetStreamValue
subroutine SetStreamValue(cStream, COMP_NAME, Value)
type (Stream)   :: cStream
character       :: COMP_NAME*(*)
double precision:: Value

Return the location index in the stream water quality array for component CODE.
The array index is returned using the Fortran convention that the first array
element is at index 1, while C/C++ uses the convention that the first array
element is at index 0.
!DEC$ ATTRIBUTES DLLEXPORT, STDCALL, REFERENCE:: GetIndex
!DEC$ ATTRIBUTES ALIAS: '_GetIndex@8':: GetIndex
integer function GetIndex(Code)
character:: Code*(*)

The same as GetIndex, but written to be easier to access from Fortran
!DEC$ ATTRIBUTES DLLEXPORT, STDCALL, REFERENCE:: GetIndexF
!DEC$ ATTRIBUTES ALIAS: '_GetIndexF':: GetIndexF
integer function GetIndexF(Code)
character:: Code*(*)

The number of determinands in the current flowsheet. The maximum number of
permitted determinands is set as 500, and would require recompiling SimEau if
more than 500 water quality parameters were needed.
!DEC$ ATTRIBUTES DLLEXPORT, STDCALL, REFERENCE:: NumberOfDeterminands
!DEC$ ATTRIBUTES ALIAS: '_NumberOfDeterminands':: NumberOfDeterminands
integer function NumberOfDeterminands()
```

APPENDIX IV: Coding requirements

Configuration file

The structure of a configuration file is based on XML.

Comments may either use the XML approach, being enclosed in `<!--` and `-->`, or as an extension by being on a single line, preceded by the C++-style comment marker of `//`. There are two more extensions over standard XML:

- keywords are not case sensitive, so that `<model>`, `<Model>` and `<MODEL>` will all be interpreted the same; and
- Elements may be included within the keyword. Strict XML would require, for example, `<name>This is the model name</name>`, while SimEau supports the extension `<name = "This is the model name" />`.

A later release of SimEau may enforce the strict XML standards, so it is advised that these extensions are not used. They are documented here because the current examples use these extensions, but they will eventually be converted to standard XML.

Aside from comments, the data within the file must start with `<model>` and end with `</model>`.

The structure of the file proceeds as:

```
<model>
  <description>A description of what the model does.</description>
  <name>A short name for the model</name>
  <dll>The name of the DLL file.</name> The DLL file should be in the subfolder
                                        .\UserModels for user-written models,
                                         and .\Library for standard models.
Subroutine names may be decorated in a DLL. Typically, a compiler may add one
or two leading underscores, and may add a trailing @ followed by a number – the
number is usually four times the number of arguments. C++ compilers may
additionally many other leading and trailing characters, unless the extern "C"
attribute is used when defining the subroutine.
  <diff-entry>Name of differential equation subroutine</diff-entry>
  <alg-entry>Name of stream assignment subroutine</alg-entry>
  <ImageData>               The bitmaps to be used
    <SymbolCount>Number of images</SymbolCount>
    <Inlet-Name>
      [Comma-separated list of names, each name enclosed in quotes]
    </Inlet-Name>
    <Outlet-Name>
      [Comma-separated list of names, each name enclosed in quotes]
    </Outlet-Name>
    <Icon>                  Repeated for each image
      <Filename>
        Location of file, qualified from .\Library or .\UserModels folder
      </Filename>
      <Inlet-Loc>
        [Comma separated X, Y pairs, each paid separated by |]
        The values are the (X, Y) locations, in pixels, of the inlets
        for the image.
      </Inlet-Loc>
      <Outlet-Loc>
```

```
        [Comma separated X, Y pairs, each paid separated by |]
      </Outlet-Loc>
    </Icon>
  <static-data>              This marks the start of the static data section[4].
    <parameter>              Repeated for as many parameters as needed
      <name>Short name</name>
      <legend>Descriptive name</legend>
      <type>STRING, REAL, INTEGER, LIST, DATE, CHECK-BOX or GRID</type>
      <list>Comma-separated list. Only if <type> is LIST</list>
      <default>Default value</default>
      <sf>Number of significant figures when displayed</sf>
      <dp>Number of decimal places when displayed</dp>
      <range>Lower value; Upper value</range>
      <constraint>LESS-THAN(name); GREATER-THAN(name)</constraint>
      <tooltip>Short help or description</tooltip>
    </parameter>
  </static-data>
```

Specify if the <operation> section is to be made time-varying or not. Not that <stage-operation> CANNOT be time varying, and that there is no mechanism to have some operational settings time-varying and some not.

```
<time-varying-operation>TRUE or FALSE</time-varying-operation>
  <WaterQualityParameters>           Determinands that MUST be present in the
                                            Stream definitions.
    <determinands>
      <wq-code>CODE</wq-code>        CODE must match with the code dictionary
      <wq-legend>Display name</wq-legend>
      <wq-units>Display units</wq-units>
  </determinand>
</WaterQualityParameters>
```

Differential equation subroutine

```
subroutine locDLLSubDY(t, Stages, Descriptor, Streams,      &
                       y,  yStage, x, xStage,               &
                       dy, dyStage, Results, StageResults,  &
                       Operation, StageOperation, ModelData)
  use CoreTypes
  implicit none
  double precision:: t
  integer         :: Stages
  type(Process)   :: Descriptor
  type(Stream)    :: Streams(*)
  double precision:: y(*),         yStage(Stages, *)
  double precision:: x(*),         xStage(Stages, *)
  double precision:: dy(*),        dyStage(Stages, *)
  double precision:: Results(*),   StageResults(Stages, *)
  double precision:: Operation(*), StageOperation(Stages, *)
  double precision:: ModelData(*)
end subroutine locDLLSubDY
```

[4] **Other sections are:**

```
       <stage-static-data>,  Stage data need only be
                             specified for one stage,
                             and the GUI will use it for
                             the required number of
                             stages
       <operational-data>
       <stage-operational-data>
       <model-parameters>
       <reporting>
       <results>
       <stage-results>
       <ic>                  Initial conditions
       <stage-ic>            Stage-wise initial conditions
       <discrete>
       <stage-discrete>
       <determinands>
```

Stream assignment subroutine

```
subroutine locDLLSubAssign(t, Stages, Descriptor, Streams,      &
                           y,  yStage, x, xStage,               &
                           Results, StageResults,               &
                           Operation, StageOperation, ModelData)
  use CoreTypes
  implicit none
  double precision:: t
  integer         :: Stages
  type(Process)   :: Descriptor
  type(Stream)    :: Streams(*)
  double precision:: y(*),         yStage(Stages, *)
  double precision:: x(*),         xStage(Stages, *)
  double precision:: Results(*),   StageResults(Stages, *)
  double precision:: Operation(*), StageOperation(Stages, *)
  double precision:: ModelData(*)
end subroutine locDLLSubAssign
```

Structures

```
  integer, parameter:: MAX_DETERMINANDS = 500
  integer, parameter:: MAX_INLETS = 3
  integer, parameter:: MAX_OUTLETS = MAX_INLETS
  type Process
    integer:: ModelID
    integer:: ModelIndex
    integer:: InStream(MAX_INLETS)
    integer:: OutStream(MAX_OUTLETS)
    integer:: Stages
    integer:: y
    integer:: yStage
    integer:: x
    integer:: xStage
    integer:: Results
    integer:: StageResults
    integer:: Operation
    integer:: StageOperation
    integer:: ModelData
  end type

  type Stream
    double precision:: flow
    double precision:: t        ! Temperature
    double precision:: pH
    double precision:: Value(MAX_DETERMINANDS)
  end type
```