DTS200

Laboratory Setup Three - Tank - System
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assembly and Start-Up</td>
<td>1</td>
</tr>
<tr>
<td>Practical Instructions</td>
<td>2</td>
</tr>
<tr>
<td>Program Operation</td>
<td>3</td>
</tr>
<tr>
<td>Technical Data</td>
<td>4</td>
</tr>
<tr>
<td>DTS200 Software</td>
<td>5</td>
</tr>
<tr>
<td>Extension Kit Electrical Control Valves</td>
<td>6</td>
</tr>
</tbody>
</table>
1 Assembly and Start-Up

1.1 Unpacking ................................................................. 1-1
1.2 Setting up the System .................................................. 1-1
1.3 The Rear Panel ............................................................ 1-2
  1.3.1 The Power-Switch .................................................. 1-2
  1.3.2 Mains input ......................................................... 1-2
  1.3.3 Fuse ................................................................. 1-2
  1.3.4 System ............................................................ 1-2
  1.3.5 PC-Connector X1 .................................................. 1-2
1.4 The Front Panel ........................................................... 1-2
  1.4.1 Actuators (SERVO module) ....................................... 1-2
  1.4.2 Power Servo ....................................................... 1-3
  1.4.3 Mains Supply (POWER module) ................................ 1-3
  1.4.4 Signal Adaption Unit (SIGNAL ADAPTION module) with output stage release ... 1-3
  1.4.5 Measurement outputs (SENSOR module) ....................... 1-3
  1.4.6 Electrical Disturbance Module (SIGNAL ERROR module) .... 1-4
1.5 Connecting the System Components .................................... 1-4
  1.5.1 Option 200-02 ...................................................... 1-4
  1.5.2 Option 200-05 ...................................................... 1-5
1.6 Output Stage Release .................................................... 1-5
1.7 Filling the Tank System ................................................ 1-6
1.8 Venting the Pressure Sensor Lines ................................... 1-6
1.9 Maintenance .................................................................. 1-6
1.10 Transport ................................................................. 1-6
1.11 Start-Up and Test of Function ....................................... 1-6
1.12 Locating Errors .......................................................... 1-7
1 Assembly and Start-Up

1.1 Unpacking

After the DTS200 has been unpacked, all components are to be checked visually for damages as well as for completeness. Should anything be damaged notify us and save the item and the packaging material until final clarification.

The standard shipment of the DTS200 consists of:

- A premounted tank system complete with pumps, sensors and system cable.
- An actuator consisting of two servo amplifiers for the pumps (SERVO), a power supply unit with an integrated signal adaption unit (POWER) and a module for measuring outputs (SENSOR), contained within a 19” module box and a mains supply lead.
- An additional filling tube.
- A bottle of water colorant.

Depending on the desired system option, the shipment also includes the following items:

- Option 200-02
  A PC plug-in card MF624 for a PC with PCI slot, a 37 pol. connection cable as well as a CD containing the executable program for the system control.

- Option 200-03
  The CD contains the C++ source files of the program from the option 200-02 with additional library functions for graphic output.

- Option 200-05
  An electrical disturbance module (SIGNAL ERROR) in 19” plug-in design, which is contained in the actuator box at the time of delivery.

1.2 Setting up the System

Before setting up the system, please check whether your mains supply is identically to the mains supply indicated on the type plate (230V, 50/60Hz resp. 110V, 50/60Hz).

Before selecting a place to set up the system, you should consider the following points:

**Tank system:**

- Choose a place, where the tank system is not exposed to extreme temperatures. In particular direct sun light and direct heat radiation, e. g. by a radiator, are to be avoided.

- The tank system must be placed on a solid surface.

- Make sure that the area you selected is able to support the weight of all system components without difficulties.

- To guarantee a perfect control, the tank system must be completely leveled.

- Choose a hard surface area. Soft surfaces like carpets can hold a static charge. In case of contact, this can lead to discharge and to damages of sensitive circuits inside the actuator.

**Actuator:**

- The air must be able to circulate freely above, below as well as behind the actuator and the personal computer respectively.

- Do not place any heavy objects on top of the actuator.

- It is important, not to expose the actuator to extreme temperatures. Avoid intensive or direct sun radiation as well as any other heat sources.
Humidity and dust must be avoided. High humidity can lead to malfunctions and damages of the actuator.

1.3 The Rear Panel

Before putting the connections of the DTS200 into their positions, have a good look at the rear panel and locate the position of the connection sockets.

1.3.1 The Power-Switch

The power switch turns the power supply (mains supply) to the actuator ON and OFF.

1.3.2 Mains input

The mains supply lead is to be connected to the mains input.

1.3.3 Fuse

The fuse is a glass-tube fuse with M1.6 A (medium slow-blowing) in order to secure the 230 V mains supply.

1.3.4 System

The lead for the connection between the actuator and the tank system is to be connected here.

1.3.5 PC-Connector X1

In case you have selected Option 200-02, plug in the 37-pol. lead for the connection between the actuator and the 37-pol. connector labeled with X1 of the A/D-D/A card of the personal computer.

Note

The mains supply lead is to be connected only after the peripheral devices are connected. When peripheral devices are to be connected or disconnected, you must make sure that all components are switched off.

1.4 The Front Panel

Turn the actuator around so you can see the front panel. The following figure displays the components located on the front panel.

1.4.1 Actuators (SERVO module)

Both servo controllers for the pumps are located on the left-hand side of the front panel. The left module (servo controller 1) controls the flow of pump 1 for the left tank, the right module (servo controller 2) controls pump 2 for the right tank. Two light emitting diodes indicate the state of operation of each module.

Figure 1.1: Rear panel with denotations
• Ready (green) : The voltage supply is active.

• Limit (red) : The maximum liquid level of the corresponding tank is reached and the pump is shut down (overflow protection).

• Switch Automatic/Manual : This switch allows to change from computer based control (Automatic) to manual control (Manual) of the pumps.

• Potentiometer : For manual control of the servo controller, the flow of the corresponding pump is adjusted by the potentiometer.

### 1.4.2 Power Servo

This module provides the AC supply voltages for the two servo amplifiers. It does not contain any control or display element.

### 1.4.3 Mains Supply (POWER module)

The power module contains the power supply for the electronics.

- +15V (green) : A voltage of +15V is available.
- -15V (green) : A voltage of -15V is available.
- +5V (green) : A voltage of +5V is available.

### 1.4.4 Signal Adaption Unit (SIGNAL ADAPTION module) with output stage release

The signal adaption module contains the amplifiers for the adaption of the sensor signals as well as the electronics for the output stage release. The green LED on the front panel indicates whether the servo amplifiers are enabled. The output stage release is serviced by the external controller. By placing a jumper on the board into a certain position it is possible to enable the servo amplifiers permanently. More detailed description of the output stage release you may find later in this chapter.

### 1.4.5 Measurement outputs (SENSOR module)

The sensor module provides for external processing of the three sensor signals, the two control signals as well as ground at its 6 jacks.

- Tank1 : Processed signal of the liquid level sensor of tank 1.
- Tank2 : Processed signal of the liquid level sensor of tank 2.
- Tank3 : Processed signal of the liquid level sensor of tank 3.

Figure 1.2 : Front panel with denotations
• Q1: Processed control signal for the servo controller 1.

• Q2: Processed control signal for the servo controller 2.

• GND: Electrical ground potential corresponding to the described measurement outputs.

Note:

The signals of the liquid level sensors are processed as follows:
The sensor signal is amplified such that the resulting range is +9V to -9V. We use this reduced range (instead of +/- 10V) due to the fact that the offset of the sensor signals may be variable. Mainly air bubbles inside the pressure sensor line which cannot be avoided during the filling operation may produce different offset values. The variation of the resulting pressure signal offset will always be less than +/-1V so that the signal will remain in the limited range of +/-10V. At least after the first tank filling operation the pressure sensors have to be calibrated. This is performed with the assistance of the PC controller program (see "Practical Instructions").

1.4.6 Electrical Disturbance Module (SIGNAL ERROR module)

The signal error module is present only in case your system is equipped with Option 200-05. Otherwise, a blank panel will be installed in its place.

The module allows for a scaling of the processed sensor signals and the control signals in a range of 0% to 100%. By means of three switches, a simulation of total sensor failures is possible.

• Potentiometer Tank 1: Scales the amplified signal of the liquid level sensor of tank 1 in a range of 0% to 100%.

• Switch Tank 1: The switch setting "0%" simulates a complete failure of the liquid level sensor of tank 1. This is accomplished by grounding the voltage coming from the sensor amplifier before it is scaled by the corresponding potentiometer.

• Potentiometer Tank 2: Scales the amplified signal of the liquid level sensor of tank 2 in a range of 0% to 100%.

• Switch Tank 2: The switch setting "0%" simulates a complete failure of the liquid level sensor of tank 2. This is accomplished by grounding the voltage coming from the sensor amplifier before it is scaled by the corresponding potentiometer.

• Potentiometer Tank 3: Scales the amplified signal of the liquid level sensor of tank 3 in a range of 0% to 100%.

• Switch Tank 3: The switch setting "0%" simulates a complete failure of the liquid level sensor of tank 3. This is accomplished by grounding the voltage coming from the sensor amplifier before it is scaled by the corresponding potentiometer.

• Potentiometer Q 1: Scales the control signal for the servo controller of pump 1 in a range of 0% to 100% (this is ineffective for the switch setting "Manual" of servo controller 1).

• Potentiometer Q 2: Scales the control signal for the servo controller of pump 2 in a range of 0% to 100% (this is ineffective for the switch setting "Manual" of servo controller 2).

1.5 Connecting the System Components

1.5.1 Option 200-02

If your DTS200 is equipped with option 200-02 a personal computer is used as a controller. The PC plug-in card included in option 200-02 must be installed in your PC-AT computer. Please consult the manual of your computer for the installation of an additional card. All necessary components are included with the shipment of option 200-02.
After the personal computer has been set up and put into operation plug in the 37 pol. lead for the connection label with X1 between the PC plug-in card and the actuator to the jack "PC-Connector X1" on the rear panel.

### 1.5.2 Option 200-05

Option 200-05 is an electrical disturbance module which can operate together with option 200-02. At the time of delivery this module is already mounted in the box of the actuator.

Adjust the potentiometers for the signals Tank 1, Tank 2 and Tank 3 to 100%. To start operation, turn the potentiometers for the control signals Q 1 and Q 2 to 100%. None of the toggle switches is switched to the position "0%" (switch knob down).

When all potentiometers are turned to 100%, the actuator functions without the disturbance module.

### 1.6 Output Stage Release

If you use your own controller please think of the release of the output stage. The output stage release is a safety function so that in case of program failure the pumps stop immediately.

You need two digital signals for the output stage release. DOUT1 (pin 31 of the 37-pol. connector X1) gets first a high-level with pulse to low, duration 40 - 100 μs. After going high DOUT2 (pin 32 of the 37-pol. connector X1) needs within the next 100 ms a rect-signal in the range of 10 Hz and 1 kHz. (see fig. 1.3)

If you want to switch off this safety function move jumper JP5 about one position in direction to the board connector (see Figure 1.4).

![Figure 1.3: Signals for the output stage release](image_url)

Figure 1.3: Signals for the output stage release
1.7 Filling the Tank System

In order to avoid deposition of calcium, the tank system should only be filled with deionized or distilled water (total volume of about 40 litres). The system is filled by using the upper pump. Unscrew the left spigot nut of the upper pump and connect the delivered filling tube here. Close all four drain valves and open both connecting valves.

Connect the system cable of the tank system to the actuator. Connect the actuator to the required supply voltage. There is no need to connect a PC. Set the switch of the left SERVO-amplifier to the "Manual"-position and switch on the actuator. By means of the potentiometer, located below the switch, the pump is controlled. Fill each tank to a height of approx. 60 cm.

Using the delivered colorant you are able to fix the depth of colour of the water. This fluid contains beside the food colour also antifungal and germicidal substances. It can be added drop by drop to the water.

Open the four drain valves. Now there are approx. 28 litres of water inside the main reservoir. Close the drain valves again and fill the three tanks to height of approx. 25 cm. Now the system contains the required 40 litres of water.

Remove the filling tube and connect again the original tube to the pump by screwing the spigot nut. Make sure that the screwed connection is watertight.

1.8 Venting the Pressure Sensor Lines

After filling the system with water, venting of the pressure sensor lines inside the base of each tank is strictly recommended. To start with this operation, each tank has to be filled up to approx. 20 cm by controlling the pumps manually. At the rear side of each tank the pressure sensor is screwed normally to the tanks base. A drilled hole leading to the threaded connection of the pressure sensor is closed by a 4mm screw. This sealing is to be unscrewed until water is flowing from the bottom of the tank into the drilled hole inside its base. The screw is tightened again when water is flowing out of the tanks base at the screw location. Carrying-out this operation for each tank completes venting the pressure sensor lines.

1.9 Maintenance

The cover of the reservoir on which the three cylindrical tanks and the pumps are mounted must not be removed!

To avoid calcium deposition in the tank system or growth of algae, the system should only be used with distilled or deionized water. The Plexiglas can be cleaned with a standard cleaner for synthetic material and a soft, lint-free cloth.

The pumps and all other components do not require maintenance.

1.10 Transport

The filled tank system must not be transported! It can be emptied by means of one of the pumps. Open the four drain valves so that all the water can flow into the main reservoir. Unscrew the right spigot nut of the upper pump. It should be taken into account that a little bit of water flows out of the dismantled tube. Connect the delivered filling tube to the pump. The corresponding pump can be manually controlled on the actuator. There is no need to connect a PC. The remaining water below the siphon tube of the container can remain in the system during transportation. If the system is not used for a longer time, it is recommended to drain off the remaining water using the drain valve at the back, right-hand side of the system. Naturally the total volume of water can be drained off using this valve.

1.11 Start-Up and Test of Function

After all components of the DTS200 have been connected and adjusted as described in the previous sections, you can start-up the system according to the following
procedure:

1. Fill the tank system (1.7).

2. Turn off all components and connect the mains supply.

3. When Option 200-02 is used, turn on the personal computer and start the installation program SETUP.EXE from the enclosed CD for Windows 98SE/ME/2000/XP. Following a successful installation a desktop icon DTS200W32 is created for the controller program which may be started immediately.

4. When the electrical disturbance module (Option 200-05) is used, the potentiometers for the control signals and the sensors must be adjusted to 100%.

5. Turn on the actuator.

At this time the controller is still not started. Toggle the switches at the servo amplifiers to "Manual". Using the potentiometers located below the switches you can now test the function of the pumps. At the same time the sensors are tested by measuring the output voltages of the sensor amplifiers at the measurement outputs. The voltages have to be in a range between -10V and +10V.

With this, start-up of your DTS200 is complete if you don’t use the PC controller program of opt. 200-02.

To test the DTS200 with the controller program of opt. 200-02 please ensure that the switches of the servo amplifiers are turned to the position "Automatic". Select the item "Open Loop Control" from the menu "Run" so that the current readings from the system are displayed in the monitor window. The readings for the levels should be around zero in case of empty tanks. To check the pumps and sensors please select the item "Adjust Setpoint" from the menu "Run". Input 30 for the offset of Tank 1 and 15 for the offset of Tank 2. Acknowledge just with "OK" by pressing "Return". The pumps should fill Tank 1 and Tank 2 with a flow rate of 30 resp. 15 ml/s and the sensor readings should increase accordingly.

Differences between the water levels that are shown on the screen and those of the real plant are due to the fact that the sensors are not yet calibrated.

### 1.12 Locating Errors

First try to eliminate faults with the help of the following table. In case you cannot solve problems with your DTS200 by yourself, please contact us.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Possible cause</th>
</tr>
</thead>
<tbody>
<tr>
<td>The LEDs do not light up</td>
<td>Mains supply switched on? Check the connection to the mains supply. Check the fuse for the mains supply (rearpanel).</td>
</tr>
<tr>
<td>Green LEDs voltage supply do not light up</td>
<td>Check the fuses on the mains supply card.</td>
</tr>
<tr>
<td>Green LED of Servo-Module does not light up</td>
<td>Check the secondary fuse of the corresponding servo module.</td>
</tr>
<tr>
<td>Limit LEDs of the servos light up, without exceeding the limit of the liquid level (60 cm)</td>
<td>Check the lead connection between actuator and tank system.</td>
</tr>
<tr>
<td>Controller does not work</td>
<td>Check the connection between actuator and PC.</td>
</tr>
</tbody>
</table>
Practical Instructions
1 Introduction

2 Theoretical Foundations

3 The System "Three-Tank-System"
   3.1 The Plant
   3.1.1 The Controller
   3.1.2 The Signal Adaption Unit
   3.1.3 The Disturbance Modul
   3.2 Mathematical Model

4 Preparations for the Laboratory Experiment
   4.1 Controller Design
   4.2 Disturbance behaviour in case of a leak in tank 2
   4.3 PI–Control of the decoupled subsystems

5 Carrying-Out the Experiment
   5.1 Determination of Characteristics and Outflow Coefficients
   5.2 Behavior of Reference and Disturbance Variable without PI-control
   5.3 Behaviour of the Reference Variable with PI–Control

6 Evaluation of the Experiments

7 Reference

8 Solutions
8.1 Solutions of the Preparation Problems ............................................................... 8-1
8.2 Solutions to Carrying-Out the Experiment and the Questions .................................. 8-3
1 Introduction

The control of nonlinear systems, particularly multi-variable systems, plays a more and more important part in the scope of the advancing automation of technical processes. Due to the ever increasing requirements of process control (e.g. response time, precision, transfer behavior) nonlinear controller designs are necessary. One design method with practical applications was developed among others by Prof. Dr.-Ing. E. Freund and successfully used for trajectory control of robots. In this context it is about the so-called principle of nonlinear control and decoupling. The decoupling refers to the input/output behavior, i.e.: after successful decoupling every input affects only the corresponding output. Thus it is possible, to subdivide the multi-variable system into subsystems which are mutually decoupled. These subsystems are linear, i.e. simple to analyze. By means of determining freely choosable parameters, the transfer behavior of the subsystems can be designed subject to some restrictions.

During this laboratory experiment, the application of this principle to a three–tank–system with two inputs and three measurable state variables is to be examined. To that end, the step response and the disturbance behavior of the control system following the controller design is analyzed and compared with those of a standard PI-controlled system.

Because the theoretical foundations are formulated in a general manner, you should be able to apply the "Nonlinear System Decoupling and Control" even to more complicated systems after carrying out the experiment.
2 Theoretical Foundations

The nonlinear control and decoupling is applicable to nonlinear, time-variant multi-variable systems. The system description must have the following form:

\[
\frac{dx(t)}{dt} = A(x,t) + B(x,t) u(t) \quad \text{Eq.2.1}
\]

\[
y(t) = C(x,t) + D(x,t) u(t) \quad \text{Eq.2.2}
\]

with the initial conditions: \(x(t_0) = x_0\) and the definitions:

- \(x(t)\) : state vector with dimension \(n\) \((x_1(t), x_2(t), ..., x_n(t))\)
- \(u(t)\) : input vector of dimension \(m\)
- \(y(t)\) : output vector of dimension \(m\)
- \(A(x,t)\) : \((n \times 1)\)–matrix
- \(B(x,t)\) : \((n \times m)\)–matrix
- \(C(x,t)\) : \((m \times 1)\)–matrix
- \(D(x,t)\) : \((m \times m)\)–matrix

A fundamental requirement is that the numbers of system inputs and outputs must be identical.

Now a state feedback in the following form is introduced:

\[
u(t) = F(x,t) + G(x,t) w(t), \quad \text{Eq.2.3}
\]

where:

- \(F(x,t)\) : column vector of dimension \(m\)
- \(G(x,t)\) : in \((x,t)\) non-singular \((m \times m)\)–matrix
- \(w(t)\) : new \(m\)–dimensional input vector (setpoint vector)

Figure 2.1 shows graphically the closed loop system with \(D=0\):

Now one has to determine \(F\) and \(G\) such that the \(i\)–th input \(w_i\) \((i=1, ..., m)\) only affects the \(i\)–th output \(y_i\). Moreover, it is possible to adjust the dynamics of the decoupled subsystems by placing the poles.

The difference order \(d_i\) referring to the \(i\)–th output signal \(y_i\) is of great importance for the decoupling. The difference order indicates which total time derivative of the output \(y_i\) is directly affected by the input \(u_i\). It is a measure of the number of poles that can be placed.

![Figure 2.1: Block diagram with system decoupling](image-url)
arbitrarily.

In order to define the difference order, in the following the i–th component of the output vector is used instead of all the m components:

\[ y_i(t) = C_i(x,t) + D_i(x,t) u(t) \quad \text{Eq.2.4} \]

where:

- \( C_i(x,t) \) : i–th component of the vector \( C(x,t) \)
- \( D_i(x,t) \) : i–th row vector of the matrix \( D(x,t) \)

In case \( D_i(x,t) \neq 0 \), the difference order \( d_i \) is now defined as follows:

\[ d_i = 0 \]

In this case, as will be shown later, decoupling is possible using direct compensation.

In case \( D_i(x,t) = 0 \), the result is:

\[ d_i > 0 \]

In order to determine the exact value in this case, the i–th output equation:

\[ y_i = C_i(x,t) \quad \text{Eq.2.5} \]

is repeatedly differentiated with respect to time, which is shown exemplary in the following. The first differentiation yields:

\[ y_i'(t) = \frac{d}{dt} \left[ C_i(x,t) \right] + \frac{\partial}{\partial x} \left[ C_i(x,t) \right] \dot{x}(t) \quad \text{Eq.2.6} \]

where:

\[ \frac{\partial}{\partial x} \left[ C_i(x,t) \right] = (\frac{\partial}{\partial x_1} [C_i(x,t)], ..., \frac{\partial}{\partial x_m} [C_i(x,t)]) \quad \text{Eq.2.7} \]

Substituting Eq.2.1 into Eq.2.7 results in:

\[ \frac{\partial}{\partial x} \left[ N_A^{j-1} C_i(x,t) \right] B(x,t) \neq 0 \]
for the first time. This means, the input vector $u(t)$ directly affects the $j$–th differentiation of the output signal $y_i(t)$. With this, the difference order of the $i$–th subsystem results in:

$$d_i = j$$

In summary the difference order can be defined as follows:

- **a)** if $D_i(x,t) \neq 0$ : $d_i=0$  
  Eq.2.12
- **b)** if $D_i(x,t) = 0$ : $d_i=\min\{j:|\partial/\partial x (N_A^{j-1} C_i(x,t))|\} / B(x,t) \neq 0$  
  Eq.2.13

For simplification it is assumed in the following, that $d_i$ ($i=1, ..., m$) is constant with respect to all $x(t)$ and $t$.

Using the above mentioned method for all $m$ outputs of the nonlinear, time-variant system, the result is:

$$y^*(t)=C^*(x,t)+D^*(x,t)u(t) \quad Eq.2.14$$

where:

$$y^*(t)=(y_1^{(d_1)}(t), \ldots, y_m^{(d_m)}(t))^T$$

$C^*(x,t)$ : $m$–dimensional column vector

$D^*(x,t)$ : $(m \times m)$–matrix;

In this case the $i$–th component of $C^*(x,t)$ can be stated as follows:

$$C^*_i(x,t)=N_A^{d_i} C_i(x,t) \quad Eq.2.15$$

Moreover the $i$–th row vector of the matrix $D^*(x,t)$ is defined by:

$$D^*_i(x,t)=\begin{bmatrix} D_i(x,t) & \text{für } d_i=0 \\ [\delta x N_A^{d_i-1} C_i(x,t)]B(x,t) & \text{für } d_i \neq 0 \end{bmatrix} \quad Eq.2.16$$

On the assumption that the rank of the matrix $D^*(x,t)$ is constant, all its row vectors are unequal to the null vector.

Eq.2.14 is the initial equation to derive the decoupling matrices and the introduction of assignable dynamics of the decoupled subsystems.

If the relation:

$$u(t)=E_1(x,t)=-D^{-1} C^*(x,t) \quad Eq.2.17$$

is substituted in Eq.2.14, the result is:

$$y^*(t)=0 \quad Eq.2.18$$

This means, each of the $m$ outputs of the multi–variable system is decoupled. Extending the above relation in the following manner:

$$u(t)=E_1(x,t)+G(x,t)w(t) \quad Eq.2.19$$

where:

$$G(x,t)=D^{-1} L \quad Eq.2.20$$

$L=\text{diag}\{l_i\} , \quad (i=1,2,\ldots,m)$

yields:

$$y^*(t)=L w(t) \quad Eq.2.21$$

This form of feedback additionally allows a free adjustment of the input amplification of the $i$–th subsystem. In order to make it possible to influence the dynamic of the decoupled subsystem the above relation is changed as follows:

$$u(t)=F(x,t)+G(x,t)w(t) \quad Eq.2.22$$

where:

$$F(x,t)=F_1(x,t)+F_2(x,t) \quad Eq.2.23$$

$$F_2(x,t)=-D^{-1} M^*(x,t) \quad Eq.2.24$$

and

$$M^*(x,t) : m$–dimensional vector

Three - Tank - System DTS200

Practical Instructions 2-3
Substituting this relation in Eq.2.14 yields:

\[ y^*(t) = -M^*(x,t) + L w(t) \]  \hspace{1cm} \text{Eq.2.25}

The vector \( M^*(x,t) \) has to be determined such that the dynamic of the \( m \) subsystems can be changed using pole shifting and new couplings are avoided as well.

A suitable relation of the \( i \)-th component of the vector \( M^*(x,t) \) can be stated as

\[
M^*_i(x,t) = \begin{cases} 
0 & \text{für } d_i = 0 \\
\sum a_{ki} N^k C_i(x,t) & \text{für } d_i \neq 0 
\end{cases}
\]  \hspace{1cm} \text{Eq.2.26}

The constant factors \( a_{ki} \) are freely assignable with:

\[ i = 1, 2, \ldots, m \text{ und } k = 0, 1, \ldots, (d_i - 1) \]

In can be shown (substituting Eq.2.26 in Eq.2.25) that each component of Eq.2.25 with \( d_i \neq 0 \), using the above choice for \( M^*_i(x,t) \), can be described in the following form:

\[ y_i^{(d_i)}(t) + a_{i(d_i-1)} y_i^{(d_i-1)}(t) + \ldots + a_0 y_i(t) = l_i w(t) \]  \hspace{1cm} \text{Eq.2.27}

Accordingly each subsystem results in a linear differential equation of \( d_i \)-th order.

Thus, the nonlinear, time-variant, multi-variable system is decoupled. The transfer behavior of the subsystems with \( d_i \neq 0 \) is described by Eq.2.27.

In summary \( F \) and \( G \) are determined as follows:

\[
F(x,t) = -D^{-1}(x,t) \{ C^*(x,t) + M^*(x,t) \} \\
G(x,t) = D^{-1}(x,t) L
\]

Moreover it has to be considered that the difference order \( d_i \) is invariant with respect to a transformation of the system into different coordinate systems. The requirements of the determination of \( d_i \) however depend strongly on the choice of the basis system. This is also true with respect to the further steps of determining the algorithms of decoupling and control.
3 The System "Three-Tank-System"

3.1 The Plant

The following figure 3.1 shows the principal structure of the plant.

The used abbreviations are described in the following.

The plant consists of three plexiglas cylinders T1, T3 and T2 with the cross section $A$. These are connected serially with each other by cylindrical pipes with the cross section $S_n$. Located at T2 is the single so called nominal outflow valve. It also has a circular cross section $S_n$. The outflowing liquid (usually distilled water) is collected in a reservoir, which supplies the pumps 1 and 2. Here the circle is closed.

$H_{\text{max}}$ denotes the highest possible liquid level. In case the liquid level of T1 or T2 exceeds this value the corresponding pump will be switched off automatically. $Q_1$ and $Q_2$ are the flow rates of the pumps 1 and 2.

Technical data:

- $A = 0.0154 \text{ m}^2$
- $S_n = 5 \times 10^{-5} \text{ m}^2$
- $H_{\text{max}} = 62 \text{ cm} (\pm 1 \text{ cm})$
- $Q_{1\text{max}} = Q_{2\text{max}} = 100 \text{ mltr/sec} = 6.0 \text{ ltr/min}$

The remarkable feature of the used "Three-section-diaphragm-pumps" with fixed piston stroke (Pump 1 and 2) is the well defined flow per rotation. They are driven by DC motors.

For the purpose of simulating clogs or operating errors, the connecting pipes and the nominal outflow are equipped with manually adjustable ball valves, which allow to close the corresponding pipe.

For the purpose of simulating leaks each tank additionally has a circular opening with the cross section $S_l$ and a manually adjustable ball valve in series. The following pipe ends in the reservoir.

The pump flow rates $Q_1$ and $Q_2$ denote the input signals, the liquid levels of T1 and T2 denote the output signals, which have to be decoupled, of the controlled plant.

![Figure 3.1: Structure of the plant](image-url)
necessary level measurements are carried out by pressure sensors.

### 3.1.1 The Controller

Here a digital controller realized on a PC with a A/D-D/A plug-in card is used. The connection to the controlled plant is carried out by 12-bit converters.

The A/D–converter is required to read the liquid levels of the tanks T1, T3 and T2. It is initialized for the voltage range: –10V...10V. This results in a resolution of 4.88mV. Two D/A–converters are used to control the pumps. The following figure 3.2 shows the principal structure of the complete control loop.

The control software is designed such that the decoupled subsystems can be controlled additionally by PI–controllers. Furthermore a PI–control of the liquid levels of tank 1 and 2 is possible even without decoupling. The respective controller configuration can be seen at the screen.

Each controller adjustment as well as the setpoints can be changed on–line.

#### 3.1.2 The Signal Adaption Unit

The purpose of the signal adaption unit is to adapt the voltage levels of the plant and the converter to each other. Here the output voltages of the sensors are adapted to the maximum resolution of the A/D– converters and on the other hand the output voltage range of the D/A–converters is adapted the servo amplifier of the corresponding pump.

#### 3.1.3 The Disturbance Modul

Using switches, sensor failures of the level measurement can be simulated. The corresponding sensor output supplies a voltage which corresponds to the liquid level 0.

Alternatively it is possible to scale the measured liquid level between the real height (100%) and the height 0 (0%) using a potentiometer.

Two other potentiometers allow to simulate defects of the pumps. To do this the control signal can be reduced up to 0% of its original value, which is equivalent to decreasing the pump flow rate.

---

**Figure 3.2 : Control loop structure**
3.2 Mathematical Model

The following figure 3.3 once again shows the structure of the plant to define the variables and the parameters.

\( a_z \): outflow coefficients [ ]

\( h_i \): liquid levels [m]

\( Q_{ij} \): flow rates \([m^3/sec]\)

\( Q_{1,2} \): supplying flow rates \([m^3/sec]\)

\( A \): section of cylinder \([m^2]\)

\( S_l \): section of leak opening \([m^2]\)

\( S_n \): section of connection pipe \([m^2]\)

where \( i=1,2,3 \) and \((i,j)=[(1,3);(3,2);(2,0)]\)

If the balance equation:

\[ A \frac{dh}{dt} = \text{Sum of all occurring flow rates} \quad \text{Eq.3.1} \]

is used for all of the three tanks, the result is:

\[ A \frac{dh_1}{dt} = Q_1 - Q_{13} \quad \text{Eq.3.2} \]

\[ A \frac{dh_3}{dt} = Q_{13} - Q_{32} \quad \text{Eq.3.3} \]

\[ A \frac{dh_2}{dt} = Q_2 + Q_{32} - Q_{20} \quad \text{Eq.3.4} \]

The still unknown quantities \( Q_{13}, Q_{32}, \) and \( Q_{20} \) can be determined using the generalized Torricelli–rule. It can be stated as

\[ q = a_z S_n \text{sgn(}\Delta h\text{)} (2g |\Delta h|^{1/2}) \quad \text{Eq.3.5} \]

where:

\( g \): earth acceleration

\( \text{sgn}(z) \): sign of the argument \( z \)

\( \Delta h \): liquid level difference between two tanks connected to each other

\( a_z \): outflow coefficient (correcting factor, dimensionless, real values ranging from 0 to 1)

Figure 3.3: Principle structure to define the variables and parameters
So the result for the unknown quantities is:

\[ \begin{align*}
Q_{13} &= a z_1 S_n \text{sgn}(h_1 - h_3) \left(2g \left|h_1 - h_3\right|\right)^{1/2} \quad \text{Eq.3.6} \\
Q_{32} &= a z_3 S_n \text{sgn}(h_3 - h_2) \left(2g \left|h_3 - h_2\right|\right)^{1/2} \quad \text{Eq.3.7} \\
Q_{20} &= a z_2 S_n \left(2g h_2\right)^{1/2} \quad \text{Eq.3.8}
\end{align*} \]

With the vector definitions:

\[ \begin{align*}
h &= (h_1, h_2, h_3)^T \\
Q &= (Q_1, Q_2)^T \\
\Delta(h) &= (-Q_{13}, Q_{32} - Q_{20}, Q_{13} - Q_{32})^T / A \quad \text{Eq.3.11} \\
y &= (h_1, h_2)^T \quad \text{Eq.3.12}
\end{align*} \]

and the matrix definition:

\[ B = \frac{1}{A} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \quad \text{Eq.3.13} \]

the above system of equations can be transformed to:

\[ \begin{align*}
\frac{dh}{dt} &= \Delta(h) + B Q \quad \text{state equation} \quad \text{Eq.3.14} \\
y &= (h_1, h_2)^T \quad \text{output equation} \quad \text{Eq.3.15}
\end{align*} \]

With this the plant is described completely. Because the physical values \( h \) and \( Q \) are not accessible directly, converters and sensors are used. These are parts of the plant and are described using their characteristics (characteristics of pumps and sensors). Furthermore the values of the outflow coefficients are required to get a quantitative description of Eq.3.14 and to carry out the nonlinear decoupling in practice. Therefore, the neccessary data are determined interactively using the control program before the actual control starts.

The model described by Eq.3.14/Eq.3.15 is now used as the initial basis for the nonlinear decoupling of the Three–Tank–System. The corresponding controller design is carried out within the framework of the preparations. The solutions of these preparations are therefore the fundamental basis to carry out the experiment.
4 Preparations for the Laboratory Experiment

4.1 Controller Design

a) Determine the difference orders $d_1, d_2$ of the model as described by Eq.3.14/Eq.3.15.

Hint: Either use the general definition or transform Eq.3.14/Eq.3.15 directly to a form similar to Eq.2.14 with $D_i(x,t)$ unequal 0.

What is the behaviour of each subsystem after the decoupling?

b) Determine the vector $C^*(x,t)$ and the matrix $D^*(x,t)$. What is the rank of $D^*(x,t)$? Can the system be decoupled?

c) Determine the vector $F_1(x,t)$ and the matrix $G(x,t)$.

d) Determine the vector $M^*(x,t)$ with freely assignable parameters $a_{ki}$ using a suitable formulation. Determine the vector $F_2(x,t)$.

e) Determine the overall transfer behaviour of the decoupled subsystems. Roughly draw the step responses. What is the condition each of the real controller parameter $l_i$ and $a_{ki}$ has to meet so that the transfer behaviour of the subsystems is stable with an overall amplification of one?

4.2 Disturbance behaviour in case of a leak in tank 2

a) How do the model equations Eq.3.14/ Eq.3.15 change in case of a leak with the section $S_i$ and the outflow coefficient $a_{zi}$ in tank 2? The leak is assumed to be located at the level of the connection pipes.

b) What is the behaviour of the subsystems in case the controller design determined in 4.1 is used? To do this, formulate the corresponding differential equations of the systems. Are the subsystems still decoupled?

c) Compute the stationary output value of the 2nd subsystem (output signal $h_2$) in case of this disturbance and a constant input signal. Is there a steady state difference compared with the disturbance-free case? If so, what is its value?

4.3 PI–Control of the decoupled subsystems

The decoupled subsystems can be PI–controlled additionally in accordance with the following block diagram:

a) Compute the overall transfer function: $G_{gi}(s)=H_i(s)/WE_i(s)$

![Figure 4.1 : PI-control of the subsystems](image)
b) What is the condition the real parameters \( v_i, k_i \) and \( P_i \) have to fulfill so that each PI-controlled loop is stable with an overall amplification of one?

c) What is the overall transfer behaviour in case \( P_i = 0 \)?
5 Carrying-Out the Experiment

Start the controller program (further details will be found in the chapter Program Operation).

5.1 Determination of Characteristics and Outflow Coefficients

Before you start with the experiments the system has to be calibrated. To do this you select the menu item Parameters from the upper menu bar. A pulldown menu will appear offering further items for the system calibration.

a) Use the menu item Characteristic Liquid Level Sensor to determine the characteristic: liquid level \(\leq\) ADC input voltage by experiment. Control the pumps manually in this process (switches to "Manual").

b) Change to the determination of the outflow coefficients by means of the menu item Outflow Coefficients and use experimental measuring. Approximately 60 sec after the interactive start, the determination is performed by the program which computes the following equations:

\[
\begin{align*}
az_1 &= A \frac{h_1'}{(-S_n (2g (h_1-h_3))^{1/2}} \\
az_2 &= A \frac{(h_1'+h_2'+h_3')}{(-S_n (2g h_2)^{1/2})} \\
az_3 &= A \frac{(h_1'+h_3')}{(-S_n (2g (h_3-h_2))^{1/2})}
\end{align*}
\]

These equations can be derived from the modell equations Eq.3.14 with \(Q_1=Q_2=0\) and \(h_1\geq h_3\geq h_2\).

Record the outflow coefficients.

c) Use the menu item Characteristic Pump Flow Rate to determine the characteristic: DAC output voltage \(\leq\) pump supply using experimental measuring (Do not use the default values!).

Here, according to an amount of 8 DAC output voltages, the difference of levels which is measured in a time intervall of 20 sec is used to determine the pump flow rates (supplies).

Please follow exactly the instructions of the controller program.

After the determination of the plant parameters the menu item File/Save System Parameters may be used to store these values to the hard disk.

The window called "DTS-Monitor" is the central part of the screen. The upper frames display the selected control structure and the status of the measurement buffer used to store the measurements. The following controller structure contains current data of the plant and of the controller like liquid levels and flowrates.

A graphical evaluation of any experiment result is possible only if previously the storage of measurements was switched on.

Now open the connection valves and the nominal outflow valve! Start the controller by means of the menu item RUN.

5.2 Behavior of Reference and Disturbance Variable without PI-control

The following tasks require the control structure with a decoupling network (select the menu item Decoupling-Controller from the menu RUN). The accompanying parameter is adjustable by selection of the menu item Decoupling-Controller from the pulldown menu Parameter. As mentioned with the preparation tasks the parameters are set according to 'Decoup' \(=i_1=a_{ij}\) meaning that the amplification of the decoupled subsystems is equal to 1.
a) Step Responses

At first adjust the decoupling parameter to 'Decoup'=0.03 and choose the setpoints \(w_1=32\text{cm}\) (offset tank 1) and \(w_2=20\text{cm}\) (offset tank 2) (Use the menu item Adjust Setpoint from the menu RUN, enter the corresponding setpoint values with a constant signal shape and prompt with 'OK'). Wait until the steady state is reached. Now switch on the storing of measured data with a measuring time=240 sec (Use the menu item Measuring from the menu RUN, enter appropriate values and prompt with 'OK').

Now generate a step of the setpoint \(w_1\) (tank 1) to 37cm (Use the menu item Adjust Setpoint from the menu RUN, enter the corresponding setpoint value with a constant signal shape and prompt with 'OK'). After the steady state is reached (approximately after 50% of the measuring time) change the decoupling parameter to 'Decoup'=0.04 and generate a step of the setpoint \(w_2\) (tank 2) to 25cm (Use the menu item Adjust Setpoint from the menu RUN, enter the corresponding setpoint value with a constant signal shape and prompt with 'OK'). The measuring is finished when the status line of the measurement buffer displays the message "Completed". Using the menu View and its menu item Plot Recorded Data the measured data are displayable in a graphic representation on the screen. The graphic may be sent to a printer by using the menu item File/Print Plots. Furthermore the data may be saved in a file using the menu item File / Save Recorded Data.

Notes for the user:

The series of events, storing data and plot output will occur during all of the following experiments. Therefore it will not be mentioned explicitly in the following.

b) Disturbance behaviour in case of leaks

Adjust the decoupling parameter to 'Decoup'=0.2 and the setpoints to 40cm (tank 1) and 15cm (tank 2).

Store measurements (150 sec):
Create a leak in tank 3 by opening the corresponding valve. Close the valve after 30 sec. Now use 'Decoup'=0.1 sec and create a leak in tank 2. Wait until the steady states of the water levels are reached. Close the valve after this.

c) Disturbance behaviour in case of closed connection valve

Adjust the setpoints to 25cm (tank 1) und 20cm (tank 2) with 'Decoup'=0.1. Wait until the steady states of the water levels are reached.

Store measurements (210 sec):
Close the connection valve between tank 3 and tank 2. Wait until the water levels of tank 3 and tank 1 are the same. Now create a step of the setpoint \(w_1\) to 50cm.

At the end of the measuring do not forget to open the connection valve between tank 3 and tank 2 again.

d) Disturbance behaviour in case of sensor failure

If the disturbance modul is not present this experiment cannot be performed.

Choose the parameter 'Decoup'=0.2 and adjust the setpoints to 25cm (tank 1) and 20cm (tank 2).

Store measurements (100 sec):
Create a sensor failure in tank 1 until 50% of the measuring time is reached (Knob down, switch "Tank1" of the disturbance module). Then switch on the sensor again.
5.3 Behaviour of the Reference Variable with PI–Control

To obtain an overall amplification of 1 for the closed control loop the controller parameters $k_i$ are set equal to $v_i$.

a) Reference behaviour of the decoupled, PI–controlled subsystems

Choose the parameter 'Decoup'=0.05 and adjust the parameters of the PI-controller to $P_i = 0, K_i = 0.1 \text{sec}^{-1}$ (Use the menu item PI-Controller from the menu Parameter, enter the corresponding values and prompt with 'OK'). Now adjust the setpoints to 30cm (tank 1) and 20cm (tank 2). Then activate the PI–controller with decoupling mode (Select the menu item PI-Controller from the pulldown menu RUN). Wait until the steady states of the water levels are reached.

Store mesurements (120sec):
Create a step of the setpoint $w_1$ to 34cm.

b) Reference behaviour of the coupled, PI–controlled subsystems

Activate the decoupling controller by using the menu item Decoupling Controller. Adjust the setpoints to 30cm (tank 1) and 15cm (tank 2). Close the connection valve between tank 3 and tank 2. Wait until the water levels $h_2$ and $h_3$ are settled to steady state. Adjust the proportional portion of the PI-controller to 0 and select $k_i=0.5 \text{sec}^{-1}$.

In the following, only the reference behaviour of the water level $h_2$ will be considered.

Store measurements (450sec):
Now activate the PI–controller without decoupling. The water levels of tank 1 and tank 2 are now not any longer decoupled; they are controlled be the PI–controller.

Adjust the parameter $P_i=15 \text{sec}$ after approximately 50% of the measuring time.
6 Evaluation of the Experiments

ref. 5.2a):
Explain the characteristic behaviour and examine the time constants.

ref. 5.2b):
A leak in tank 3 does not change the steady states of the output variables (water levels of tank 1 and tank 2). A leak in tank 2 results in difference of the steady states. Why? Use this difference to calculate the opening of the leak valve. Use a value of 0.7 for the outflow coefficient of the leak valve.

Will a PI-controller settle the leak disturbance in tank 2? Use the differential equations of the system to show this on the assumption that the control loop is stable.

ref. 5.2c):
Explain the characteristic behaviour. Does the nonlinear controller design decouple the subsystems furthermore in case of such a failure?

ref. 5.2d):
Explain the characteristic behaviour. Does the nonlinear controller design furthermore decouple the output variable \( h_2 \)? What would be the result of a sensor failure in tank 3?

ref. 5.3a):
Will such parameters lead to stable or unstable control loops? Explain the system behaviour.

ref. 5.3b):
Is the control of the output variable \( h_2 \) stable if you use \( P_2=0 \) or \( P_2=10 \text{sec} \)? Show this by linearization of the plant model of tank 2 around the reference variable and by calculating the transfer function of the control loop.

Which is the order of the linearized plant in the nominal state?
7 Reference

/1/  E. Freund u. H. Hoyer:
    Das Prinzip der nichtlinearen
    Systementkopplung mit der Anwendung
    auf Industrieroboter
    Zeitschrift: "Regelungstechnik", Nr. 28 im
8 Solutions

8.1 Solutions of the Preparation Problems

ref. 4.1a) and 4.1b):

The equations Eq. 3.14 and Eq. 3.15 can be transformed to:

\[ y_1' = h_1' = (-Q_{13} + u_1)/A \quad \text{Eq. 8.1} \]
\[ y_2' = h_2' = (Q_{32} - Q_{20} + u_2)/A \quad \text{Eq. 8.2} \]

The result is that the input variable \( u_i \) (i=1,2) of each subsystem already affects the first differentiation of the corresponding output variable \( y_i \). It follows:

\[ d_1 = d_2 = 1 \quad \text{Eq. 8.3} \]

So the subsystems behave like a first order lag.

A comparison with Eq. 3.14 yields:

\[ C^* = \frac{1}{A} \begin{bmatrix} -Q_{13} \\ Q_{32} - Q_{20} \end{bmatrix} \quad \text{Eq. 8.4} \]
\[ D^* = \frac{1}{A} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad \text{Eq. 8.5} \]

\( D^{*-1} \) exists, because the rank \( \{D^*\} = 2 \). So the system can be decoupled.

ref. 4.1c):

with:

\[ D^{*-1} = A \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad \text{Eq. 8.6} \]

follows:

\[ E_i(x,t) = -A C^* \quad \text{Eq. 8.7} \]

Using \( L = \text{diag} \{l_i\} \) (i=1,2) the matrix \( G(x,t) \) can be determined:

\[ G(x,t) = A \begin{bmatrix} l_1 & 0 \\ 0 & l_2 \end{bmatrix} \quad \text{Eq. 8.8} \]

ref. 4.1d):

Because of \( d_i = 1 \), an additional parameter as a freely placeable pole can be introduced in each subsystem. Using the formulation:

\[ \mathbf{M}^* (x,t) = (a_{01} x_1, a_{02} x_2)^T : (a_{01}, a_{02}) \text{ arb.} \quad \text{Eq. 8.9} \]
yields:

\[ E_i(x,t) = -A \mathbf{M}^* (x,t) \quad \text{Eq. 8.10} \]

ref. 4.1e):

If Eq. 8.3 with \( \mathbf{F} = \mathbf{F}_1 + \mathbf{F}_2 \) is substituted in Eq. 8.41, the result is:

\[ h_i' = -a_{0i} h_i + l_i w_i \quad ; \quad i=1,2 \quad \text{Eq. 8.11} \]

This results in the transfer function \( G_i(s) \):

\[ G_i(s) = \frac{l_i}{s + a_{0i}} \quad \text{first order lag} \quad \text{Eq. 8.12} \]

with the step response \( \bar{U}_{G_i}(t) \):

\[ \bar{U}_{G_i}(t) = \frac{1}{a_{0i}} \]

Figure 8.1 : Step response
Every stable subsystem has an overall amplification of one if the following is true:

\[ l_i = a_{0i} > 0 \]  
Eq.8.13

**ref. 4.2a):**

A leak in tank 2 results in the following system equations:

\[ h_1' = \frac{(u_1 - Q_{13})}{A} \] (unchanged)  
Eq.8.14

\[ h_2' = \frac{(u_2 + Q_{32} - Q_{20} - Q_{21}(h_2))}{A} \]  
Eq.8.15

where:

\[ Q_{21}(h_2) = a_{z1} S_i (2g h_2)^{1/2} \]  
Eq.8.16

**ref. 4.2b):**

Using the controller design of 4.1):

\[ u_1 = Q_{13} + A (l_1 w_1 - a_{01} h_1) \]  
\[ u_2 = Q_{20} - Q_{32} + A (l_2 w_2 - a_{02} h_2) \]  
\[ \text{G}i = \frac{l_1}{(s + a_{01})} \]  
\[ \text{G}g = \frac{v_i}{l_1} \]  
\[ \text{G}i(s) = \frac{v_i l_1 (P_i s + 1) / NN}{2} \]  
\[ \text{NN} = s^2 + l_1 (k_i P_i + 1) s + l_1 k_i \]  
\[ \text{vi} = k_i \]  
\[ \text{Amplification: } Ver_i = 1 \]  
\[ \text{Time constant: } Ze_{ii} = \frac{1}{l_1 k_i} \]  
\[ \text{Damping: } Da_{ei} = \frac{1}{l_1 k_i} \]  

**ref. 4.2c):**

The final value of \( h_2 \) is given by the condition \( h_2' = 0 \). With \( a_{02} = l_2 \) the result of Eq.8.20 is:

\[ h_2' = 0 \Rightarrow h_2 = h_{2R} = w_2 + C_{12} (2w_2 + c(l_2)^{1/2})^{1/2}/l_2 \]  
Eq.8.21

where:

\[ c = g a_{z1} S_i (2g h_2)^{1/2} \]  
Eq.8.22

The difference \( d \) of the steady state results in:

\[ d = w_2 - h_{2R} = -c(l_2)^{1/2} + (c (2w_2 + c(l_2)^{1/2}))^{1/2}/l_2 \]  
Eq.8.23

**ref. 4.3a):**

With \( G_i(s) = \frac{l_1}{(s + a_{01})} \) and \( l_1 = a_{01} \) the result is:

\[ G_{gi}(s) = v_i l_1 (P_i s + 1) / NN \]  
Eq.8.24

where:

\[ NN = s^2 + l_1 (k_i P_i + 1) s + l_1 k_i \]  
Eq.8.25

**ref. 4.3b):**

Using the fundamental stability criterion, the control loops are stable if all of the parameters \( v_i, k_i, P_i \) and \( l_i \) have a positive value.

Using an overall amplification of one, the final value theorem of the laplace transformation results in:

\[ v_i = k_i \]  
Eq.8.26

**ref. 4.3c):**

Using \( P_i = 0 \) and \( v_i = k_i \) the result is a behaviour of a second order lag with the following characteristic coefficients:

Amplification: \( Ver_i = 1 \)  
Eq.8.27

Time constant: \( Ze_{ii} = \frac{1}{l_1 k_i} \)  
Eq.8.28

Damping: \( Da_{ei} = 1/2 \)  
Eq.8.29
8.2 Solutions to Carrying-Out the Experiment and the Questions

ref. 5.2a):

Figure 8.2 shows the graphic representation of the water levels and figure 8.3 the pump flow rates.
The adjusted time constants with 33.3sec and 25sec can be seen from the characteristics.
The output variables \( h_1 \) and \( h_2 \) are decoupled. A step of the setpoint \( w_1 \) does not change the behaviour of \( h_2(t) \) and vice versa.

ref. 5.2b):

Figures 8.4 and 8.5 show the resulting graphical representations. A leak in tank 2 was realized with decoupling mode additionally.

A leak in tank 3 does not change the output equations of the controller design (Eq.8.1 und Eq.8.2). \( h_1 \) and \( h_2 \) behave furthermore according to Eq.8.11.

According to chapter 8.1 a leak in tank 2 results in a difference \( d \) of the steady state of the reference variable \( w_2 \). If Eq.8.23 is solved for the opening of the leak valve \( S_1 \) the result is:
S_l = A \left( \frac{z}{g} \right)^{1/2}/a_z \quad \text{Eq.8.30}

where:

z = d^2 \frac{L^2}{2(w_2 - d)} \quad \text{Eq.8.31}

The interactively determined (chapter 5.1) outflow coefficient of tank 2 should have a value in a range of 0.6 ... 0.8. Using a measured value of 0.73 the result is a section of the leak opening as documented in figure 3.3.

For comparison: the real opening of the leak valve has a value of 0.5 cm² according to the technical data of the manufacturer.

Using the equation Eq.8.15, the parameter denotation from Figure 8.5 and a PI-controller for this subsystem the results are the following system differential equations:

\[ w_2' = v_2 \left( w_2 - k_2 h_2 + P_2 \right) \quad \text{Eq.8.32} \]

\[ h_2' = -a_z S_l \left( \frac{2g h_2}{A} \right)^{1/2} + l_2 w_2 - a_{02} h_2 \quad \text{Eq.8.33} \]

With \( v_2 = k_2 \) and under stationary condition (differentiations with respect to time are set to zero) the result is directly:

\[ w_2 = h_2 \quad \text{Eq.8.34} \]

So a leak disturbance in tank 2 is settled by a PI-controller.

ref. 5.2c):

Figures 8.6 and 8.7 show the resulting graphic representations.

If the connection valve between tank 3 and tank 2 is closed this stands for \( a_{z3} = 0 \). With that Eq.8.2 is:

\[ h_2' = (-Q_{20} + u_2)/A \quad \text{Eq.8.35} \]

With Eq.8.18 it follows:

\[ h_2' + a_{02} h_2 = l_2 w_2 - Q_{32}(h_3, h_2)/A \quad \text{Eq.8.36} \]

![Figure 8.6: State variables ref. experiment 5.2c)](image1)

![Figure 8.7: Flow rates ref. experiment 5.2c)](image2)
The output variable $h_3$ is now coupled with $h_1$. The system behaviour of the output variable $h_1$ is furthermore described by Eq.8.11.

This behaviour can be seen from the characteristic: $h_1$ behaves similar to a first order lag and does not change its steady state in case of such a disturbance. The steady state of the output variable $h_3$ depends on the water level of tank 3.

ref. 5.2d):

Figure 8.8 and 8.9 show the resulting graphic representations.

A total pressure sensor failure in tank 1 means the assignment $h_1=0$ in Eq.8.17. So the result is:

$$u_1 = Q_3^\wedge + A_1 w_1$$  \hspace{1cm} \text{Eq.8.37}

where:

$$Q_3^\wedge = Q_{13}(h_1=0,h_3) = -a_1 S_1 (2g h_3)^{1/2}$$  \hspace{1cm} \text{Eq.8.38}

If the above equation is substituted in Eq.8.1, the result is:

![Figure 8.8: State variables ref. experiment 5.2d)](image1)

![Figure 8.9: Flow rates ref. experiment 5.2d)](image2)
With this $h_1$ is coupled with the real level $h_3$. Using Eq.8.39, the steady state value of $h_1$ can be determined in dependency on $h_3$. According to the chosen parameters this value is over 60cm. The behaviour of the output variable $h_2$ is furthermore described by Eq.8.11.

A total pressure sensor failure results in an assignment $h_3=0$ in Eq.8.17 and Eq.8.18. So it can be shown accordingly that both of the output variables are coupled with the actual level $h_3$.

Figures 8.10 and 8.11 shows the graphic representations. All of the parameters are chosen with a positive value; because of that the control loops are stable. The decoupled subsystems behave like a first order lag with the damping:

$$\text{Dae} = \frac{0.05}{0.1} \frac{1}{2} = 0.35 < 1$$

Eq.8.40

The corresponding settling behaviour can be seen from the characteristic.
ref. 5.3b):  

Figures 8.12 and 8.13 show the resulting graphic representations.

Linearization of the system model of tank 2 around the operating point $H_2$ yields:

$$h_2' = (-\text{const } h_2 + q_2)/A \quad \text{Eq.8.41}$$

where:

$$\text{const} = 0.5 a z_2 S_n ((2g)/H_2)^{1/2} \quad \text{Eq.8.42}$$

If Eq.8.41 is transformed into laplace, with:

$$q_2 = v_2 (1/s + P_2) (w e_2 - h_2) \quad \text{Eq.8.43}$$

the result is the following transfer function $G(s)$:

$$G(s) = v_2 (1+P_2s)/((A s^2+(v_2 P_2+\text{const}) s+v_2) \quad \text{Eq.8.44}$$

All of the variables are defined as differences from the operating point.

The poles of $G(s)$ are:

$$s_{1,2} = -((\text{const}+v_2 P_2)/(2 A))^{1/2} \quad \text{Eq.8.45}$$

Using $P_2=0$ and $P_2=10\text{sec}$, each pole has a negative real part. Because of that the control loop is stable in both cases.

With $az_2=0.7$ the result at the operating point is: const=$2.0 \text{ cm}^2/\text{sec}$.

Because of that the real parts of the poles with $P_2=0$ are close to the boundary of the stability region. Using $P_2=10\text{sec}$, the real parts are strong negative accordingly.

This can be seen from the characteristic: With $P_2=0$ the behaviour is oscillating, with $P_2=10$ the behaviour is damped.

The order of the linearized plant is 3 in the nominal state.

So it can be noted that the nonlinear decoupling results in an order reduction of the control loops.
Program Operation

Date: 29.11.2007
# Three – Tank – System DTS200 Table of Contents

## Program Operation

1. **Program Start** ........................................................................................................ 1-1
2. **Menu 'File'** ........................................................................................................ 1-3
3. **Menu 'IO-Interface'** .......................................................................................... 1-4
4. **Menu 'Parameters'** ............................................................................................ 1-5
5. **Menu 'Run'** ........................................................................................................ 1-8
6. **Menu 'View'** ....................................................................................................... 1-11
7. **Menu 'Help'** ....................................................................................................... 1-15
8. **The Demo Version** ............................................................................................ 1-16
9. **Format of the Documentation File *.PLD** .......................................................... 1-18
10. **Format of the Calibration Data File DEFAULT.CAL** ......................................... 1-19
1 Program Operation

1.1 Program Start

The correct execution of the Windows program DTS200W32.EXE requires that the following files are available in the actual directory:

- DTS200W32.EXE
- DEFAULT.CAL
- DTS200.HLP
- dtssrv32.DLL
- DACSHNTB.DLL
- Timer32B.DLL
- plotBC32.DLL

The last three DLL’s may be contained in another directory with a public path (like Windows/System). But it is recommended to leave those DLL’s as they were installed in case more than one controller program of an amira laboratory setup is located at the same PC. The file DEFAULT.CAL is used to store the calibration data of the system. A detailed description of this file is given in chapter 1.10. The help file DTS200.HLP allows for operating the program without having this manual at hand. The function key F1 or a specific 'Help' button presented in a dialog is to be used to activate the corresponding help section.

After starting the application program DTS200W32.EXE on a PC with correctly installed adapter card (MF624) and adapter card driver (DRVMF624.sys) the main desktop appears on the screen as shown in the following figure:

![Figure 1.1: The main desktop of the DTS200 controller program](image)

As can be seen from the figure above, the desktop contains a main menu bar, the title “DTS-Monitor”, two panels indicating the “Active Controller” as well as the “Measurement” status and the current controller structure together with the three tanks. The status bar at the bottom may display hints for the menu items. Because there is no control-
 ler selected, the structure only shows two long arrows from left to right followed by a block, which represents the mapping of a flow rate control signal to a corresponding voltage for the servo amplifier of a pump. The other two blocks with arrows from right to left represent the mappings of the liquid level sensor voltages for tank 1 and 2 to the corresponding levels in cm. The numerical values shown are according to the default parameters from “DE-FAULT.CAL” and liquid levels of zero for each tank of the connected Three-Tank-System. Any other initial settings will display different values.

In case of any error message the following description of program steps during start-up may help to locate the error.

After starting the application program DTS200W32.EXE at first the system registry is searched for the key "amiraW32.INI" with the section "DTS200" to read the driver type and the card number of the PC adapter card used to control the tank system. If this section is missing it is initialized with the type "MF624.DRV" and a card number of “1”.

Then the program tries to load the dtssrv32.dll by calling a specific function of the Timer32B.dll. In case this function fails, the message "Service DLL 'DTSSRV32.DLL' is missing !" appears on the screen.

At next the program checks if it the dtssrv32.dll is a DEMO version. Only in this case the menu item "IO-Interface" is disabled, the menu item "Simulated System Errors" is enabled and the string "(Demo-Version)" is appended to the title of the monitor.

In the next step the program checks for valid driver type and card number of the PC adapter card. If this operation detects an invalid driver type the message "SelectDriver failed !" appears on the screen. Otherwise the check marks below the menu item "IO-Interface" are set accordingly.

At next the controller parameters as well as measured data are read from the dtssrv32.dll. Appropriate error messages appear on the screen if one of these operations fails. Then the controller parameters are set to default values (no controller, proportional factor=10, integral factor=1, decoupling factor=0.04, set point tank 1 constant 30 cm, set point tank 2 constant 15 cm) and transferred to the dtssrv32.dll.

Finally the program tries to open the adapter card driver and to start the timer handling the sampling period (50ms) of the controller. If this function fails the error message "StartTimer failed !" is sent to the screen. Additional errors may appear in case of problems with the adapter card itself or with its driver (see chapter 1.3).

A typical problem may be that the installed adapter card is different to MF624. In that case the user has just to activate the main menu item “IO-Interface” to select the correct card.
1.2 Menu 'File'

The pull down menu File provides functions to save and load recorded data or system parameters, to print plot windows or the desktop, to configure the printer as well as to terminate the program (see Figure 1.2).

![DTS200 W32]

**Figure 1.2:** The sub menu ‘File’

The function **Save Recorded Data** is enabled after the first measurement acquisition. It writes the measured data together with the selected system configuration like controller type and parameters to a file (documentation file). The file name is selected by the user by means of a file save dialog window. The extension of the file name has to be *.PLD.

The function **Load Recorded Data** opens a file open dialog window allowing the user to select a documentation file (*.PLD). The file data (from previous measurements) may be displayed in a graphic using the function **Plot File Data** from the menu View. The system configuration read from this file may be displayed in an information box using the function **Parameter From *.PLD File** from the menu View.

The function **Save System Parameters** stores the calibration data of the tank system (sensor characteristics, outflow coefficients and pump characteristics) to a file with an assignable name on the hard disk. (The file format is described with Calibration Data File). This option should be used to create the file DEFAULT.CAL when this file is missing. Otherwise a warning message will appear any time the program is started. This menu item is disabled when any controller is active.

The function **Load System Parameters** reads the calibration data of the tank system (sensor characteristics, outflow coefficients and pump characteristics) from a selectable file. This menu item is disabled when any controller is active.

The function **Print Plots** opens the Print Window Dialog to select one or several plot windows for print output. This dialog presents a list box containing the titles of all open plot windows. One or several windows may be selected for print output on the currently selected printer device (see **Print Setup ...**). A single window is printed on the upper half of a DIN A4 paper. The second window would be printed on the lower half of this paper. The following windows are printed on the next pages accordingly.

The function **Print Desktop** opens the standard printer setup dialog of Windows to select and adjust the output device for a screen copy of the current desktop.

The function **Print Setup ...** opens the Windows dialog to select a printer and to adjust its options.
Selecting the menu item **Exit** (equivalent to pressing Ctrl+F4) will terminate the program when the message box "Exit program ?" is confirmed. Another message box has to be confirmed in case a new sensor calibration has been carried-out without saving the system parameters.

1.3 **Menu 'IO-Interface'**

The pull down menu **IO-Interface** provides functions to manipulate the driver for the PC plug-in card (see Figure 1.3).

Each item represents a selectable 32Bit driver for the IO-adapter card, which may be installed in the PC. Selecting one of these means selecting the corresponding driver type. The program then searches for up to 4 entries of this driver type in the system’s registry. If more than one entry was found a dialog is presented to the user to select one of the entries that means one of the card numbers. If this driver could be opened successfully its type and card number are written to the program’s section in the system’s registry and a check mark will appear in front of the active menu item.

**Note:**
It is assumed that multiple driver entries correspond to multiple IO-adapter cards of the same type installed in the PC! Furthermore the DTS200 is controllable only if it is connected to the selected card!

On program start the selected driver and the card number are read from the program’s section in the system’s registry automatically.

Possible error messages:

"It's not possible to configure this driver"
There is no driver of the current type installed in the system (missing entry in the system’s registry).

"Only one card/driver is present and selected !!"
The system’s registry contains only one driver entry of the current type (this is just a warning).

"Could not open Dialog !!"
System error while creating a dialog box to select a driver entry.

"StartTimer failed !"  
In this case the selected driver could be opened, but the selected IO-adapter card could not be initialized (card is missing or defect).
1.4 Menu 'Parameters'

The pull down menu Parameters as shown in Figure 1.4 contains functions to adjust or calibrate system parameters.

The function Decoupling-Controller allows the user to adjust the common amplification factor of the decoupling network interactively. The resulting control loop behaves like a control loop with a proportional controller with an integrating plant. The coefficients \( li = a0i = \text{Decoup} \) (see theoretical backgrounds) are equal to the proportional amplification. This value is active in addition, when decoupling mode is activated for the PI controller.

The function PI-Controller opens a dialog (see Figure 1.6) allowing the user to adjust the proportional and the integral portion of the PI controller interactively. When 'decoupled' is enabled the same factor as mentioned above is adjustable meaning an active decoupling network in addition.
The menu item **Characteristic Liquid Level Sensor** starts a measuring to determine the sensor parameters such that a straight line defined by a gradient and an offset maps the sensor voltage in [Volt] to a real liquid level in [cm]. A dialog (see Figure 1.7) is opened automatically where the user is informed which actions are to be carried-out. At first the default calibration levels at 20 cm and 50 cm may be changed. Then the user is asked to control both pumps manually such that the calibration levels one after the other are reached in the 3 tanks. Reaching a specific calibration level is to be prompted by pressing 'measure 1' for the first and pressing 'measure 2' for the second calibration level. The function will then determine the accompanying sensor characteristics (gradient and offset as mentioned above) for each tank. After pressing 'measure 2' the dialog is terminated immediately. To abort the operations the 'Cancel' button can be used before 'measure 2' is activated. In that case the previous calibration data will be left unchanged. The same happens when the user activates the 'measure 1/2' buttons one after the other without really changing the tank levels by controlling the pumps manually.

![Figure 1.7: The dialog ‘Characteristic liquid level sensor’](image)

The menu item **Simulated System Errors** is available only for the Demo version of the program (see chapter 1.8).

The menu item **Outflow Coefficient** opens a dialog (see Figure 1.8) to start an automatic determination of the outflow coefficients. By means of a dialog the user is asked to adjust the maximum initial liquid levels of 60 cm in all tanks by controlling the pumps manually. After pressing the 'Start' button the determination of the outflow coefficients is carried out automatically. The liquid levels are measured the first time after a delay time of 60 sec. The second measurement is taken after another 15 sec. The progress in percentage of the measuring time is displayed in form of a gauge bar. At the end the outflow coefficients are calculated by substituting the level differences in the mathematical model of the tank system. The resulting outflow coefficients are then displayed in an information box which may be obtained in addition by activating **Outflow Coefficients** from the main menu item **View**. The measuring may be aborted at any time by pressing the 'Cancel' button.

![Figure 1.8: The dialog ‘Determination of outflow coefficients’](image)
The menu item **Characteristic Pump Flow Rate** starts an automatic determination of the pump characteristics by controlling the pumps with a constant signal for a specific time and measuring the resulting liquid level changes. By means of a dialog (see Figure 1.9) the user is asked to close all valves and to adjust the initial liquid levels of 10 cm in all tanks by controlling the pumps manually. Then the pump data are determined automatically by switching on the pumps for a specific period of time with a step-wise (12.5%) increasing control signal. The current liquid levels in the tanks 1 and 2 are measured after each step. The flow rate is calculated from the liquid difference and the on-switching time of the pumps for each control signal. The results are displayed in a table with nine rows. To terminate the dialog an ‘Ok’ button is presented at the end of the measurements. The ‘Cancel’ button may be used before the end of the measurements to abort the operations. But any result obtained in the mean time (any filled row of the table) is taken as a valid pump calibration base point.

![Figure 1.9: The dialog ‘Determination of pump flow rate’](image)

The menu item **Cross Section of Tanks** allows for displaying or editing the effective cross sections of the three tanks by means of a dialog (see Figure 1.10). Standard values are 154 [cm²] for tanks without and 149 [cm²] for tanks with an inner flow pipe.

![Figure 1.10: The dialog ‘Tank Dimensions’](image)
1.5 **Menu 'Run'**

The pull down menu **Run** from Figure 1.11 provides adjustments of the system configuration.

![Figure 1.11: The sub menu ‘Run’](image)

The function **Open Loop Control** installs an open loop control. When another controller was active previously the set point generators (for the pump flow rate) are reset. The appearing track bars are manually adjustable only when the shape of the corresponding set point or control signal is constant. The same is true for the following closed loop controller modes.

The function **Decoupling Controller** installs a control loop with decoupling network (P controller). When another controller was active previously the set point generators (for the liquid levels in the tanks 1 and 2) are reset.

The function **PI-Controller** installs a control loop with PI controller. When another controller was active previously the set point generators (for the liquid levels in the tanks 1 and 2) are reset.

The function **Reset PI-Controller** resets the controller variables from the previous sampling period.

The function **Stop Controller** switches off any of the selected controllers, disables the menu item **Adjust Set Point** and enables the menu items **Load/Save System Parameter** as well as the three items to control the system calibration.

The function **Adjust Set Point** is available only when one of the above system configurations is activated. This menu item opens a dialog window (see Figure 1.12) to adjust 2 set points or control signals, that means level set points (tank 1 and tank 2) for the controlled tank system or pump flow rates for the system in an open loop control. The signal shape (constant, rectangle, triangle, ramp, sine) together with an offset, an amplitude and a timing period are adjustable for each set point signal. The period is meaningless in case of a constant signal shape. The real signal is always formed by the sum of offset and amplitude. The selected signal becomes valid after terminating the dialog with 'Ok'.
The resulting signals from the different shape selections can be seen from the following Figure 1.13:

The function **Measuring** is always enabled. This menu item opens a window to adjust the measuring time and to assign trigger conditions to start recording the measurements. Figure 1.14 shows this window. The measuring time in seconds is entered to the right to the title 'Total Time [s]'. When 'Slope' is set to 'no trigger' measurement recording is started directly after closing the window using the 'Ok' button. The currently adjusted system configuration like controller type and parameters are stored in a data structure, which may be saved in a documentation file (*.PLD).
The trigger signal for conditional measuring ('Slope:' is set 'positive' or 'negative') is selected below the title 'Trigger Channel'. The measurement recording starts after this signal raises above or falls below, depending on the settings of 'Slope', the limit value 'Trigger Value'. In addition 'Prestore:' allows for adjustment of a time range for recording measurements before the trigger condition is valid. This time has always to be shorter than the adjusted measuring time.

Figure 1.14: The dialog ‘Setup Measuring Function’
1.6 Menu 'View'

The pull down menu View from Figure 1.15 provides functions for graphic representations of currently recorded measurements or of data from the documentation file (*.PLD).

![Figure 1.15: The sub menu ‘View’](image)

The menu item Plot Measured Data opens a dialog window to select the data, which are to be displayed in a graphic representation. The selectable data are (see also Figure 1.16):

- Liquid levels (3 curves)
- Liquid levels and set points (5 curves)
- Pump flow rates (3 curves, includes nominal outflow)
- Pump control signals (2 curves)
- Levels and flow rates (5 curves)

![Figure 1.16: The dialog ‘Select Plot Data’](image)

Pressing the 'Ok' button will display the selected graphic immediately (see Figure 1.17 as an example) while 'Cancel' will abort this dialog and return to the main window. An open plot window may be stored to the clipboard or to an assignable Windows Meta File by activating corresponding items from its system menu.
The plot window from above results from the following settings:

PI-controller with decoupling.
Level set point for tank 1 is a square signal with offset 30cm, amplitude 2cm and a period of 50sec.
Level set point for tank 2 is a sine signal with offset 15cm, amplitude 1cm and a period of 50sec.
Triggered measuring for a positive trigger slope of the level set point for tank 1, trigger level 30cm and 100sec total time.

The plot window is shown with an open window menu, which enables the user to save the windows content to a WMF file or to the clipboard.

Further manipulations of the plot window content are selectable by using the mouse as follows:

Moving the cursor inside the axis area while holding the left mouse button pressed opens a zoom rectangle for the plot area. To return to the original scaling the left mouse button is to be clicked outside of the axis area.
Moving the cursor outside of the axis area and outside of the line style table area and pressing the right mouse button opens the plot layout dialog (see Figure 1.18).
Moving the cursor into the axis area or into the line style table area and pressing the right mouse button opens the curves layout dialog (see Figure 1.19):

Moving the cursor onto the x-axis or y-axis and pressing the right mouse button opens the corresponding axis layout dialog (see Figure 1.20)
Figure 1.20: The dialog ‘Attributes of the X axis’:

The menu item Plot File Data operates similar to the item Plot Measured Data. But the data are loaded from the file selected by the menu item Load Plot Data from the menu File.

The menu item Parameter From *.PLD File generates an information box (see Figure 1.23) displaying the controller type and parameters read from the currently selected documentation file (*.PLD). This menu item is enabled only when such a file was loaded successfully.

Figure 1.21: The dialog ‘PLD File Information’

The menu item Characteristic Liquid Level Sensor displays a graphic representation of the 3 sensor characteristics.

The menu item Characteristic Pump Flow Rate displays a graphic representation of the 2 pump characteristics.

The menu item Outflow Coefficients displays the outflow coefficients in an information box.

The menu item Characteristic outflow displays a graphic representation of the nominal outflow calculated with respect to the mathematical model for tank 2. It is reserved for future use.
1.7 Menu ‘Help’

The pull down menu Help as shown in Figure 1.22 provides functions to control the Windows help function and to obtain general information about the program.

Figure 1.22: The sub menu ‘Help’

The menu item Contents displays the contents of the help file DTS200.HLP, while Search for Help On ... searches for keywords contained in this help file. The item How to Use Help opens the Use Help Dialog of Windows.

Activating the menu item About opens an information box displaying the program version, the copyright and the IO-adapter card requirements (see Figure 1.23).

Figure 1.23: The dialog ‘Program Info’
1.8 The Demo Version

The demo version of the program DTS200W32.EXE is indicated by the title "DTS - Monitor (Demo-Version)" in the monitor window (see Figure 1.24). It operates with a mathematical model instead of reading sensor signals from the IO-adapter card and writing control signals to this card. Besides the functions to control the calibration and to select the IO-Interface all of the menu items are available.

But two special features are introduced. At first the simulation runs 10 times faster than the real system. At second the menu item Simulated System Errors (see Figure 1.25) from the main menu Parameter is provided to simulate signal errors for the three level sensors and the 2 pumps. Single or multiple leaks and clogs may be defined in addition. The errors are defined in the range from 0 to 100% meaning an undisturbed signal for 0% and a complete missing signal for 100%.

During calculation of the mathematical model the errors are considered in the range from 0 to 1. This results for tank 1 in example:

\[
A_{\text{tank}} \frac{dh}{dt} = Q_{\text{in}} - Q_{\text{out}} - Q_{\text{Leak}} \\
Q_{\text{in}} = Q_{\text{pump1}} (1 - \text{Pump1 Error}) \\
Q_{\text{out}} = c_{\text{valve}} (1 - \text{Valve1 Error}) \sqrt{2g} |h_1 - h_3| \\
Q_{\text{Leak}} = c_{\text{valve}} (1 - \text{Valve1 Error}) \sqrt{2gh_1} \\
h_{\text{Sensor}} = h_1 (1 - \text{Sensor1 Error}) \\
\]

where

\( h_1, h_3 = \) real liquid levels of the tanks 1 and 3
$A_{tank} \cdot A_{valve} =$ cross sections of tank resp. valve

c$_1 =$ outflow coefficient connection valve 1 -> 3

Figure 1.25: The dialog ‘System Error Simulation’
1.9 Format of the Documentation File *.PLD

Measured data stored in a data file are reloadable and may be output in a graphic representation. In addition the system settings (CTRLSTATUS), which were active during the start of the data acquisition are stored in this file. They are displayable in a separate window.

The data file contains data in binary format stored in the following order:

- The structure PROJEKT PRJ,
- The structure CTRLSTATUS,
- The structure DATASTRUCT,
- The data array with float values (4 bytes per value)

The size of the data array is defined in the structure DATASTRUCT. With the DTS200 the number of the stored channels is always 8 (the length of the measurement vector is 8, i.e. equal to 32 bytes). The vector contains the following signals:

- the set point for tank 1 (in cm),
- the set point for tank 2 (in cm),
- the measured liquid level from tank 1 (in cm),
- the measured liquid level from tank 2 (in cm),
- the measured liquid level from tank 3 (in cm),
- the control signal for pump 1 (in ml/s),
- the control signal for pump 2 (in ml/s)
- the calculated nominal outflow q20 (in ml/s)

The number of the stored measurement acquisitions (vectors) depends on the adjusted values for the sampling period and the measuring time. The maximum number of sampled vectors is 1024. The time distance between two successive acquisitions is an integral multiple of the sampling period used by the controller.

The structures mentioned above are defined as follows:

```
struct PROJECT {
    char number[10]; // P335
    char name[10]; // DTS200
    char Titel[20]; // DTS200 – Monitor
    char Version[10]; // last version
    char Date[10]; // last date
    char Dummy[10]; // Reserve
};
struct CTRLSTATUS {
    SHORT OpenLoop; // open loop controller type
    SHORT PX; // decoupling controller type
    SHORT PI; // PI-controller type
    SHORT DeCoupled; // (PI-) controller type with decoupling
    double Decoup; // decoupling controller parameter
    double Kp; // PI-controller parameter (proportional)
    double Ki; // PI-controller parameter (integral)
    long timeofmeasure; // date and time of the measurement
};
```
struct DATASTRUCT {
    SHORT nchannel;      // number of components in the measurement-vector
    SHORT nvalues;       // number of measurement-vectors (samples)
    float deltatime;     // time period between samples
};

1.10 Format of the Calibration Data File DEFAULT.CAL

The calibration data file DEFAULT.CAL which may be loaded or saved by Load System Parameters resp. Save System Parameters from the main menu item File contains the sensor characteristics (gradient and offset), the outflow coefficients as well as nine base points for each pump characteristic in an ASCII format. When this file is missing during starting the program DTS200W32, it may be created with the default calibration data just by activating Save System Parameters without carrying-out any calibration. In this case the result would be as follows:

-3.42857 [gradient]
30.8571 [offset]
-3.42857 [gradient]
30.8571 [offset]
-3.42857 [gradient]
30.8571 [offset]
0.5 0.6 0.5 [Outflow Coef.]
0 [ml/s at 0.0 % Controller output]
12.5 [ml/s at 12.5 % Controller output]
25 [ml/s at 25.0 % Controller output]
37.5 [ml/s at 37.5 % Controller output]
50 [ml/s at 50.0 % Controller output]
62.5 [ml/s at 62.5 % Controller output]
75 [ml/s at 75.0 % Controller output]
87.5 [ml/s at 87.5 % Controller output]
100 [ml/s at 100.0 % Controller output]
0 [ml/s at 0.0 % Controller output]
12.5 [ml/s at 12.5 % Controller output]
25 [ml/s at 25.0 % Controller output]
37.5 [ml/s at 37.5 % Controller output]
50 [ml/s at 50.0 % Controller output]
62.5 [ml/s at 62.5 % Controller output]
75 [ml/s at 75.0 % Controller output]
87.5 [ml/s at 87.5 % Controller output]
100 [ml/s at 100.0 % Controller output]
149 149 149 [Tank cross sections]
Technical Data

Date: 13. December 2009

(Technical data are subject to change)
1 Technical Data

1.1 Tank system, pumps, sensors .................................................. 1-1
1.2 Actuator ................................................................................. 1-2
  1.2.1 Servo modules ................................................................. 1-2
  1.2.2 Mains supply ................................................................. 1-3
  1.2.3 Signal adaption unit ....................................................... 1-3
  1.2.4 Module "POWER SERVO" .................................................. 1-3
  1.2.5 Measurement Outputs of the Module "Sensor" ..................... 1-3
  1.2.6 Signal distribution (Rear Panel) ........................................ 1-3
  1.2.7 Controller ....................................................................... 1-4
  1.2.8 Electrical disturbance module (Option 200-05) .................. 1-4
1.3 Pin reservations of the plug connections ................................. 1-4
  1.3.1 Connection Actuator to the PC Plug-in Card ....................... 1-4
  1.3.2 Connection Actuator to the Three-Tank-System ................ 1-5
# Technical Data

## 1.1 Tank system, pumps, sensors

<table>
<thead>
<tr>
<th>Dimension Sizes and Weight</th>
<th>Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length</td>
<td>1300</td>
<td>mm</td>
</tr>
<tr>
<td>Depth</td>
<td>360</td>
<td>mm</td>
</tr>
<tr>
<td>Height</td>
<td>880</td>
<td>mm</td>
</tr>
<tr>
<td>Weight</td>
<td>40</td>
<td>kg</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reservoir</th>
<th>Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length</td>
<td>1210</td>
<td>mm</td>
</tr>
<tr>
<td>Depth</td>
<td>360</td>
<td>mm</td>
</tr>
<tr>
<td>Height</td>
<td>150</td>
<td>mm</td>
</tr>
<tr>
<td>Capacity ca.</td>
<td>55</td>
<td>l</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cylinder tank</th>
<th>Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diameter outside</td>
<td>150</td>
<td>mm</td>
</tr>
<tr>
<td>Diameter inside</td>
<td>140</td>
<td>mm</td>
</tr>
<tr>
<td>Diameter inner flow pipe</td>
<td>25</td>
<td>mm</td>
</tr>
<tr>
<td>Height incl. socket and cover</td>
<td>720</td>
<td>mm</td>
</tr>
<tr>
<td>max. liquid level</td>
<td>630</td>
<td>mm</td>
</tr>
<tr>
<td>Capacity ca.</td>
<td>9</td>
<td>l</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ball Valves</th>
<th>Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection pipes</td>
<td>0.5</td>
<td>cm²</td>
</tr>
<tr>
<td>Nominal outflow</td>
<td>0.5</td>
<td>cm²</td>
</tr>
<tr>
<td>Leakage openings</td>
<td>0.5</td>
<td>cm²</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pumps</th>
<th>Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>DC-motor with &quot;three chamber diaphragm pump&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rated voltage</td>
<td>12</td>
<td>V</td>
</tr>
<tr>
<td>Rated current (open flow)</td>
<td>1.4</td>
<td>A</td>
</tr>
<tr>
<td>Rated current (max.)</td>
<td>4.5</td>
<td>A</td>
</tr>
<tr>
<td>Flow rate (open flow)</td>
<td>7</td>
<td>l/min</td>
</tr>
<tr>
<td>Pressure</td>
<td>1.4</td>
<td>bar</td>
</tr>
<tr>
<td>Weight</td>
<td>1.8</td>
<td>kg</td>
</tr>
<tr>
<td>Length</td>
<td>197</td>
<td>mm</td>
</tr>
<tr>
<td>Depth</td>
<td>127</td>
<td>mm</td>
</tr>
<tr>
<td>Height</td>
<td>113</td>
<td>mm</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sensors</th>
<th>Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Principle of measurement: plate capacitor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pressure range</td>
<td>0...0.1</td>
<td>bar</td>
</tr>
<tr>
<td>Supply voltage</td>
<td>12...30</td>
<td>V</td>
</tr>
<tr>
<td>Output signal nominal</td>
<td>4...20</td>
<td>mA</td>
</tr>
</tbody>
</table>
## Technical Data

### Sensors

<table>
<thead>
<tr>
<th>Principle of measurement: plate capacitor</th>
<th>Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>max. allowable burden</td>
<td>37.5* (Ub-12)</td>
<td>Ohm</td>
</tr>
<tr>
<td>max. current consumption</td>
<td>23</td>
<td>A</td>
</tr>
<tr>
<td>Characteristic</td>
<td>linear</td>
<td></td>
</tr>
<tr>
<td>Deviation from linearity</td>
<td>&lt; 0.5</td>
<td>%</td>
</tr>
<tr>
<td>Repeatability and hysteresis</td>
<td>&lt;0.02</td>
<td>%</td>
</tr>
<tr>
<td>Response time (63%)</td>
<td>5</td>
<td>ms</td>
</tr>
</tbody>
</table>

### Dimension Sizes and Weight

<table>
<thead>
<tr>
<th></th>
<th>Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length</td>
<td>471</td>
<td>mm</td>
</tr>
<tr>
<td>Depth</td>
<td>310</td>
<td>mm</td>
</tr>
<tr>
<td>Height</td>
<td>147</td>
<td>mm</td>
</tr>
<tr>
<td>Weight</td>
<td>7.5</td>
<td>kg</td>
</tr>
</tbody>
</table>

### Inputs

<table>
<thead>
<tr>
<th>Pump supply voltages</th>
<th>0 ... +12</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensor supply voltages</td>
<td>18</td>
<td>V</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Liquid level from sensors</th>
<th>Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Current range</td>
<td>4 ... 14</td>
<td>mA</td>
</tr>
<tr>
<td>- Height range</td>
<td>0 ... 63</td>
<td>cm</td>
</tr>
<tr>
<td>- Resolution</td>
<td>0.16</td>
<td>mA/cm</td>
</tr>
</tbody>
</table>

### 1.2 Actuator

### 1.2.1 Servo modules

<table>
<thead>
<tr>
<th>Voltage supply</th>
<th>15</th>
<th>V~</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fuse (15 V~)</td>
<td>2.0</td>
<td>A M</td>
</tr>
<tr>
<td>Sensor signal overflow protection</td>
<td>1 ... +5.6</td>
<td>V</td>
</tr>
<tr>
<td>Control signals</td>
<td>0 ... +10</td>
<td>V</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Pump voltage supply (PWM)</th>
<th>+15</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rated current</td>
<td>1.6</td>
<td>A</td>
</tr>
<tr>
<td>Dyn. peak current</td>
<td>4.5</td>
<td>A</td>
</tr>
<tr>
<td>Overflow protection</td>
<td>TTL signal</td>
<td></td>
</tr>
<tr>
<td>0 = Overflow</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 = No overflow</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Switch position</td>
<td>TTL signal</td>
<td></td>
</tr>
<tr>
<td>0 = Automatic</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 = Manual</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
1.2.2 Mains supply

<table>
<thead>
<tr>
<th>Mains supply</th>
<th>Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input voltage</td>
<td>230</td>
<td>V~</td>
</tr>
<tr>
<td>2 x Fuse primary</td>
<td>100</td>
<td>mA T</td>
</tr>
<tr>
<td>Frequency</td>
<td>50/60</td>
<td>Hz</td>
</tr>
<tr>
<td>Output voltages</td>
<td>±15</td>
<td>V</td>
</tr>
<tr>
<td>(short-circuit protected)</td>
<td>+5</td>
<td>V</td>
</tr>
</tbody>
</table>

1.2.3 Signal adaption unit

<table>
<thead>
<tr>
<th>Inputs signal adaption unit</th>
<th>Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 sensor signals</td>
<td>1.1 ... 3.8</td>
<td>V</td>
</tr>
<tr>
<td>(at 274 Ohm)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 control signals</td>
<td>-10...+10</td>
<td>V</td>
</tr>
<tr>
<td>TTL-signal release control signal</td>
<td>0 ... +5</td>
<td>V</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Outputs signal adaption unit</th>
<th>Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 liquid levels</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- voltage range</td>
<td>+10 ... -10</td>
<td>V</td>
</tr>
<tr>
<td>- level range</td>
<td>0 ... 70</td>
<td>cm</td>
</tr>
<tr>
<td>- resolution</td>
<td>0.286</td>
<td>V/cm</td>
</tr>
<tr>
<td>2 control sig. for servos</td>
<td>0 ... +10</td>
<td>V</td>
</tr>
</tbody>
</table>

1.2.4 Module "POWER SERVO"

<table>
<thead>
<tr>
<th>Mains supply</th>
<th>Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input voltage</td>
<td>230</td>
<td>V~</td>
</tr>
<tr>
<td>Fuse primary</td>
<td>630</td>
<td>mA T</td>
</tr>
<tr>
<td>Frequency</td>
<td>50/60</td>
<td>Hz</td>
</tr>
<tr>
<td>Output voltage</td>
<td>2*15</td>
<td>V~</td>
</tr>
<tr>
<td>Rated current</td>
<td>2*2.6</td>
<td>A</td>
</tr>
</tbody>
</table>

1.2.5 Measurement Outputs of the Module "Sensor"

<table>
<thead>
<tr>
<th>Measurement jacks</th>
<th>Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tank 1 ... Tank 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(= Outputs of sig. adapt. unit or disturbance mod.)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Range of liquid levels</td>
<td>+10 ... -10</td>
<td>V</td>
</tr>
<tr>
<td>- Nominal value for 0cm</td>
<td>+9</td>
<td>V</td>
</tr>
<tr>
<td>- Nominal value for 60cm</td>
<td>-9</td>
<td>V</td>
</tr>
<tr>
<td>Q1 and Q2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(= Outputs of PC-controller)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Flow rate 0 ml/s</td>
<td>-10</td>
<td>V</td>
</tr>
<tr>
<td>- Flow rate 100 ml/s</td>
<td>+10</td>
<td>V</td>
</tr>
</tbody>
</table>

1.2.6 Signal distribution (Rear Panel)

<table>
<thead>
<tr>
<th>Inputs signal distribution unit (37-pol. Connector)</th>
<th>Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>D/A0 (Pin 20) Control signal servo 1</td>
<td>-10 ... +10</td>
<td>V</td>
</tr>
<tr>
<td>D/A1 (Pin 21) Control signal servo 2</td>
<td>-10 ... +10</td>
<td>V</td>
</tr>
<tr>
<td>DO1 (Pin 31) Pulse (release output stage) HiIgh-level with pulse to low, duration 40-100 μs</td>
<td>TTL-Signal</td>
<td></td>
</tr>
<tr>
<td>DO2 (Pin 32) Rect (release output stage) min. 10 Hz, max. 1 kHz</td>
<td>TTL-Signal</td>
<td></td>
</tr>
</tbody>
</table>
### Technical Data

#### Three - Tank - System DTS200

<table>
<thead>
<tr>
<th>Outputs signal distribution (37-pol. connector)</th>
<th>Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>A/D0 (Pin 1) Level tank 1</td>
<td>-10 ... +10V</td>
<td>V</td>
</tr>
<tr>
<td>A/D1 (Pin 2) Level tank 2</td>
<td>-10 ... +10V</td>
<td>V</td>
</tr>
<tr>
<td>A/D2 (Pin 3) Level tank 3</td>
<td>-10 ... +10V</td>
<td>V</td>
</tr>
</tbody>
</table>

### 1.3 Pin reservations of the plug connections

#### 1.3.1 Connection Actuator to the PC Plug-in Card

<table>
<thead>
<tr>
<th>Signal Actuator 37pol. DSUB</th>
<th>PC Plug-in card MF624</th>
<th>Pin Denotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>X1 (37 pol. DSUB)</td>
<td></td>
<td>Signal denotation</td>
</tr>
<tr>
<td>AGND</td>
<td>9</td>
<td>AGND</td>
</tr>
<tr>
<td>AGND</td>
<td>8</td>
<td>AD7 (not used)</td>
</tr>
<tr>
<td>AGND</td>
<td>7</td>
<td>AD6 (not used)</td>
</tr>
<tr>
<td>AGND</td>
<td>6</td>
<td>AD5 (not used)</td>
</tr>
<tr>
<td>AGND</td>
<td>5</td>
<td>AD4 (not used)</td>
</tr>
<tr>
<td>Level tank 1</td>
<td>1</td>
<td>AD0</td>
</tr>
<tr>
<td>Level tank 2</td>
<td>2</td>
<td>AD1</td>
</tr>
<tr>
<td>Level tank 3</td>
<td>3</td>
<td>AD2</td>
</tr>
<tr>
<td>Pulse (servo release)</td>
<td>31</td>
<td>DOUT1</td>
</tr>
<tr>
<td>Rect (servo release)</td>
<td>32</td>
<td>DOUT2</td>
</tr>
<tr>
<td>DGND</td>
<td>29</td>
<td>GND</td>
</tr>
<tr>
<td>Control signal servo 1</td>
<td>20</td>
<td>DA0</td>
</tr>
<tr>
<td>Control signal servo 2</td>
<td>21</td>
<td>DA1</td>
</tr>
</tbody>
</table>

### 1.2.7 Controller

<table>
<thead>
<tr>
<th>PC with Opt. 200-02</th>
<th>A/D-D/A-card for PC</th>
</tr>
</thead>
<tbody>
<tr>
<td>MF624</td>
<td>MF624</td>
</tr>
</tbody>
</table>

#### Inputs

<table>
<thead>
<tr>
<th>Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>-10 ... 10</td>
<td>V</td>
</tr>
</tbody>
</table>

#### Outputs

<table>
<thead>
<tr>
<th>Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>-10 ... +10</td>
<td>V</td>
</tr>
</tbody>
</table>

### 1.2.8 Electrical disturbance module

**Option 200-05**

<table>
<thead>
<tr>
<th>Option 200-05</th>
<th>Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensor signals for liquid levels - scaleable - can be switched off</td>
<td>0 - 100</td>
<td>%</td>
</tr>
<tr>
<td>Control signals - scaleable</td>
<td>0 - 100</td>
<td>%</td>
</tr>
</tbody>
</table>
## 1.3.2 Connection Actuator to the Three-Tank-System

<table>
<thead>
<tr>
<th>Pin- No.</th>
<th>Reservation</th>
<th>Pin- No.</th>
<th>Reservation</th>
<th>Pin- No.</th>
<th>Reservation</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>Signal sensor 1</td>
<td>b1</td>
<td>n.c.</td>
<td>c1</td>
<td>n.c.</td>
</tr>
<tr>
<td>a2</td>
<td>Signal sensor 3</td>
<td>b2</td>
<td>n.c.</td>
<td>c2</td>
<td>n.c.</td>
</tr>
<tr>
<td>a3</td>
<td>Signal sensor 2</td>
<td>b3</td>
<td>15 V</td>
<td>c3</td>
<td>n.c.</td>
</tr>
<tr>
<td>a4</td>
<td>Screen</td>
<td>b4</td>
<td>Screen</td>
<td>c4</td>
<td>n.c.</td>
</tr>
<tr>
<td>a5</td>
<td>Pump 1 +</td>
<td>b5</td>
<td>Pump 1 -</td>
<td>c5</td>
<td>n.c.</td>
</tr>
<tr>
<td>a6</td>
<td>Pump 2 +</td>
<td>b6</td>
<td>Pump 2 -</td>
<td>c6</td>
<td>n.c.</td>
</tr>
<tr>
<td>a7</td>
<td>n.c.</td>
<td>b7</td>
<td>n.c.</td>
<td>c7</td>
<td>n.c.</td>
</tr>
<tr>
<td>a8</td>
<td>n.c.</td>
<td>b8</td>
<td>n.c.</td>
<td>c8</td>
<td>n.c.</td>
</tr>
<tr>
<td>a9</td>
<td>n.c.</td>
<td>b9</td>
<td>n.c.</td>
<td>c9</td>
<td>n.c.</td>
</tr>
<tr>
<td>a0</td>
<td>n.c.</td>
<td>b0</td>
<td>n.c.</td>
<td>c0</td>
<td>n.c.</td>
</tr>
</tbody>
</table>
DTS200 Win32 Software V3.2

Date: 15.11.2007
# 1 Source Files of the DTS200 Controller Program

1.1 General

1.2 Global Data and Functions

1.3 Dialogs and Windows of the Desktop

1.3.1 The Class TMainForm

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMainForm::TMainForm</td>
<td>1-10</td>
</tr>
<tr>
<td>TMainForm::InitBitmaps</td>
<td>1-10</td>
</tr>
<tr>
<td>TMainForm::FormResize</td>
<td>1-10</td>
</tr>
<tr>
<td>TMainForm::FormCloseQuery</td>
<td>1-10</td>
</tr>
<tr>
<td>TMainForm::FormClose</td>
<td>1-11</td>
</tr>
<tr>
<td>TMainForm::FormShow</td>
<td>1-11</td>
</tr>
<tr>
<td>TMainForm::File1Click</td>
<td>1-12</td>
</tr>
<tr>
<td>TMainForm::SaveRecordedData1Click</td>
<td>1-12</td>
</tr>
<tr>
<td>TMainForm::LoadRecordedData1Click</td>
<td>1-12</td>
</tr>
<tr>
<td>TMainForm::SaveSystemParameters1Click</td>
<td>1-13</td>
</tr>
<tr>
<td>TMainForm::LoadSystemParameters1Click</td>
<td>1-13</td>
</tr>
<tr>
<td>TMainForm::Print1Click</td>
<td>1-13</td>
</tr>
<tr>
<td>TMainForm::PrintDesktop1Click</td>
<td>1-14</td>
</tr>
<tr>
<td>TMainForm::PrintSetup1Click</td>
<td>1-14</td>
</tr>
<tr>
<td>TMainForm::ExitProgram1Click</td>
<td>1-14</td>
</tr>
<tr>
<td>TMainForm::SelectDAC</td>
<td>1-14</td>
</tr>
<tr>
<td>TMainForm::Parameters1Click</td>
<td>1-15</td>
</tr>
<tr>
<td>TMainForm::DecouplingController1Click</td>
<td>1-15</td>
</tr>
<tr>
<td>TMainForm::PIController1Click</td>
<td>1-15</td>
</tr>
<tr>
<td>TMainForm::SimulatedSystemErrors1Click</td>
<td>1-16</td>
</tr>
<tr>
<td>TMainForm::CharacteristicLiquidLevelSensor1Click</td>
<td>1-16</td>
</tr>
<tr>
<td>TMainForm::OutflowCoefficients1Click</td>
<td>1-16</td>
</tr>
<tr>
<td>TMainForm::CharacteristicPumpFlowRate1Click</td>
<td>1-17</td>
</tr>
<tr>
<td>TMainForm::CrossSectionsofTanks1Click</td>
<td>1-17</td>
</tr>
</tbody>
</table>
Table of Contents

Three – Tank – System DTS200

TMainForm::Run1Click..................................................................................................................... 1-17
TMainForm::OpenLoopControl1Click ............................................................................................. 1-18
TMainForm::DecouplingController2Click ...................................................................................... 1-18
TMainForm::PIController2Click ................................................................................................... 1-18
TMainForm::ResetPIController1Click............................................................................................ 1-19
TMainForm::StopController1Click................................................................................................ 1-19
TMainForm::AdjustSetpoint1Click ............................................................................................... 1-19
TMainForm::Measuring1Click ....................................................................................................... 1-20
TMainForm::View1Click.............................................................................................................. 1-20
TMainForm::PlotRecordedData1Click ......................................................................................... 1-20
TMainForm::PlotFileData1Click .................................................................................................. 1-20
TMainForm::ParametersfromPLDFile1Click .............................................................................. 1-21
TMainForm::CharacteristicLiquidLevelSensors1Click............................................................... 1-21
TMainForm::CharacteristicPumpFlowRates1Click....................................................................... 1-21
TMainForm::OutflowCoefficients2Click...................................................................................... 1-22
TMainForm::CharacteristicOutflow1Click.................................................................................. 1-22
TMainForm::Contents1Click........................................................................................................... 1-22
TMainForm::SearchforHelpOn1Click ............................................................................................ 1-22
TMainForm::HowtoUseHelp1Click................................................................................................ 1-22
TMainForm::About1Click .............................................................................................................. 1-23
TMainForm::TrackBar1Change..................................................................................................... 1-23
TMainForm::TrackBar2Change..................................................................................................... 1-23
TMainForm::PBControllerPaint .................................................................................................... 1-24
TMainForm::UpdateOnTimer......................................................................................................... 1-24
TMainForm::DrawController......................................................................................................... 1-24
TMainForm::DrawArrow............................................................................................................... 1-25
TMainForm::DrawArrowVert......................................................................................................... 1-25

1.3.2 The Class TAboutBox............................................................................................................... 1-26
TAboutBox::TAboutBox .............................................................................................................. 1-26
TAboutBox::FormShow................................................................................................................ 1-26

ii  DTS200 Win32 Software V3.2
1.3.3 The Class TCalSensDlg ........................................................................................................... 1-26
TCalSensDlg::TCalSensDlg ........................................................................................................... 1-26
TCalSensDlg::FormShow ............................................................................................................. 1-27
TCalSensDlg::BtMeasLowClick .................................................................................................... 1-27
TCalSensDlg::BtMeasHighClick ................................................................................................. 1-27
TCalSensDlg::HelpBtnClick ....................................................................................................... 1-28

1.3.4 The Class TDecoupParamDlg ............................................................................................... 1-28
TDecoupParamDlg::TDecoupParamDlg ....................................................................................... 1-28
TDecoupParamDlg::FormShow .................................................................................................... 1-28
TDecoupParamDlg::OKBtnClick ................................................................................................. 1-28
TDecoupParamDlg::HelpBtnClick ............................................................................................... 1-29

1.3.5 The Class TMeasureDlg ...................................................................................................... 1-29
TMeasureDlg::TMeasureDlg ....................................................................................................... 1-29
TMeasureDlg::OKBtnClick ......................................................................................................... 1-29
TMeasureDlg::HelpBtnClick ....................................................................................................... 1-30

1.3.6 The Class TOutFlowCoefDlg .............................................................................................. 1-30
TOutFlowCoefDlg::TOutFlowCoefDlg ......................................................................................... 1-30
TOutFlowCoefDlg::FormShow ..................................................................................................... 1-30
TOutFlowCoefDlg::StartBtnClick ............................................................................................... 1-30
TOutFlowCoefDlg::CancelButtonClick ....................................................................................... 1-31
TOutFlowCoefDlg::HelpBtnClick ............................................................................................... 1-31

1.3.7 The Class TPIParmDlg ........................................................................................................ 1-31
TPIParmDlg::TPIParmDlg .......................................................................................................... 1-31
TPIParmDlg::FormShow ............................................................................................................. 1-32
TPIParmDlg::OKBtnClick .......................................................................................................... 1-32
TPIParmDlg::CBDecoupClick .................................................................................................... 1-32
TPIParmDlg::HelpBtnClick ........................................................................................................ 1-33

1.3.8 The Class TPLDInfoDlg ................................................................................................... 1-33
TPLDInfoDlg::TPLDInfoDlg ..................................................................................................... 1-33
TPLDInfoDlg::FormShow ........................................................................................................... 1-33
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPLDInfoDlg::HelpBtnClick</td>
<td>1-34</td>
</tr>
<tr>
<td>1.3.9 The Class TPlotDlg</td>
<td>1-34</td>
</tr>
<tr>
<td>TPlotDlg::TPlotDlg</td>
<td>1-34</td>
</tr>
<tr>
<td>TPlotDlg::OKBtnClick</td>
<td>1-34</td>
</tr>
<tr>
<td>TPlotDlg::HelpBtnClick</td>
<td>1-35</td>
</tr>
<tr>
<td>1.3.10 The Class TPrintPlotDlg</td>
<td>1-35</td>
</tr>
<tr>
<td>TPrintPlotDlg::TPrintPlotDlg</td>
<td>1-35</td>
</tr>
<tr>
<td>TPrintPlotDlg::FormShow</td>
<td>1-35</td>
</tr>
<tr>
<td>TPrintPlotDlg::PrinterBitBtnClick</td>
<td>1-36</td>
</tr>
<tr>
<td>TPrintPlotDlg::OKBtnClick</td>
<td>1-36</td>
</tr>
<tr>
<td>TPrintPlotDlg::HelpBtnClick</td>
<td>1-36</td>
</tr>
<tr>
<td>1.3.11 The Class TPumpCharDlg</td>
<td>1-37</td>
</tr>
<tr>
<td>TPumpCharDlg::TPumpCharDlg</td>
<td>1-37</td>
</tr>
<tr>
<td>TPumpCharDlg::FormShow</td>
<td>1-37</td>
</tr>
<tr>
<td>TPumpCharDlg::StartBtnClick</td>
<td>1-37</td>
</tr>
<tr>
<td>TPumpCharDlg::CancelBtnClick</td>
<td>1-38</td>
</tr>
<tr>
<td>TPumpCharDlg::HelpBtnClick</td>
<td>1-38</td>
</tr>
<tr>
<td>1.3.12 The Class TSetpointDlg</td>
<td>1-38</td>
</tr>
<tr>
<td>TSetpointDlg::TSetpointDlg</td>
<td>1-38</td>
</tr>
<tr>
<td>TSetpointDlg::FormShow</td>
<td>1-38</td>
</tr>
<tr>
<td>TSetpointDlg::OKBtnClick</td>
<td>1-39</td>
</tr>
<tr>
<td>TSetpointDlg::HelpBtnClick</td>
<td>1-39</td>
</tr>
<tr>
<td>LimitEditStr</td>
<td>1-39</td>
</tr>
<tr>
<td>1.3.13 The Class TSimulationDlg</td>
<td>1-40</td>
</tr>
<tr>
<td>TSimulationDlg::TSimulationDlg</td>
<td>1-40</td>
</tr>
<tr>
<td>TSimulationDlg::OKBtnClick</td>
<td>1-40</td>
</tr>
<tr>
<td>TSimulationDlg::HelpBtnClick</td>
<td>1-41</td>
</tr>
<tr>
<td>1.3.14 The Class TTankDimsDlg</td>
<td>1-41</td>
</tr>
<tr>
<td>TTankDimsDlg::TTankDimsDlg</td>
<td>1-41</td>
</tr>
<tr>
<td>TTankDimsDlg::FormShow</td>
<td>1-41</td>
</tr>
</tbody>
</table>
TTankDimsDlg::OKBtnClick ............................................................................................................ 1-41
TTankDimsDlg::HelpBtnClick .......................................................................................................... 1-42
1.4 Overview of DTSSRV.DLL Classes and Interfaces ................................................................. 1-43
1.5 References of the DTSSRV32.DLL........................................................................................... 1-46
1.5.1 The DTSSRV32.DLL Interface Functions ............................................................................. 1-48
InitDll............................................................................................................................................... 1-48
SetDriverHandle ............................................................................................................................. 1-48
DoService........................................................................................................................................ 1-49
SetParameter .................................................................................................................................. 1-49
GetData .......................................................................................................................................... 1-50
ResetPIController................................................................................................................ 1-50
SenGetLevel.................................................................................................................................... 1-50
SenSetLevel .................................................................................................................................... 1-50
SenEichLevel .................................................................................................................................. 1-50
SenGetVolt....................................................................................................................................... 1-50
OfcEichStart.................................................................................................................................... 1-50
OfcEichStop..................................................................................................................................... 1-50
OfcEichGet ..................................................................................................................................... 1-51
PfrEichStart..................................................................................................................................... 1-51
PfrEichStop..................................................................................................................................... 1-51
PfrEichGet....................................................................................................................................... 1-51
LoadCalibration ............................................................................................................................... 1-51
SaveCalibration............................................................................................................................... 1-51
MeasureStart ................................................................................................................................... 1-51
MeasureLevel.................................................................................................................................. 1-52
MeasureStatus .................................................................................................................................. 1-52
SetDemo.......................................................................................................................................... 1-52
IsDemo.......................................................................................................................................... 1-52
1.5.2 The DLL Interface PLOT ..................................................................................................... 1-53
ReadPlot......................................................................................................................................... 1-54
Table of Contents

Three – Tank – System DTS200

WritePlot............................................................................................................................................. 1-54
Plot..................................................................................................................................................... 1-55
GetPlot............................................................................................................................................... 1-56
PrintPlot............................................................................................................................................. 1-56
GetPldInfo.......................................................................................................................................... 1-57

1.5.3 The Class LLS in the dtssrv32.dll...........................................................................................1-58
LLS::LLS() ........................................................................................................................................ 1-58
LLS::calcm....................................................................................................................................... 1-59
LLS::calcV.......................................................................................................................................... 1-59
LLS::SetL1......................................................................................................................................... 1-59
LLS::SetL2......................................................................................................................................... 1-59
LLS::SetV1......................................................................................................................................... 1-60
LLS::SetV2......................................................................................................................................... 1-60
LLS::newsensor ................................................................................................................................. 1-60
LLS::GetLevel................................................................................................................................... 1-60
LLS::Load.......................................................................................................................................... 1-61
LLS::Save.......................................................................................................................................... 1-61
LLS::GetStatus................................................................................................................................... 1-61

1.5.4 The Class OFC in the dtssrv32.dll ..........................................................................................1-62
OFC::OFC() ....................................................................................................................................... 1-62
OFC::SetMaxLevel............................................................................................................................ 1-62
OFC::SetMinLevel............................................................................................................................. 1-63
OFC::SetTime.................................................................................................................................... 1-63
OFC::CalcOfc .................................................................................................................................... 1-63
OFC::Load......................................................................................................................................... 1-63
OFC::Save.......................................................................................................................................... 1-64
OFC::GetStatus................................................................................................................................... 1-64

1.5.5 The Class PFR in the dtssrv32.dll ...........................................................................................1-65
PFR::PFR() ........................................................................................................................................ 1-65
PFR::pumpi2Q .................................................................................................................................... 1-65
<table>
<thead>
<tr>
<th>Class/Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PFR::pumpQ2V</td>
<td>1-66</td>
</tr>
<tr>
<td>PFR::GetVoltage</td>
<td>1-66</td>
</tr>
<tr>
<td>PFR::SetOut</td>
<td>1-66</td>
</tr>
<tr>
<td>PFR::SetMeasureLevel1</td>
<td>1-66</td>
</tr>
<tr>
<td>PFR::SetMeasureLevel2</td>
<td>1-67</td>
</tr>
<tr>
<td>PFR::SetMeasureTime</td>
<td>1-67</td>
</tr>
<tr>
<td>PFR::CalcQzu</td>
<td>1-67</td>
</tr>
<tr>
<td>PFR::Load</td>
<td>1-68</td>
</tr>
<tr>
<td>PFR::Save</td>
<td>1-68</td>
</tr>
<tr>
<td>PFR::GetStatus</td>
<td>1-68</td>
</tr>
<tr>
<td>TUSTINPI::TUSTINPI()</td>
<td>1-69</td>
</tr>
<tr>
<td>TUSTINPI::SetKP</td>
<td>1-70</td>
</tr>
<tr>
<td>TUSTINPI::GetKP</td>
<td>1-70</td>
</tr>
<tr>
<td>TUSTINPI::SetKI</td>
<td>1-70</td>
</tr>
<tr>
<td>TUSTINPI::GetKI</td>
<td>1-70</td>
</tr>
<tr>
<td>TUSTINPI::SetTA</td>
<td>1-71</td>
</tr>
<tr>
<td>TUSTINPI::GetTA</td>
<td>1-71</td>
</tr>
<tr>
<td>TUSTINPI::ResetControl</td>
<td>1-71</td>
</tr>
<tr>
<td>TUSTINPI::Control</td>
<td>1-71</td>
</tr>
<tr>
<td>TUSTINPI::CalcZCoef</td>
<td>1-72</td>
</tr>
<tr>
<td>STOREBUF::ResetBufIndex</td>
<td>1-73</td>
</tr>
<tr>
<td>STOREBUF::STOREBUF</td>
<td>1-74</td>
</tr>
<tr>
<td>STOREBUF::~STOREBUF()</td>
<td>1-74</td>
</tr>
<tr>
<td>STOREBUF::StartMeasure</td>
<td>1-74</td>
</tr>
<tr>
<td>STOREBUF::WriteValue</td>
<td>1-74</td>
</tr>
<tr>
<td>STOREBUF::SetOutChan</td>
<td>1-75</td>
</tr>
<tr>
<td>STOREBUF::ReadValue</td>
<td>1-75</td>
</tr>
<tr>
<td>STOREBUF::GetBufLen</td>
<td>1-76</td>
</tr>
<tr>
<td>Function/Class</td>
<td>Page</td>
</tr>
<tr>
<td>----------------</td>
<td>------</td>
</tr>
<tr>
<td>STOREBUF::GetBufTa</td>
<td>1-76</td>
</tr>
<tr>
<td>STOREBUF::GetStatus</td>
<td>1-76</td>
</tr>
<tr>
<td>STOREBUF::GetBufferLevel</td>
<td>1-77</td>
</tr>
<tr>
<td>AFBUF::AFBUF()</td>
<td>1-78</td>
</tr>
<tr>
<td>AFBUF::~AFBUF()</td>
<td>1-78</td>
</tr>
<tr>
<td>AFBUF::NewFBuf</td>
<td>1-78</td>
</tr>
<tr>
<td>AFBUF::ReadFBuf</td>
<td>1-79</td>
</tr>
<tr>
<td>AFBUF::WriteFBuf</td>
<td>1-79</td>
</tr>
<tr>
<td>TWOBUFFER::TWOBUFFER()</td>
<td>1-81</td>
</tr>
<tr>
<td>TWOBUFFER::New2Buffer</td>
<td>1-81</td>
</tr>
<tr>
<td>TWOBUFFER::Write2Buffer</td>
<td>1-81</td>
</tr>
<tr>
<td>TWOBUFFER::Read2Buffer</td>
<td>1-82</td>
</tr>
<tr>
<td>SIGNAL::SIGNAL()</td>
<td>1-83</td>
</tr>
<tr>
<td>SIGNAL::InitTime</td>
<td>1-83</td>
</tr>
<tr>
<td>SIGNAL::MakeSignal</td>
<td>1-84</td>
</tr>
<tr>
<td>SIGNAL::ReadNextValue</td>
<td>1-84</td>
</tr>
<tr>
<td>SIGNAL::SetRange</td>
<td>1-85</td>
</tr>
<tr>
<td>SIGNAL::WriteBuffer</td>
<td>1-85</td>
</tr>
<tr>
<td>SIGNAL::Stuetzstellen</td>
<td>1-85</td>
</tr>
<tr>
<td>DACDRV::DACDRV</td>
<td>1-86</td>
</tr>
<tr>
<td>DACDRV::~DACDRV</td>
<td>1-88</td>
</tr>
<tr>
<td>DACDRV::SetTankDims</td>
<td>1-89</td>
</tr>
<tr>
<td>DACDRV::ReadTank1</td>
<td>1-89</td>
</tr>
<tr>
<td>DACDRV::ReadTank2</td>
<td>1-90</td>
</tr>
<tr>
<td>DACDRV::ReadTank3</td>
<td>1-90</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>----------</td>
<td>------</td>
</tr>
<tr>
<td>DACDRV::SetPumpFlow1</td>
<td>1-91</td>
</tr>
<tr>
<td>DACDRV::SetPumpFlow2</td>
<td>1-91</td>
</tr>
<tr>
<td>DACDRV::SetDemo</td>
<td>1-92</td>
</tr>
<tr>
<td>DACDRV::GetSensorErrors</td>
<td>1-93</td>
</tr>
<tr>
<td>DACDRV::GetPumpErrors</td>
<td>1-93</td>
</tr>
<tr>
<td>DACDRV::ResetSimSystem</td>
<td>1-93</td>
</tr>
<tr>
<td>DACDRV::SetNoise</td>
<td>1-93</td>
</tr>
<tr>
<td>DACDRV::SensNoise</td>
<td>1-94</td>
</tr>
<tr>
<td>DACDRV::SenGetLevel</td>
<td>1-94</td>
</tr>
<tr>
<td>DACDRV::SenSetLevel</td>
<td>1-94</td>
</tr>
<tr>
<td>DACDRV::SenEichLevel</td>
<td>1-95</td>
</tr>
<tr>
<td>DACDRV::SenGetVolt</td>
<td>1-95</td>
</tr>
<tr>
<td>DACDRV::OfcEichStart</td>
<td>1-95</td>
</tr>
<tr>
<td>DACDRV::OfcEichStop</td>
<td>1-96</td>
</tr>
<tr>
<td>DACDRV::OfcEichGet</td>
<td>1-96</td>
</tr>
<tr>
<td>DACDRV::PfrEichStart</td>
<td>1-96</td>
</tr>
<tr>
<td>DACDRV::PfrEichStop</td>
<td>1-97</td>
</tr>
<tr>
<td>DACDRV::PfrEichGet</td>
<td>1-97</td>
</tr>
<tr>
<td>DACDRV::Save</td>
<td>1-97</td>
</tr>
<tr>
<td>DACDRV::Load</td>
<td>1-98</td>
</tr>
<tr>
<td>DACDRV::eichok</td>
<td>1-98</td>
</tr>
<tr>
<td>DACDRV::CheckSystem</td>
<td>1-98</td>
</tr>
<tr>
<td>DACDRV::CheckFree</td>
<td>1-98</td>
</tr>
<tr>
<td>DACDRV::StartInterrupt</td>
<td>1-99</td>
</tr>
<tr>
<td>DACDRV::TriggerEndstufe</td>
<td>1-99</td>
</tr>
<tr>
<td>1.6.2 The Class WDAC98</td>
<td>1-100</td>
</tr>
<tr>
<td>WDAC98::ReadAnalogVolt</td>
<td>1-100</td>
</tr>
<tr>
<td>WDAC98::WriteAnalogVolt</td>
<td>1-101</td>
</tr>
<tr>
<td>WDAC98::ReadDigital</td>
<td>1-101</td>
</tr>
<tr>
<td>WDAC98::WriteDigital</td>
<td>1-101</td>
</tr>
</tbody>
</table>
2 Interface Functions of the PLOTBC32.DLL ................................. 2-1

Version......................................................................................... 2-2
CreateSimplePlotWindow................................................................. 2-3
ShowPlotWindow ........................................................................... 2-4
ClosePlotWindow ........................................................................... 2-4
UpdatePlotWindow ........................................................................ 2-4
GetValidPlotHandle ..................................................................... 2-4
AddPlotTitle ................................................................................. 2-5
AddAxisPlotWindow ...................................................................... 2-5
AddXData: ..................................................................................... 2-6
AddTimeData: ................................................................................ 2-6
AddYData: ..................................................................................... 2-7
SetCurveMode ................................................................................. 2-7
SetPlotMode .................................................................................. 2-8
PrintPlotWindow ........................................................................... 2-10
CreateEmptyPlotWindow ................................................................. 2-10
ValidPlotWindow .......................................................................... 2-10
GetNumberOfCurves .................................................................... 2-11
GetidCurvePlotWindow ................................................................. 2-11
SavePlotWindow ........................................................................... 2-12
RestorePlotWindow ...................................................................... 2-12

3 Interface Functions of the TIMER32B.DLL ................................. 3-1
## Table of Contents

- SetService ............................................................................................................................................ 3-1
- SelectDriver ......................................................................................................................................... 3-2
- StartTimer ............................................................................................................................................ 3-2
- IsTimerActive ...................................................................................................................................... 3-3
- StopTimer ............................................................................................................................................ 3-3
- GetMinMaxTime ................................................................................................................................. 3-3
- GetSimTime ......................................................................................................................................... 3-4
- GetPCTimings ..................................................................................................................................... 3-4
- SetupDriver .......................................................................................................................................... 3-5

### 4 Interface Functions of the DACSHNTB.DLL ................................................................. 4-1

- OpenDACdrv ....................................................................................................................................... 4-3
- CloseDACdrv ....................................................................................................................................... 4-3
- InitDACDevice .................................................................................................................................... 4-4
- GetDACError ....................................................................................................................................... 4-4
- GetDACStatus ...................................................................................................................................... 4-5
- ReadDigital ......................................................................................................................................... 4-6
- ReadDigitalMultiple ............................................................................................................................ 4-6
- ReadDigitalMultMask .......................................................................................................................... 4-6
- ReadAllDigital ..................................................................................................................................... 4-7
- WriteDigital ......................................................................................................................................... 4-7
- WriteDigitalSeries .............................................................................................................................. 4-8
- WriteAllDigital .................................................................................................................................... 4-8
- ReadAnalogInt ..................................................................................................................................... 4-8
- ReadAnalogVolt ................................................................................................................................. 4-9
- WriteAnalogInt .................................................................................................................................... 4-9
- WriteAnalogVolt ................................................................................................................................. 4-10
- SetCounter .......................................................................................................................................... 4-10
- SetGateCounter ................................................................................................................................. 4-10
- GetCounter ......................................................................................................................................... 4-11
SetDACTimer ........................................................................................................................................ 4-11
SetDACTimerMs ............................................................................................................................ 4-11
SetGateTimer ............................................................................................................................... 4-12
GetDACTimer.................................................................................................................................. 4-12
ResetDDM .................................................................................................................................... 4-12
ResetAllDDM ................................................................................................................................ 4-13
ReadDDM ..................................................................................................................................... 4-13
ReadDDMSel ................................................................................................................................. 4-13
SetDDMFilter ............................................................................................................................... 4-14
SetDDMDir ................................................................................................................................... 4-14
SetINT ............................................................................................................................................ 4-15
SetClock ..................................................................................................................................... 4-15
WaitDAC ..................................................................................................................................... 4-16

5  WDM Drivers for DAC98, DAC6214, DIC24, MF614 and MF624......................... 5-1

Table of device or channel macros............................................................................................... 5-1
Table of user control codes ........................................................................................................... 5-2
Table of user control codes with IObuffer layout ...................................................................... 5-3
Table of special user control codes with IObuffer layout ............................................................ 5-4
Table of valid parameters depending on the type of the adapter card ........................................ 5-5
Table of initial settings after using DRVCMD_INIT ................................................................. 5-5
1 Source Files of the DTS200 Controller Program

1.1 General
The program is a 32-bit application compatible to the operating systems Windows 98SE, ME, 2000, XP. The main parts of the program are the desktop program DTS200W32.exe and the dynamic link library (DLL) DTSSRV32.dll realizing the controller of the laboratory experiment. Both program parts are available in source ’C++’ completely. The program package is completed by

the file DEFAULT.CAL containing calibration data,

the TIMER32B.DLL to handle the cyclic controller calls,

the DACSHNTB.dll to access the PC adapter card,

the PLOTBC32.DLL for the graphic output and

the help file DTS200.HLP.

Generating a new executable program is possible only by means of the development system 'Borland C++ Builder' version 6.0 (or possibly newer versions). The controller DLL may be generated using another 32-bit-C++ compiler in case a suitable project file can be created.

After installing the source package using "setup.exe" from the installation CD the following directory structure:

```plaintext
amiraDev
  DTS200
  Source
    Demo
    DT200dsk
    DTserv32
    EXE
```

will be found in a user defined directory (default: private user directory).

Where Source contains the Borland project group file DTS200W32.BPG which should be opened by the Borland IDE to edit and create a new program.

The directory DTSDESK contains the Borland project file DTS200W32.BPR together with all the accompanying C++ source files to generate the desktop program DTS200W32.exe.

The directory DTserv32 contains the Borland project file dtssrv32.bpr with all the accompanying C++ source files to create the controller-DLL dtssrv32.dll. Finally the subdirectory EXE contains all the additional files required by the executable program (DACSHNTB.dll, DEFAULT.CAL, DTS200.HLP, PlotBC32.dll, Timer32B.dll).

Attention: After creating a new desktop or a new controller-DLL, the new results are copied by default to the subdirectory EXE.

A DEMO version of the program (simulation of the mathematical model of the three-tank-system instead of accessing the PC adapter card) may be obtained simply by setting the macro __DTS_DEMO__ in the include file DTSDEFIN.H and generating a new dtssrv32.dll. Because the resulting DLL has the same name as the DLL con-
trolling the real system, it should be copied together with all the required files to the different subdirectory DEMO afterwards.

1.2 Global Data and Functions

The file `DTSDEFIN.H` contains some definitions to clarify the readability of the source code and to adjust the program mode as well as the fixed sampling period. When `__DTS_DEMO__` is defined all program functions besides system calibration are available for a simulated three tank system. The simulation of the system behaviour runs ten times faster than the real system. The PC adapter card is not required in this program mode. To control the real system the macro `__DTS_DEMO__` must not be defined!

Used definitions:

```c
#define __DTS_DEMO__
#define TRUE  (1==1)
#define FALSE  (!TRUE)
#define SAMPLETIME 0.05
#ifdef __DTS_DEMO__
    #define SIMTIME 10.0 *SAMPLETIME
#else
    #define SIMTIME SAMPLETIME
#endif
#define ERROR   -1
#define OK   0
#define AVALVE    0.5 // Sn      [cm^2]
#define ATANK     154.0 // Aq      [cm^2]
#define TWOG  1962.0  // 2.0 * g [cm/s^2]
```

The file `DTS20DAT.H` contains global data structures which are used in different instances of the software. These structures are saved in the data files used to store measurements.

```c
// Data structures:
struct PROJECT{
    char number[10]; // P335
    char name[10]; // DTS200
    char Titel[20]; // DTS200 - Monitor
    char Version[10]; // last version
    char Date[10]; // last date
    char Dummy[10]; // Reserve
};
struct CTRLSTATUS {
    SHORT OpenLoop; // Flag open loop control is active
    SHORT PX; // Flag P controller is active
    SHORT PI; // Flag PI controller is active
    SHORT DeCoupled; // Flag decoupling is active
    double Decoup; // Coefficient of the decoupling controller
    double P; // Coefficient P of the P or PI controller
    double K; // Coefficient Ki of the PI controller
    long timeofmeasure; // Date and time of the measurement acquisition
};
```
Three – Tank – System DTS200

Source Files of the DTS200 Controller Program


};
struct DATASTRUCT { // Structure to reconstruct the measured data
    SHORT nchannel; // Length of the stored measurement vectors (number of channels)
    SHORT nvalues;  // Number of the measurement vectors (samples)
    float deltatime; // Time between two samples
};

The file dtserv32.h contains the data structure definitions for those data which are interchanged between the desktop and the controller DLL.

#define NONE   0
#define OPENLOOP 1
#define PICONTROLLER 2
#define DECOUPLED 3

typedef struct{
    WORD controller;
    WORD decoupled;
    WORD sp1shape;
    double p;
    double i;
    double decoup;
    double sp1offset;
    double sp1amplitude;
    double sp1period;
    double sp2offset;
    double sp2amplitude;
    double sp2period;
    double At1;
    double At2;
    double At3;
} ServiceParameters;

typedef struct{
    double t1setpoint;
    double t2setpoint;
    double t1level;
    double t2level;
    double t3level;
    double p1flow;
    double p2flow;
    double t1sensvolt;
    double t2sensvolt;
    double t2sensvolt;
    double p1contrvolt;
    double p2contrvolt;
} ServiceData;
The file **DTSDLls.h** contains the function declarations of all functions exported by the dynamic link libraries DACSHNTB.dll, PlotBC32.dll and Timer32B.dll as well as the global data declarations:

```c
extern ServiceParameters param;
extern ServiceData data;
```

The files **DTSMain.cpp** contains the global data:

```c
WORD cardNo = 1;  // number of PC adapter card (1...4)
ServiceParameter param;    // main instance of ServiceParameter, see "dtserv32.h"
ServiceData data;      // main instance of ServiceData, see "dtserv32.h"
Graphics::TBitmap *pDTSPict; // pointer to bitmap for controller structure
const TERR_OK = 0;  // constant for no error in Timer32B.dll function
UINT resultU;       // general result
TimerState;         // timer status
BOOL resultB;       // general result
double SetpointMax = 60.0; // maximum set point for tank level/flow rate
int PLDFileError;   // status for reading PLD file
BOOL SaveCalibData,  // status flag for saved calibration data to file
    NewCalibration;   // status flag for new calibration data
```

### The Format of the Documentation Data File *.PLD*

Measured data stored in a data file are reloadable and may be output in a graphic representation. In addition the system settings (CTRLSTATUS) which were active during the start of the data acquisition are stored in this file. They are displayable in a separate window.

The data file contains data in binary format stored in the following order:

- The structure PROJEKT PRJ
- The structure CTRLSTATUS
- The structure DATASTRUCT
- The data array with float values (4 bytes per value)

The size of the data array is defined in the structure DATASTRUCT. With the DTS200 the number of the stored channels is always 8 (the length of the measurement vector is 8, i.e. equal to 32 bytes). The vector contains the following signals:

- the set point for tank 1 (in cm),
- the set point for tank 2 (in cm),
- the measured liquid level from tank 1 (in cm),
- the measured liquid level from tank 2 (in cm),
- the measured liquid level from tank 3 (in cm),
- the control signal for pump 1 (in ml/s),
- the control signal for pump 2 (in ml/s),
- the calculated nominal outflow q20 (in ml/s)

The number of the stored measurement acquisitions (vectors) depends on the adjusted values for the sampling period and the measuring time. The maximum number of sampled vectors is 1024. The time distance between two successive acquisitions is an integral multiple of the sampling period used by the controller.
1.3 Dialogs and Windows of the Desktop

The file DTS200W32.CPP is the main application program for the desktop. It only contains the WINAPI function `WinMain` which initializes the application, creates all the required forms for the dialogs and runs the application.

The main window with its menu bar as well as all of the following dialogs and message boxes are realized by the following files.

The files DTSMain.cpp (DTSMain.h, DTSMain.dfm) contain the class `TMainForm` with the functions:

```cpp
__fastcall TMainForm( TComponent* Owner ) : TForm(Owner)

void __fastcall InitBitmaps( void )

void __fastcall FormResize( TObject* Sender )

void __fastcall FormCloseQuery( TObject* Sender, bool &CanClose)

void __fastcall FormClose( TObject* Sender, TCloseAction &Action)

void __fastcall FormShow(TObject* Sender)

void __fastcall File1Click(TObject* Sender)

void __fastcall SaveRecordedData1Click(TObject* Sender)

void __fastcall LoadRecordedData1Click(TObject* Sender)

void __fastcall SaveSystemParameters1Click(TObject* Sender)

void __fastcall LoadSystemParameters1Click(TObject* Sender)

void __fastcall Print1Click(TObject* Sender)

void __fastcall PrintDesktop1Click(TObject* Sender)

void __fastcall PrintSetup1Click(TObject* Sender)

void __fastcall ExitProgram1Click(TObject* Sender)

void __fastcall SelectDAC(TObject* Sender)

void __fastcall Parameters1Click(TObject* Sender)

void __fastcall DecouplingController1Click(TObject* Sender)

void __fastcall PIController1Click(TObject* Sender)

void __fastcall SimulatedSystemErrors1Click(TObject* Sender)

void __fastcall CharacteristicLiquidLevelSensor1Click(TObject* Sender)

void __fastcall OutflowCoefficients1Click(TObject* Sender)

void __fastcall CharacteristicPumpFlowRate1Click(TObject* Sender)

void __fastcall CrossSectionsofTanks1Click(TObject* Sender)

void __fastcall Run1Click(TObject* Sender)

void __fastcall OpenLoopControl1Click(TObject* Sender)

void __fastcall DecouplingController2Click(TObject* Sender)

void __fastcall PIController2Click(TObject* Sender)
```

DTS200 Win32 Software V3.2 1-5
void __fastcall ResetPIController1Click(TObject* Sender)
void __fastcall StopController1Click(TObject* Sender)
void __fastcall AdjustSetpoint1Click(TObject* Sender)
void __fastcall Measuring1Click(TObject* Sender)
void __fastcall View1Click(TObject* Sender)
void __fastcall PlotRecordedData1Click(TObject* Sender)
void __fastcall PlotFileData1Click(TObject* Sender)
void __fastcall ParametersfromPLDFile1Click(TObject* Sender)
void __fastcall CharacteristicLiquidLevelSensors1Click(TObject* Sender)
void __fastcall CharacteristicPumpFlowRates1Click(TObject* Sender)
void __fastcall OutflowCoefficients2Click(TObject* Sender)
void __fastcall CharacteristicOutflow1Click(TObject* Sender)
void __fastcall Contents1Click(TObject* Sender)
void __fastcall SearchforHelpOn1Click(TObject* Sender)
void __fastcall HowtoUseHelp1Click(TObject* Sender)
void __fastcall About1Click(TObject* Sender)
void __fastcall TrackBar1Change(TObject* Sender)
void __fastcall TrackBar2Change(TObject* Sender)
void __fastcall PBControllerPaint(TObject* Sender)
void __fastcall UpdateOnTimer(TObject* Sender)
int __fastcall DrawController( int xStart, int yStart, int TankNr )
int __fastcall DrawArrow( int xStart, int yStart, int Length )
int __fastcall DrawArrowVert( int xStart, int yStart, int Length )

The files about.cpp (about.h, about.dfm) contain the class TAboutBox with the functions:

__fastcall TAboutBox( TComponent* Owner ) : TForm(Owner)
void __fastcall FormShow(TObject* Sender)

The files CalSens.cpp (CalSens.h, CalSens.dfm) contain the class TCalSensDlg with the functions:

__fastcall TCalSensDlg( TComponent* Owner ) : TForm(Owner)
void __fastcall FormShow(TObject* Sender)
void __fastcall BtMeasLowClick(TObject* Sender)
void __fastcall BtMeasHighClick(TObject* Sender)
void __fastcall HelpBtnClick(TObject* Sender)

The files Decoup.cpp (Decoup.h, Decoup.dfm) contain the class TDecoupParamDlg with the functions:

__fastcall TDecoupParamDlg( TComponent* Owner) : TForm(Owner)
void __fastcall FormShow(TObject* Sender)
void __fastcall OkBtnClick(TObject* Sender)
void __fastcall HelpBtnClick(TObject* Sender)

The files Measure.cpp (Measure.h, Measure.dfm) contain the class TMeasureDlg with the functions:

__fastcall TMeasureDlg( TComponent* Owner) : TForm(Owner)
void __fastcall OkBtnClick(TObject* Sender)
void __fastcall HelpBtnClick(TObject* Sender)

The files Outflow.cpp (Outflow.h, Outflow.dfm) contain the class TOutFlowCoefDlg with the functions:

__fastcall TOutFlowCoefDlg( TComponent* Owner) : TForm(Owner)
void __fastcall FormShow(TObject* Sender)
void __fastcall StartBtnClick(TObject* Sender)
void __fastcall CancelBtnClick(TObject* Sender)
void __fastcall HelpBtnClick(TObject* Sender)

The files PiParm.cpp (PiParm.h, PiParm.dfm) contain the class TPIParmDlg with the functions:

__fastcall TPIParmDlg( TComponent* Owner) : TForm(Owner)
void __fastcall FormShow(TObject* Sender)
void __fastcall OkBtnClick(TObject* Sender)
void __fastcall HelpBtnClick(TObject* Sender)
void __fastcall CBDecoupClick(TObject* Sender)

The files PLDInfo.cpp (PLDInfo.h, PLDInfo.dfm) contain the class TPLDInfoDlg with the functions:

__fastcall TPLDInfoDlg( TComponent* Owner) : TForm(Owner)
void __fastcall FormShow(TObject* Sender)
void __fastcall HelpBtnClick(TObject* Sender)
The files PlotSel.cpp (PlotSel.h, PlotSel.dfm) contain the class \texttt{TPlotDlg} with the functions:

\begin{verbatim}
\_fastcall TPlotDlg( TComponent* Owner ) : TForm(Owner)
\end{verbatim}

\begin{verbatim}
void \_fastcall OkBtnClick(TObject* Sender)
\end{verbatim}

\begin{verbatim}
void \_fastcall BitBtn1Click(TObject* Sender)
\end{verbatim}

The files Printplt.cpp (Printplt.h, Printplt.dfm) contain the class \texttt{TPrintPlotDlg} with the functions:

\begin{verbatim}
\_fastcall TPParmDlg( TComponent* Owner ) : TForm(Owner)
\end{verbatim}

\begin{verbatim}
void \_fastcall FormShow(TObject* Sender)
\end{verbatim}

\begin{verbatim}
void \_fastcall PrinterBtnClick(TObject* Sender)
\end{verbatim}

\begin{verbatim}
void \_fastcall OkBtnClick(TObject* Sender)
\end{verbatim}

\begin{verbatim}
void \_fastcall HelpBtnClick(TObject* Sender)
\end{verbatim}

The files Pumpchar.cpp (Pumpchar.h, Pumpchar.dfm) contain the class \texttt{TPumpCharDlg} with the functions:

\begin{verbatim}
\_fastcall TPumpCharDlg( TComponent* Owner ) : TForm(Owner)
\end{verbatim}

\begin{verbatim}
void \_fastcall FormShow(TObject* Sender)
\end{verbatim}

\begin{verbatim}
void \_fastcall StartBtnClick(TObject* Sender)
\end{verbatim}

\begin{verbatim}
void \_fastcall CancelBtnClick(TObject* Sender)
\end{verbatim}

\begin{verbatim}
void \_fastcall HelpBtnClick(TObject* Sender)
\end{verbatim}

The files Setpoint.cpp (Setpoint.h, Setpoint.dfm) contain the class \texttt{TSetpointDlg} with the functions:

\begin{verbatim}
\_fastcall TSetpointDlg( TComponent* Owner ) : TForm(Owner)
\end{verbatim}

\begin{verbatim}
void \_fastcall FormShow(TObject* Sender)
\end{verbatim}

\begin{verbatim}
void \_fastcall OKBtnClick(TObject* Sender)
\end{verbatim}

\begin{verbatim}
void \_fastcall HelpBtnClick(TObject* Sender)
\end{verbatim}

\begin{verbatim}
AnsiString \_fastcall LimitEditStr(AnsiString EdStr, double *pVal, double min, double max)
\end{verbatim}

The files Simulat.cpp (Simulat.h, Simulat.dfm) contain the class \texttt{TSimulationDlg} with the functions:

\begin{verbatim}
\_fastcall TSimulationDlg( TComponent* Owner ) : TForm(Owner)
\end{verbatim}

\begin{verbatim}
void \_fastcall OkBtnClick(TObject* Sender)
\end{verbatim}

\begin{verbatim}
void \_fastcall HelpBtnClick(TObject* Sender)
\end{verbatim}
The files Tankdim.cpp (Tankdim.h, Tankdim.dfm) contain the class `TTankDimsDlg` with the functions:

```cpp
__fastcall TTankDimsDlg(TComponent* Owner) : TForm(Owner)

void __fastcall FormShow(TObject* Sender)

void __fastcall OKBtnClick(TObject* Sender)

void __fastcall HelpBtnClick(TObject* Sender)
```
1.3.1 The Class TMainForm

TMainForm::TMainForm

TMainForm( TComponent* Owner ) : TForm(Owner)

Parameters:

Owner is a reference to the calling component.

Description

The constructor TMainForm creates an instance of an object of type TMainForm, which is the main desktop of the application. The desktop is referenced by the global pointer MainForm. The constructor initializes a global bitmap by calling InitBitmaps and shows the current controller structure (=NONE).

TMainForm::InitBitmaps

InitBitmaps( void )

Description

The function InitBitmaps creates a new global bitmap (*pDTSPict) with a size relatively to the size of the controller structure paintbox of TMainForm. The bitmap is filled with zeros and gets the brush colour as well as the character font from this paintbox.

TMainForm::FormResize

FormResize( TObject* Sender)

Parameters:

Sender is a reference to the calling object.

Description

The function FormResize resizes the controller structure paintbox as well as the two trackbars relatively to the new size of TMainForm:: The global bitmap (*pDTSPict) is deleted in case it exists and recreated by calling InitBitmaps.

TMainForm::FormCloseQuery

FormResize( TObject* Sender, bool &CanClose )

Parameters:

Sender is a reference to the calling object.

&CanClose is a reference to a variable of TForm.

Description

The function FormCloseQuery controls the termination of the program through FormClose by setting the variable CanClose. At first the message box "Exit program?" is shown. Prompting this message by 'Ok' will set CanClose to TRUE, otherwise to FALSE. For a positive answer the controller is stopped through StopController1Click and a time delay of one second is started. When a sensor calibration was carried-out previously without saving the system pa-
rameters to a file, another message box "Exit without saving calibration data?" is shown. If this message is not prompted with 'Ok', saving the system parameters by calling SaveSystem-Parameter1Click is carried-out automatically.

### TMainForm::FormClose

`FormClose( TObject* Sender, TCloseAction &Action )`

**Parameters:**

- `Sender` is a reference to the calling object.
- `&Action` is a reference to a variable of TForm.

**Description**

The function **FormClose** stops the timer for calling the DoService function of the dtssrv32.dll and deletes the global bitmap (*pDTSPict) in case it exists.

### TMainForm::FormShow

`FormShow(TObject* Sender)`

**Parameters:**

- `Sender` is a reference to the calling object.

**Description**

The function **FormShow** is called when the object of type TMainForm is displayed on the screen. This is equivalent to starting the application (the program DTS200W32). At first the system registry is searched for the key "amiraW32.INI" with the section "DTS200" to read the driver type and the card number of the PC adapter card used to control the tank system. If this section is missing it is initialized with the type "MF624.DRV" and a card number of 1.

Then the program tries to load the dtssrv32.dll by calling the function SetService of the Timer32B.dll. In case this function fails, the message "Service DLL 'DTSSRV32.DLL' is missing !" appears on the screen.

At next the function IsDemo from the dtssrv32.dll is called to determine if it is a DEMO version. Only in this case the menu item "IO-Interface" is disabled, the menu item "Simulated System Errors" is enabled and the string "](Demo-Version)" is appended to the title of the monitor. Then the function SelectDriver of the Timer32B.dll is called to inform this DLL about the driver type and card number of the PC adapter card. If this function detects an invalid driver type the message "SelectDriver failed !" appears on the screen. Otherwise the check marks below the menu item "IO-Interface" are set accordingly. By calling GetParameter and GetData the global parameter structures param and data are obtained from the dtssrv32.dll.

Appropriate error messages appear on the screen if one of these functions fails due to wrong structure sizes. Then the parameter structure param is set to default values (no controller, proportional factor=10, integral factor=1, decoupling factor=0.04, set point tank 1 constant 30 cm, set point tank 2 constant 15 cm) and transferred to the dtssrv32.dll by calling SetParameter. The state for reading the documentation file (extension *.PLD) as well as for saving calibration data is reset. At next the timer handling the sampling period (50ms) of the controller is
started by calling **StartTimer** of the Timer32B.dll. If this function fails the error message
"StartTimer failed !" is sent to the screen. At last the desktop is updated with the current controller settings.

**TMainForm::File1Click**

File1Click(TObject* Sender)

**Parameters:**

*Sender* is a reference to the calling object.

**Description**

The function **File1Click** is an event handler for clicking once on the calling object (*Sender*),
which was activated by the main menu item "File". When measurements are available in the
memory the menu item "Save Recorded Data" is enabled, else it is disabled. When no controller
is active, the menu items "Save System Parameter" and "Load System Parameter" are enabled, else they are disabled.

**TMainForm::SaveRecordedData1Click**

SaveRecordedData1Click(TObject* Sender)

**Parameters:**

*Sender* is a reference to the calling object.

**Description**

The function **SaveRecordedData1Click** is an event handler for clicking once on the calling object (*Sender*), which was activated by the menu item "File/Save Recorded Data". A Windows system dialog appears offering the storage (**WritePlot**) of the current measurements contained in the memory to a documentation file with an adjustable name (extension *.PLD). The error message "Error writing *.PLD file" appears on the screen in case writing to file failed.

**TMainForm::LoadRecordedData1Click**

LoadRecordedData1Click(TObject* Sender)

**Parameters:**

*Sender* is a reference to the calling object.

**Description**

The function **LoadRecordedData1Click** is an event handler for clicking once on the calling object (*Sender*), which was activated by the menu item "File/Load Recorded Data". A Windows system dialog appears, which allows for selecting a documentation file with an adjustable name (extension *.PLD) to load measurements (**ReadPlot**) into the memory. The error message "Error reading *.PLD file" appears on the screen in case writing to file failed. The global status **PLDDeviceError** is set accordingly.
TMaintForm::SaveSystemParameters1Click

SaveSystemParameters1Click(TObject* Sender)

Parameters:
Sender is a reference to the calling object.

Description
The function SaveSystemParameters1Click is an event handler for clicking once on the calling object (Sender), which was activated by the menu item "File/Save System Parameters". A Windows system dialog appears offering the storage (SaveCalibration) of the current calibration data (sensor and pump characteristics, outflow coefficients) to a file with an adjustable name (extension *.CAL). The global status SaveCalibData is set in case of successful writing to file. Otherwise the message box "Error writing system calibration data file" appears on the screen.

TMaintForm::LoadSystemParameters1Click

LoadSystemParameters1Click(TObject* Sender)

Parameters:
Sender is a reference to the calling object.

Description
The function LoadSystemParameters1Click is an event handler for clicking once on the calling object (Sender), which was activated by the menu item "File/Load System Parameters". A Windows system dialog appears, which allows for selecting a file with an adjustable name (extension *.CAL) to load calibration data (sensor and pump characteristics, outflow coefficients) into the memory (LoadCalibration). The message box "Error reading system calibration data file" appears on the screen in case the file read operation failed.

TMaintForm::Print1Click

Print1Click(TObject* Sender)

Parameters:
Sender is a reference to the calling object.

Description
The function Print1Click generates a modal dialog (PrintPlotDlg) to select previously created plot windows, which are to be printed to an output device (i.e. printer).
TMainForm::PrintDesktop1Click

PrintDesktop1Click(TObject* Sender)

Parameters:

Sender is a reference to the calling object.

Description

The function PrintDesktop1Click opens the standard printer setup dialog of Windows to select and adjust the output device for a screen copy of the current desktop (form).

TMainForm::PrintSetup1Click

PrintSetup1Click(TObject* Sender)

Parameters:

Sender is a reference to the calling object.

Description

The function PrintSetup1Click calls the standard printer setup dialog of Windows to select and adjust the output device.

TMainForm::ExitProgram1Click

ExitProgram1Click(TObject* Sender)

Parameters:

Sender is a reference to the calling object.

Description

The function ExitProgram1Click terminates the running application, that means the program DTS200W32 is terminated.

TMainForm::SelectDAC

SelectDAC(TObject* Sender)

Parameters:

Sender is a reference to the calling object.

Description

The function SelectDAC is an event handler for clicking once on the calling object (Sender), which was activated by the menu item "IO-Interface/<item>", where <item> is a placeholder for DAC6214, DAC98, MF614 or MF624.

If IsTimerActive returns true the timer for the controller is stopped (StopTimer) and the global status TimerState is set accordingly. Any check mark for <item> is reset. Then the function calls SelectDriver of the Timer32B.dll to inform this DLL about the driver type corresponding to <item> and the current card number of the PC adapter card. With SetupDriver the program tries to open up to 4 driver entries of type <item>. If there is no such driver available the message box "It's not possible to configure this driver." appears on the screen. In case
there is only 1 driver available the user is informed by "Only one card/driver is present and se-
lected !!". If more than 1 entry was detected another dialog appears to select a suitable card
number. With a positive result of SetupDriver the check mark for <item> is set, the driver
type and card number are written to the registry key "amiraW32.INI" with the section
"DTS200" and the timer for the controller is restarted with StartTimer. If this operation failed
the user is informed by "StartTimer failed !".

TMainForm::Parameters1Click

Parameters1Click(TObject* Sender)

Parameters:  
Sender is a reference to the calling object.

Description  
The function Parameters1Click is an event handler for clicking once on the calling object
(Sender), which was activated by the menu item "Parameter". The menu items to record the
calibration data (sensor and pump characteristics, outflow coefficients) are enabled when no
controller is active. These menu items are disabled in case of an active controller or a DEMO
version.

TMainForm::DecouplingController1Click

DecouplingController1Click(TObject* Sender)

Parameters:  
Sender is a reference to the calling object.

Description  
The function DecouplingController1Click is an event handler for clicking once on the call-
ing object (Sender), which was activated by the menu item "Parameters/Decoupling Control-
ero". A modal dialog (DecoupParamDlg) appears to adjust the decoupling parameter.

TMainForm::PIController1Click

PIController1Click(TObject* Sender)

Parameters:  
Sender is a reference to the calling object.

Description  
The function PIController1Click is an event handler for clicking once on the calling object
(Sender), which was activated by the menu item "Parameters/PI-Controller". A modal dialog
(PIParamDlg) appears to adjust the proportional and integral factor of the PI controller as
well as the decoupling parameter only if the decoupling structure was selected in addition.
TMainForm::SimulatedSystemErrors1Click

SimulatedSystemErrors1Click(TObject* Sender)

Parameters:

Sender is a reference to the calling object.

Description

The function SimulatedSystemErrors1Click is an event handler for clicking once on the calling object (Sender), which was activated by the menu item "Parameters/Simulated System Errors". A modal dialog (SimulationDlg) appears to adjust simulated system errors (sensor and control signal errors as well as leaks and clogs) only for the DEMO version.

TMainForm::CharacteristicLiquidLevelSensor1Click

CharacteristicLiquidLevelSensor1Click(Sender: TObject)

Parameters:

Sender is a reference to the calling object.

Description

The function CharacteristicLiquidLevelSensor1Click is an event handler for clicking once on the calling object (Sender), which was activated by the menu item "Parameters/Characteristic Liquid Level Sensor". After switching off any currently active controller a modal dialog (CalcSensDlg) appears supporting the user with recording the sensor characteristics. The global status SaveCalibData is reset and the global status NewCalibration is set.

TMainForm::OutflowCoefficients1Click

OutflowCoefficients1Click(TObject* Sender)

Parameters:

Sender is a reference to the calling object.

Description

The function OutflowCoefficients1Click is an event handler for clicking once on the calling object (Sender), which was activated by the menu item "Parameters/Outflow Coefficients". After switching off any currently active controller a modal dialog (OutFlowCoefDlg) appears supporting the user with recording the sensor characteristics. The global status SaveCalibData is reset and the global status NewCalibration is set.
TMainForm::CharacteristicPumpFlowRate1Click

CharacteristicPumpFlowRate1Click(Sender: TObject)

Parameters:

Sender is a reference to the calling object.

Description

The function CharacteristicPumpFlowRate1Click is an event handler for clicking once on the calling object (Sender), which was activated by the menu item "Parameters/Characteristic Pump Flow Rate". After switching off any currently active controller a modal dialog (Pump-CharDlg) appears supporting the user with recording the pump characteristics. The global status SaveCalibData is reset and the global status NewCalibration is set.

TMainForm::CrossSectionsofTanks1Click

CrossSectionsofTanks1Click(Sender: TObject)

Parameters:

Sender is a reference to the calling object.

Description

The function CrossSectionsofTanks1Click is an event handler for clicking once on the calling object (Sender), which was activated by the menu item "Parameters/Cross Section of Tanks". The modal dialog (TankDimsDlg) appears allowing the user to view or to enter the effective cross sections of the three tanks.

TMainForm::Run1Click

Run1Click(TObject* Sender)

Parameters:

Sender is a reference to the calling object.

Description

The function Run1Click is an event handler for clicking once on the calling object (Sender), which was activated by the main menu item "Run". The menu items to select a controller are enabled and the menu items to adjust the set points and to start the measurement acquisition are disabled when no controller is active. In the opposite case the menu items to select a controller are disabled and the menu items to adjust the set points and to start the measurement acquisition are enabled.
TMainForm::OpenLoopControl1Click

OpenLoopControl1Click(TObject* Sender)

Parameters:

Sender is a reference to the calling object.

Description

The function OpenLoopControl1Click is an event handler for clicking once on the calling object (Sender), which was activated by the menu item "Run/Open Loop Control". The flags indicating a selected controller (for the menu items as well as in the parameter structure param) are set to OPENLOOP and the parameters of the two set point generators are reset. The parameter structure changed in such a way is transferred to the dtssrv32.dll (SetParameter). The global SetpointMax is set for maximum flow rate and the two track bars for adjusting the control signals are set accordingly. All adjustments are carried-out only in case open loop control was inactive previously.

TMainForm::DecouplingController2Click

DecouplingController2Click(TObject* Sender)

Parameters:

Sender is a reference to the calling object.

Description

The function DecouplingController2Click is an event handler for clicking once on the calling object (Sender), which was activated by the menu item "Run/Decoupling Controller". The flags indicating a selected controller (for the menu items as well as in the parameter structure param) are set to DECOUPLED (proportional decoupling controller) and the parameters of the two set point generators are set to standard values (set point tank 1 constant 30 cm, set point tank 2 constant 15 cm). The parameter structure changed in such a way is transferred to the dtssrv32.dll (SetParameter). The global SetpointMax is set for maximum liquid level and the two track bars for adjusting the set points are set accordingly. All adjustments are carried-out only in case the proportional decoupling controller was inactive previously.

TMainForm::PIController2Click

PIController2Click(TObject* Sender)

Parameters:

Sender is a reference to the calling object.

Description

The function PIController2Click is an event handler for clicking once on the calling object (Sender), which was activated by the menu item "Run/PI-Controller". The flags indicating a selected controller (for the menu items as well as in the parameter structure param) are set to PICONTROLLER (PI controller) and the parameters of the two set point generators are set to standard values (set point tank 1 constant 30 cm, set point tank 2 constant 15 cm). The parameter structure changed in such a way is transferred to the dtssrv32.dll (SetParameter). The
global SetpointMax is set for maximum liquid level and the two track bars for adjusting the set points are set accordingly. All adjustments are carried-out only in case the PI controller was inactive previously.

---

**TMainForm::ResetPIController1Click**

```cpp
ResetPIController1Click(TObject* Sender)
```

**Parameters:**

*Sender* is a reference to the calling object.

**Description**

The function `ResetPIController1Click` is an event handler for clicking once on the calling object (*Sender*), which was activated by the menu item "Run/Reset PI-Controller". The function `ResetPIController` is called to reset the PI controller states.

---

**TMainForm::StopController1Click**

```cpp
StopController1Click(TObject* Sender)
```

**Parameters:**

*Sender* is a reference to the calling object.

**Description**

The function `StopController1Click` is an event handler for clicking once on the calling object (*Sender*), which was activated by the menu item "Run/Stop All". The flags indicating a selected controller (for the menu items as well as in the parameter structure `param`) are reset. The parameter structure changed in such a way is transferred to the dtssrv32.dll (`SetParameter`). The track bars for set points or control signals disappear from the screen.

---

**TMainForm::AdjustSetpoint1Click**

```cpp
AdjustSetpoint1Click(TObject* Sender)
```

**Parameters:**

*Sender* is a reference to the calling object.

**Description**

The function `AdjustSetpoint1Click` is an event handler for clicking once on the calling object (*Sender*), which was activated by the menu item "Run/Adjust Setpoint". A modal dialog (`GeneratorDlg`) appears for adjusting the tank level set points or the pump control signals respectively. For a constant signal shape the two track bars for set points or control signals are enabled and set accordingly. The track bars are disabled for other signal shapes.
TMainForm::Measuring1Click

Measuring1Click(TObject* Sender)

Parameters:

Sender is a reference to the calling object.

Description

The function Measuring1Click is an event handler for clicking once on the calling object (Sender), which was activated by the menu item "Run/Measuring". A modal dialog (Measur-eDlg) appears for adjusting the conditions of a measurement acquisition.

TMainForm::View1Click

View1Click(TObject* Sender)

Parameters:

Sender is a reference to the calling object.

Description

The function View1Click is an event handler for clicking once on the calling object (Sender), which was activated by the menu item "View". The menu item "Plot Measured Data" is en-abled when measurements are available in the memory, else it is disabled. When data from a documentation file (extension *.PLD) have been loaded, the menu items "Plot File Data" and "Parameter from *.PLD File" are enabled, else they are disabled.

TMainForm::PlotRecordedData1Click

PlotRecordedData1Click(TObject* Sender)

Parameters:

Sender is a reference to the calling object.

Description

The function PlotRecordedData1Click is an event handler for clicking once on the calling object (Sender), which was activated by the menu item "View/Plot Measured Data". A modal dialog (PlotDlg) appears to select those components of the last measurement acquisition which are to be represented in a plot window.

TMainForm::PlotFileData1Click

PlotFileData1Click(TObject* Sender)

Parameters:

Sender is a reference to the calling object.

Description

The function PlotFileData1Click is an event handler for clicking once on the calling object (Sender), which was activated by the menu item "View/Plot File Data". A modal dialog
(PlotDlg) appears to select those components of a loaded documentation file which are to be represented in a plot window.

TMainForm::ParametersfromPLDFile1Click

ParametersfromPLDFile1Click(TObject* Sender)

Parameters:

Sender is a reference to the calling object.

Description

The function ParametersfromPLDFile1Click is an event handler for clicking once on the calling object (Sender), which was activated by the menu item "View/Parameters from *.PLD File". A modal dialog (PLDInfoDlg) displaying the essential controller settings from a loaded documentation file appears directly.

TMainForm::CharacteristicLiquidLevelSensors1Click

CharacteristicLiquidLevelSensors1Click(Sender: TObject)

Parameters:

Sender is a reference to the calling object.

Description

The function CharacteristicLiquidLevelSensors1Click is an event handler for clicking once on the calling object (Sender), which was activated by the menu item "View/Characteristic Liquid Level Sensors". A plot window representing the three sensor characteristics will appear directly.

TMainForm::CharacteristicPumpFlowRates1Click

CharacteristicPumpFlowRates1Click(Sender: TObject)

Parameters:

Sender is a reference to the calling object.

Description

The function CharacteristicPumpFlowRates1Click is an event handler for clicking once on the calling object (Sender), which was activated by the menu item "View/Characteristic Pump Flow Rates". A plot window representing the two pump characteristics will appear directly.
TMainForm::OutflowCoefficients2Click

```
OutflowCoefficients2Click(TObject* Sender)
```

**Parameters:**

*Sender* is a reference to the calling object.

**Description**

The function `OutflowCoefficients2Click` is an event handler for clicking once on the calling object (*Sender*), which was activated by the menu item "View/Outflow Coefficients". A message box displaying the current outflow coefficients will appear directly.

TMainForm::CharacteristicOutflow1Click

This function is reserved for future use

```
CharacteristicOutflow1Click(Sender: TObject)
```

**Parameters:**

*Sender* is a reference to the calling object.

**Description**

The function `CharacteristicOutflow1Click` is an event handler for clicking once on the calling object (*Sender*), which was activated by the menu item "View/Characteristic Outflow". A plot window representing the outflow characteristic of the nominal outflow will appear directly.

TMainForm::Contents1Click

```
Contents1Click(TObject* Sender)
```

**Parameters:**

*Sender* is a reference to the calling object.

**Description**

The function `Contents1Click` is activated by the menu item "Help/Contents" to display the content of the help file.

TMainForm::SearchforHelpOn1Click

```
SearchforHelpOn1Click(TObject* Sender)
```

**Parameters:**

*Sender* is a reference to the calling object.

**Description**

The function `SearchforHelpOn1Click` is activated by the menu item "Help/Search for Help On" to start the Windows dialog to search for selectable keywords.
HowtoUseHelp1Click(TObject* Sender)

Parameters:

Sender is a reference to the calling object.

Description

The function HowtoUseHelp1Click is activated by the menu item "Help/How to Use help" to start the Windows dialog displaying hints how to use the help function.

TMainForm::About1Click

About1Click(TObject* Sender)

Parameters:

Sender is a reference to the calling object.

Description

The function About1Click is activated by the menu item "Help/About" displaying information about the actual program in a message box.

TMainForm::TrackBar1Change

TrackBar1Change(TObject* Sender)

Parameters:

Sender is a reference to the calling object.

Description

The function TrackBar1Change is activated when the user changes the position of track bar 1. In case the shape of the set point/control signal for tank 1/pump 1 is constant the global structure param is updated accordingly and sent to the dtssrv32.dll using SetParameter.

TMainForm::TrackBar2Change

TrackBar2Change(TObject* Sender)

Parameters:

Sender is a reference to the calling object.

Description

The function TrackBar2Change is activated when the user changes the position of track bar 2. In case the shape of the set point/control signal for tank 2/pump 2 is constant the global structure param is updated accordingly and sent to the dtssrv32.dll using SetParameter.
TMainForm::PBControllerPaint

PBControllerPaint(TObject* Sender)

Parameters:
Sender is a reference to the calling object.

Description
The function PBControllerPaint is an event handler for the system message 'paint' received by the paintbox displaying the global bitmap (*pDTSPict) of the controller structure. It calls the function UpdateOnTimer.

TMainForm::UpdateOnTimer

UpdateOnTimer(TObject* Sender)

Parameters:
Sender is a reference to the calling object.

Description
The function UpdateOnTimer is an event handler for the desktop timer, which produces timer events with an interval of 1000 msec. The function reads the global structures param and data from the dtssrv32.dll using GetParameter and GetData. The label for the active controller is set accordingly. The same happens to the track bar positions in case the corresponding signal shape is not constant. The current status of the data recording is read by MeasureStatus and displayed as a label and a progress bar position.

Then the global bitmap (*pDTSPict) holding the controller structure is cleared, a new controller structure is drawn (DrawController) to this bitmap and the bitmap is copied to the paintbox. Finally the paintbox is shown on the form.

TMainForm::DrawController

int DrawController( int xStart, int yStart, int TankNr )

Parameters:
xStart is the left position of the structure in pixel units.
yStart is the upper position of the structure in pixel units.
TankNr is the tank number (1, 2, 3) in the structure.

Description
The function DrawController draws the complete controller structure with signal values together with a tank symbol onto the global bitmap (*pDTSPict). Only for tank with number 3 the controller structure is omitted. The controller structure depends on the settings open loop or closed loop with/without decoupling and/or PI-control. The sizes of the boxes, arrows and Tank symbols are relative to the size of the paintbox. But the structure will contain all of its elements in readable form only when the height and width of the paintbox is big enough.

Return
Is always 1.
TMainForm::DrawArrow

int DrawArrow(TObject* Sender)

Parameters:

xStart is the horizontal (x) starting point in pixel units.
yStart is the vertical (y) starting point in pixel units.

Length is the length of the arrow (>0 from left to right, <0 from right to left).

Description

The function DrawArrow draws a horizontal arrow onto the global bitmap (*pDTSPict) beginning from the start position (xStart, yStart). Its length including the arrowhead is always the absolute value of Length and its orientation is from left to right or for a negative value of Length from right to left. The arrowhead is a triangle of 4*4*8 pixel filled with black colour.

Return

The final x position after the arrow is drawn.

TMainForm::DrawArrowVert

int DrawArrowVert(TObject* Sender)

Parameters:

xStart is the horizontal (x) starting point in pixel units.
yStart is the vertical (y) starting point in pixel units.

Length is the length of the arrow (>0 from bottom to top, <0 top to bottom).

Description

The function DrawArrowVert draws a vertical arrow onto the global bitmap (*pDTSPict) beginning from the start position (xStart, yStart). Its length excluding the arrowhead is always the absolute value of Length and its orientation is from bottom to top or for a negative value of Length from top to bottom. The arrowhead is a triangle of 4*4*8 pixel filled with black colour.

Return

The final x position after the arrow is drawn.
1.3.2 The Class TAboutBox

TAboutBox::TAboutBox

TAboutBox( TComponent* Owner ) : TForm(Owner)

Parameters:

Owner is a reference to the calling component.

Description

TAboutBox is the standard constructor of this class without any further action. The dialog is referenced by the global pointer AboutBox.

TAboutBox::FormShow

FormShow(TObject* Sender)

Parameters:

Sender is a reference to the calling object.

Description

The function FormShow is called, when the object of type TAboutBox is displayed on the screen (TMainForm::About1Click). Short information about the program (name, version, copyright, required PC adapter card) are presented in a window. One text line indicates if the program is a DEMO version or a version for the real plant by calling the function IsDemo.

1.3.3 The Class TCalSensDlg

TCalSensDlg::TCalSensDlg

TCalSensDlg( TComponent* Owner ) : TForm(Owner)

Parameters:

Owner is a reference to the calling component.

Description

TCalSensDlg is the standard constructor of this class without any further action. The dialog is referenced by the global pointer CalSensDlg.
TCalSensDlg::FormShow

FormShow(TObject* Sender)

Parameters:
Sender is a reference to the calling object.

Description
The function FormShow is called, when the object of type TCalSensDlg is displayed on the screen (TMainForm::CharacteristicLiquidLevelSensor1Click). The input fields for the lower and upper calibration point in [cm] are enabled and filled with the current values for these points (DACDRV::SenGetLevel). The "measure_1" button is enabled while the "measure_2" button is disabled to guarantee the sequence of the sensor calibration.

TCalSensDlg::BtMeasLowClick

BtMeasLowClick(TObject* Sender)

Parameters:
Sender is a reference to the calling object.

Description
The function BtMeasLowClick is an event handler for clicking once on the "measure_1" button from the dialog to calibrate the liquid level sensors. This button as well as the input field for the lower calibration point in [cm] are disabled. The content of this input field is converted to a number and taken as the lower calibration point (DACDRV::SenSetLevel) only when it is inside the range from 0 to 60 [cm]. The current sensor value in [Volt] for the liquid level of each tank (the user has to adjust this level by controlling the pumps manually) is recorded and assigned to the calibration point (DACDRV::SenEichLevel). Finally the "measure_2" button is enabled.

TCalSensDlg::BtMeasHighClick

BtMeasHighClick(TObject* Sender)

Parameters:
Sender is a reference to the calling object.

Description
The function BtMeasHighClick is an event handler for clicking once on the "measure_2" button from the dialog to calibrate the liquid level sensors. This button as well as the input field for the upper calibration point in [cm] are disabled. The content of this input field is converted to a number and taken as the upper calibration point (DACDRV::SenSetLevel) only when it is inside the range from 0 to 60 [cm]. The current sensor value in [Volt] for the liquid level of each tank (the user has to adjust this level by controlling the pumps manually) is recorded and assigned to the calibration point (DACDRV::SenEichLevel). Finally the modal dialog is terminated.
TCalSensDlg::HelpBtnClick

\[\text{HelpBtnClick}(\text{TObject* } \text{Sender})\]

**Parameters:**
- **Sender**: a reference to the calling object.

**Description**
The function \text{HelpBtnClick} is an event handler for clicking once on the "Help" button from the dialog to calibrate the liquid level sensors. The accompanying section of the help file will be displayed in a window on the screen.

1.3.4 The Class TDecoupParamDlg

TDecoupParamDlg::TDecoupParamDlg

\[\text{TDecoupParamDlg}(\text{TComponent* } \text{Owner}) : \text{TForm(Owner)}\]

**Parameters:**
- **Owner**: a reference to the calling component.

**Description**
TDecoupParamDlg is the standard constructor of this class without any further action. The dialog is referenced by the global pointer \text{DecoupParamDlg}.

TDecoupParamDlg::FormShow

\[\text{FormShow}(\text{TObject* } \text{Sender})\]

**Parameters:**
- **Sender**: a reference to the calling object.

**Description**
The function \text{FormShow} is called, when the object of type TDecoupParamDlg is displayed on the screen (TMainForm::DecouplingController1Click). The decoupling parameter from the global parameter structure \text{param} is converted to an ASCII-string, which is inserted in the input field of the dialog.

TDecoupParamDlg::OKBtnClick

\[\text{OKBtnClick}(\text{TObject* } \text{Sender})\]

**Parameters:**
- **Sender**: a reference to the calling object.

**Description**
The function \text{OKBtnClick} is an event handler for clicking once on the "Ok" button from the dialog to adjust the decoupling parameter. The content of the input field of the dialog is converted to a number when this number does not exceed the range from 0 to 100. The global parameter structure \text{param} is transferred to the controller of the dtssrv32.dll.
### TDecoupParamDlg::HelpBtnClick

**HelpBtnClick(TObject* Sender)**

**Parameters:**

*Sender* is a reference to the calling object.

**Description**

The function **HelpBtnClick** is an event handler for clicking once on the "Help" button from the dialog to adjust the decoupling parameter. The accompanying section of the help file will be displayed in a window on the screen.

### 1.3.5 The Class TMeasureDlg

#### TMeasureDlg::TMeasureDlg

**TMeasureDlg(TComponent* Owner) : TForm(Owner)**

**Parameters:**

*Owner* is a reference to the calling component.

**Description**

**TMeasureDlg** is the standard constructor of this class without any further action. The dialog is referenced by the global pointer *MeasureDlg*. It is shown by **TMainForm::Measuring1Click**.

#### TMeasureDlg::OKBtnClick

**OKBtnClick(TObject* Sender)**

**Parameters:**

*Sender* is a reference to the calling object.

**Description**

The function **OKBtnClick** is an event handler for clicking once on the "Ok" button from the dialog to adjust the conditions for the measurement acquisition. The contents of three input fields are converted to numbers for the total measuring time (*time* = 0 to 1000 sec), for the time before reaching the trigger condition (*prestore* = 0 to total measuring time) and for the trigger level (*trigger* = 0 to 100) only when none of the numbers exceeds the valid range. Two further groups of radio buttons are used to determine the trigger channel (*channel*) as well as the trigger condition (*slope*). The trigger condition is either not existing or defined as a slope, meaning that the measured value of the trigger channel has to exceed the trigger level either in positive or in negative direction. The measuring is started directly after terminating the dialog. Measured values are the set points, liquid levels as well as the pump control signals.
TMeasureDlg::HelpBtnClick

HelpBtnClick(TObject* Sender)

Parameters:

Sender is a reference to the calling object.

Description

The function HelpBtnClick is an event handler for clicking once on the "Help" button from the dialog to adjust the conditions for the measurement acquisition. The accompanying section of the help file will be displayed in a window on the screen.

1.3.6 The Class TOutFlowCoefDlg

TOutFlowCoefDlg::TOutFlowCoefDlg

TOutFlowCoefDlg( TComponent* Owner ) : TForm(Owner)

Parameters:

Owner is a reference to the calling component.

Description

TOutFlowCoefDlg is the standard constructor of this class without any further action. The dialog is referenced by the global pointer OutFlowCoefDlg.

TOutFlowCoefDlg::FormShow

FormShow(TObject* Sender)

Parameters:

Sender is a reference to the calling object.

Description

The function FormShow is called, when the object of type TOutFlowCoefDlg is displayed on the screen (TMainForm::OutflowCoefficients1Click). The "Start" button is enabled and the display for the progress of the measurement time is reset.

TOutFlowCoefDlg::StartBtnClick

StartBtnClick(TObject* Sender)

Parameters:

Sender is a reference to the calling object.

Description

The function StartBtnClick is an event handler for clicking once on the "Start" button from the dialog to determine (calibrate) the outflow coefficients. After a delay time of 60 sec the actual measurements are taken in between 15 sec (OfcEichStart, OfcEichStop). If the "Cancel" button is not pressed during the overall time of 75 sec, the progress of which is displayed in a
bar in percentage below the "Start" button, the determined outflow coefficients are displayed in a message box. Prompting this message will also terminate the dialog.

**TOutFlowCoefDlg::CancelBtnClick**

`CancelBtnClick(TObject* Sender)`

**Parameters:**

`Sender` is a reference to the calling object.

**Description**

The function `CancelBtnClick` is an event handler for clicking once on the "Cancel" button. The dialog is either aborted directly or indirectly by resetting the flag of a current measurement.

**TOutFlowCoefDlg::HelpBtnClick**

`HelpBtnClick(TObject* Sender)`

**Parameters:**

`Sender` is a reference to the calling object.

**Description**

The function `HelpBtnClick` is an event handler for clicking once on the "Help" button from the dialog to determine the outflow coefficients. The accompanying section of the help file will be displayed in a window on the screen.

1.3.7 The Class TPIParmDlg

**TPIParmDlg::TPIParmDlg**

`TPIParmDlg( TComponent* Owner ) : TForm(Owner)`

**Parameters:**

`Owner` is a reference to the calling component.

**Description**

`TPIParmDlg` is the standard constructor of this class without any further action. The dialog is referenced by the global pointer `PIParmDlg`.

DTS200 Win32 Software V3.2
TPIParamDlg::FormShow

FormShow(TObject* Sender)

Parameters:

Sender is a reference to the calling object.

Description

The function FormShow is called, when the object of type TPIParamDlg is displayed on the
screen (TMainForm::PIController1Click). The proportional and integral factor from the PI
controller as well as the decoupling parameter from the global parameter structure param are
converted to ASCII-strings which are inserted in the three input fields of the dialog. When the
control structure is selected with decoupling the corresponding check mark is emphasized.

TPIParamDlg::OKBtnClick

OKBtnClick(TObject* Sender)

Parameters:

Sender is a reference to the calling object.

Description

The function OKBtnClick is an event handler for clicking once on the "Ok" button from the
dialog to adjust the proportional and integral factor from the PI controller as well as the de-
coupling parameter. The contents of the input fields of the dialog are converted to numbers
when each of these numbers does not exceed the range from 0 to 100. The control structure is
set with or without decoupling according to the check mark. The global parameter structure
param is transferred to the controller of the dtssrv32.dll.

TPIParamDlg::CBDecoupClick

CBDecoupClick(TObject* Sender)

Parameters:

Sender is a reference to the calling object.

Description

The function CBDecoupClick is an event handler for clicking once on the check mark for the
decoupling structure. The input field to enter the decoupling parameter is enabled or disabled
accordingly.
TPIParamDlg::HelpBtnClick

HelpBtnClick(TObject* Sender)

Parameters:
Sender is a reference to the calling object.

Description
The function HelpBtnClick is an event handler for clicking once on the "Help" button from the dialog to adjust the parameters of the PI controller. The accompanying section of the help file will be displayed in a window on the screen.

1.3.8 The Class TPLDInfoDlg

TPLDInfoDlg::TPLDInfoDlg

TPLDInfoDlg(TComponent* Owner) : TForm(Owner)

Parameters:
Owner is a reference to the calling component.

Description
TPLDInfoDlg is the standard constructor of this class without any further action. The dialog is referenced by the global pointer TPLDInfoDlg.

TPLDInfoDlg::FormShow

FormShow(TObject* Sender)

Parameters:
Sender is a reference to the calling object.

Description
The function FormShow is called, when the object of type TPLDInfoDlg is displayed on the screen (TMainForm::ParametersfromPLDFile1Click). The essential controller settings from a loaded documentation file are displayed in a dialog. The settings consist of the controller structure (PI controller with/without decoupling, decoupling controller, open loop control), the accompanying parameters, the measuring time as well as the sampling period of the measuring.
TPLDInfoDlg::HelpBtnClick

HelpBtnClick(TObject* Sender)

Parameters:

Sender is a reference to the calling object.

Description

The function HelpBtnClick is an event handler for clicking once on the "Help" button from the dialog to display the essential controller settings from a loaded documentation file. The accompanying section of the help file will be displayed in a window on the screen.

1.3.9 The Class TPlotDlg

Public data:

int command indicates the source of data (=1 from memory, =2 from file)

TPlotDlg::TPlotDlg

TPlotDlg(TComponent* Owner) : TForm(Owner)

Parameters:

Owner is a reference to the calling component.

Description

TPlotDlg is the standard constructor of this class without any further action. The dialog is referenced by the global pointer PlotDlg. It is shown by TMainForm::PlotRecordedData1Click with command=1 or by TMainForm::PlotFileData1Click with command=2.

TPlotDlg::OKBtnClick

OKBtnClick(TObject* Sender)

Parameters:

Sender is a reference to the calling object.

Description

The function OKBtnClick is an event handler for clicking once on the "Ok" button from the dialog to select the channels of recorded data either from memory or from file which are to be represented in a plot window. The selectable channels are the liquid levels, the liquid levels together with the set points, the pump flow rates, the pump control signals as well as the liquid levels together with the pump flow rates. The function PlotMeas is called to show the plot window.
TPlotDlg::HelpBtnClick

`HelpBtnClick(TObject* Sender)`

Parameters:

`Sender` is a reference to the calling object.

Description

The function `HelpBtnClick` is an event handler for clicking once on the "Help" button from the dialog to select the channels of a measuring which are to be represented in a plot window. The accompanying section of the help file will be displayed in a window on the screen.

1.3.10 The Class TPrintPlotDlg

Global data:

`char PlotWindowTitle[256]` is the character string for the title of an open plot window

TPrintPlotDlg::TPrintPlotDlg

`TPrintPlotDlg( TComponent* Owner) : TForm(Owner)`

Parameters:

`Owner` is a reference to the calling component.

Description

`TPrintPlotDlg` is the standard constructor of this class without any further action. The dialog is referenced by the global pointer `PrintPlotDlg`.

TPrintPlotDlg::FormShow

`FormShow(TObject* Sender)`

Parameters:

`Sender` is a reference to the calling object.

Description

The function `FormShow` is called, when the object of type TPrintPlotDlg is displayed on the screen (TMainForm::Print1Click). At first all titles of the previously created plot windows, which are open, are inserted in a listbox. The titles are obtained by calling the function `GetPlot`.

DTS200 Win32 Software V3.2 1-35
TPrintPlotDlg::PrinterBtnClick

**PrinterBtnClick(TObject* Sender)**

**Parameters:**

*Sender* is a reference to the calling object.

**Description**

The function **PrinterBtnClick** is an event handler for clicking once on the "Printer" button from the dialog to select previously created plot windows. A modal Windows system dialog appears that permits the user to select which printer to print to, how many copies to print and further print options.

TPrintPlotDlg::OKBtnClick

**OKBtnClick(TObject* Sender)**

**Parameters:**

*Sender* is a reference to the calling object.

**Description**

The function **OKBtnClick** is an event handler for clicking once on the "Ok" button from the dialog to select previously created plot windows. All of the plot windows selected from the list box are printed directly to the current output device (by means of the function **PrintPlotMeas**). When multiple plot windows are selected an offset of 150 mm (counted from the upper margin of a DIN A4 page) is added before every second print output and a form feed follows this output.

TPrintPlotDlg::HelpBtnClick

**HelpBtnClick(TObject* Sender)**

**Parameters:**

*Sender* is a reference to the calling object.

**Description**

The function **HelpBtnClick** is an event handler for clicking once on the "Help" button from the dialog to select previously created plot windows. The accompanying section of the help file will be displayed in a window on the screen.
1.3.11 The Class TPumpCharDlg

Public data:

BOOL measuring indicates the status of measuring the tank level
BOOL busy indicates the status of measuring the tank level

TPumpCharDlg::TPumpCharDlg

TPPlotDlg( TComponent* Owner) : TForm(Owner)

Parameters:

Owner is a reference to the calling component.

Description

TPumpCharDlg is the standard constructor of this class without any further action. The dialog is referenced by the global pointer PumpCharDlg.

TPumpCharDlg::FormShow

FormShow(TObject* Sender)

Parameters:

Sender is a reference to the calling object.

Description

The function FormShow is called, when the object of type TPumpCharDlg is displayed on the screen (TMainForm::CharacteristicPumpFlowRate1Click). The "Start" as well as the "Cancel" button are enabled, the bar indicating the progress of the measurement is reset and a table containing 3 columns and 9 rows is initialized.

TPumpCharDlg::StartBtnClick

StartBtnClick(TObject* Sender)

Parameters:

Sender is a reference to the calling object.

Description

The function StartBtnClick is an event handler for clicking once on the "Start" from the dialog to determine (calibrate) the pump characteristics (each containing 9 base points for the pump control signal and the accompanying pump flow rate). At first the two flags measuring and busy are set. The pump flow rates are determined in a loop with 8 passes. In each pass the pump control signals are incremented by 12.5% and supplied to the pumps during a time period of 10 sec (PfrEichStart, PfrEichStop). After another settling time of 2.5 sec the tank levels are measured. At the end of each pass the next row of the table is filled with the values for the control signal and the determined pump flow rate in [ml/s] (PfrEichGet). The progress of the loop is indicated in a bar below the "Start" button in percentage. If the "Cancel" button
is not pressed during the loop an "Ok" button will appear to terminate the dialog. A terminated loop resets the busy flag.

### TPumpCharDlg::CancelBtnClick

**CancelBtnClick** (TObject* *Sender*)

**Parameters:**

*Sender* is a reference to the calling object.

**Description**

The function `CancelBtnClick` is an event handler for clicking once on the "Cancel" button. The dialog is either aborted directly or indirectly by resetting the flag measuring of a current measurement.

### TPumpCharDlg::HelpBtnClick

**HelpBtnClick** (TObject* *Sender*)

**Parameters:**

*Sender* is a reference to the calling object.

**Description**

The function `HelpBtnClick` is an event handler for clicking once on the "Help" button from the dialog to determine the pump characteristics. The accompanying section of the help file will be displayed in a window on the screen.

### 1.3.12 The Class TSetpointDlg

**TSetpointDlg::TSetpointDlg**

**TSetpointDlg** (TComponent* *Owner*) : TForm(*Owner*)

**Parameters:**

*Owner* is a reference to the calling component.

**Description**

`TSetpointDlg` is the standard constructor of this class without any further action. The dialog is referenced by the global pointer `SetpointDlg`.

### TSetpointDlg::FormShow

**FormShow** (TObject* *Sender*)

**Parameters:**

*Sender* is a reference to the calling object.

**Description**
The function **FormShow** is called, when the object of type TSetpointDlg is displayed on the screen (TMainForm::AdjustSetpoint1Click). The input fields as well as the radio buttons to adjust the two generators are preset according to the parameters of the global parameter structure **param**. The two generators provide the liquid level set points for tank1 and tank 2 in the closed control loop or the pump control signals in the open control loop.

---

**TSetpointDlg::OKBtnClick**

OKBtnClick(TObject* *Sender*)

**Parameters:**

*Sender* is a reference to the calling object.

**Description**

The function **OKBtnClick** is an event handler for clicking once on the "Ok" button from the dialog to adjust the two set points, respectively the two pump control signal generators. For each generator the signal shape is selectable by radio buttons (constant, rectangle, triangle, ramp, sine) and an offset, an amplitude as well as a time period are adjustable by input fields. The period is meaningless in case of a constant signal shape. The real signal is always built by the sum of offset and amplitude. The valid value ranges are 0 - 60 cm for the set points, respectively 0 - 100 ml/s for the pump control signals and 0 - 1000 sec for the period. Only when none of the corresponding number exceeds the valid value range, the numbers are stored in the parameter structure **param** which is then transferred to the controller in the dtssrv32.dll. Finally the dialog is terminated and both generators operate synchronously.

---

**TSetpointDlg::HelpBtnClick**

HelpBtnClick(TObject* *Sender*)

**Parameters:**

*Sender* is a reference to the calling object.

**Description**

The function **HelpBtnClick** is an event handler for clicking once on the "Help" button from the dialog to adjust the two set points, respectively the two pump control signal generators. The accompanying section of the help file will be displayed in a window on the screen.

---

**LimitEditStr**

AnsiString LimitEditStr(AnsiString EdStr, double *pVal, double min, double max )

**Parameters:**

*EdStr* is a reference to an Ansi string representing a floating point number.

*pVal* is a pointer to a floating point number.

*min* is the lower limit for a floating point number.

*max* is the upper limit for a floating point number.
Description

The function **LimitEditStr** takes the floating point number from the AnsiString **EdStr** and stores its double value to *pVal*. If this value is greater than *max*, the value is limited to this maximum value and the user is informed by the messagebox containing "Your input <EdStr> is set to its maximum value ". If the value is smaller than *min*, the value is limited to this minimum value and the user is informed by the messagebox containing "Your input <EdStr> is set to its minimum value ".

Finally the value is converted to an AnsiString using the format f7.3.

Return

AnsiString representation of the floating point value *pVal*.

### 1.3.13 The Class TSimulationDlg

**TSimulationDlg::TSimulationDlg**

```cpp
TSimulationDlg( TComponent* Owner ) : TForm(Owner)
```

Parameters:

*Owner* is a reference to the calling component.

Description

**TSimulationDlg** is the standard constructor of this class without any further action. The dialog is referenced by the global pointer **TSimulationDlg**. It is shown by **TMainForm::SimulatedSystemErrors1Click**.

**TSimulationDlg::OKBtnClick**

```cpp
OKBtnClick(TObject* Sender)
```

Parameters:

*Sender* is a reference to the calling object.

Description

The function **OKBtnClick** is an event handler for clicking once on the "Ok" button from the dialog to adjust the simulated system errors (sensor and pump errors, leaks and clogs) in the DEMO version. The contents of the input fields of the dialog are converted to numbers when each of these numbers does not exceed the range from 0 to 100 (%). The numbers are stored in the corresponding global parameters by calling **SetDemo**.
TSimulationDlg::HelpBtnClick

    HelpBtnClick(TObject* Sender)

Parameters:  
    Sender is a reference to the calling object.

Description  
    The function HelpBtnClick is an event handler for clicking once on the "Help" button from the dialog to adjust the simulated system errors in the DEMO version. The accompanying section of the help file will be displayed in a window on the screen.

1.3.14 The Class TTankDimsDlg

TTankDimsDlg::TTankDimsDlg

    TTankDimsDlg( TComponent* Owner ) : TForm(Owner)

Parameters:  
    Owner is a reference to the calling component.

Description  
    TTankDimsDlg is the standard constructor of this class without any further action. The dialog is referenced by the global pointer TankDimsDlg.

TTankDimsDlg::FormShow

    FormShow(TObject* Sender)

Parameters:  
    Sender is a reference to the calling object.

Description  
    The function FormShow is called, when the object of type TTankDimsDlg is displayed on the screen (TMainForm::CrossSectionsofTanks1Click). The parameter structure param is read and its corresponding values of the tanks cross sections are entered as a string to the edit fields of the dialog.

TTankDimsDlg::OKBtnClick

    OKBtnClick(TObject* Sender)

Parameters:  
    Sender is a reference to the calling object.

Description  
    The function OKBtnClick is an event handler for clicking once on the "Ok" button from the dialog to adjust the tank cross sections. Only when the three numbers are limited to the range
10 to 1000, their values are copied to the parameter structure `param`, which is then sent to the `dtssrv32.dll`.

### TTankDimsDlg::HelpBtnClick

**HelpBtnClick**( TObject* `Sender`)  

**Parameters:**  

`Sender` is a reference to the calling object.  

**Description**  

The function **HelpBtnClick** is an event handler for clicking once on the "Help" button from the dialog to adjust the two set points, respectively the two pump control signal generators. The accompanying section of the help file will be displayed in a window on the screen.
1.4 **Overview of DTSSRV.DLL Classes and Interfaces**

The file DTSERV32.CPP contains the DLL interface functions:

(WIN16) BOOL CALLBACK **LibMain** ( HINSTANCE hInstance, WORD wDatenSeg, WORD cbHeapSize, LPSTR CmdLine )

(WIN16) int CALLBACK **WEP** ( int nParameter )

(WIN16) BOOL CALLBACK **LockMemory** ( BOOL bStart, HDRVR hDrv )

(WIN16) BOOL CALLBACK **SetDriverHandle** ( HDRVR hDrv )

BOOL **InitDll** ( int hInstDLL, DWORD fdwReason )

BOOL CALLBACK **SetDriverHandle** ( HANDLE hDrv )

BOOL CALLBACK **DoService** ( DWORD counter )

BOOL CALLBACK **SetParameter** ( WORD wSize, LPSTR lpData )

BOOL CALLBACK **GetData** ( WORD wSize, LPSTR lpData )

void CALLBACK **ResetPIController** ( void )

double CALLBACK **SenGetLevel** ( int mode )

void CALLBACK **SenSetLevel** ( int mode, double val )

void CALLBACK **SenEichLevel** ( int mode )

double CALLBACK **SenGetVolt** ( int s, double h )

void CALLBACK **OfcEichStart** ( void )

void CALLBACK **OfcEichStop** ( void )

double CALLBACK **OfcEichGet** ( int i )

void CALLBACK **PfrEichStart** ( int i )

void CALLBACK **PfrEichStop** ( int i, double time )

double CALLBACK **PfrEichGet** ( int p, int i )

int CALLBACK **LoadCalibration** ( char *fname )

int CALLBACK **SaveCalibration** ( char *fname )

int CALLBACK **MeasureStart** ( double time, double trigger, double prestore, int tchannel, int slope )

double CALLBACK **MeasureLevel** ( void )

int CALLBACK **MeasureStatus** ( void )

void CALLBACK **SetDemo** ( double se1, double se2, double se3, double sp1, double sp2, double sl1, double sl2, double sl3, double sv13, double sv32, double sv20 )

int CALLBACK **IsDemo** ( void )
The files DTS20SYS.H, DTS20SYS.CPP contain the classes **LLS**, **OFC**, **PFR**, **TUSTINPI** with the functions:

**class LLS**

```cpp
LLS()
  double calcem ( double volt )
  void SetL1( double in )
  void SetL2( double in )
  void SetV1( double in )
  void SetV2( double in )
  void newsensor( void )
  double GetLevel( int which )
  int Load( void )
  int Save( void )
  int GetStatus( void )
```

**class OFC**

```cpp
OFC()
  void SetMaxLevel( double h1, double h2, double h3 )
  void SetMinLevel( double h1, double h2, double h3 )
  void SetTime( double t )
  void CalcOfc( void )
  int Load( void )
  int Save( void )
  int GetStatus( void )
```

**class PFR**

```cpp
PFR(void)
  double pumpi2Q( int i )
  double pumpQ2V( double q )
  double GetVoltage( void )
  void SetOut( int in )
  void SetMeasureLevel1( double l1 )
  void SetMeasureLevel2( double l2 )
  void SetMeasureTime( double t )
  double CalcQzu( int index )
  int Load( void )
  int Save( void )
```
int GetStatus(void)

class TUSTINPI

    TUSTINPI( void )
    void SetKP( double k )
    double GetKP( void )
    void SetKI( double k )
    double GetKI( void )
    void SetTA( double ta )
    double GetTA( void )
    void ResetControl( void )
    double Control( double xd )
    void CalcZCoef( void )

The files ARINGBUF.H, ARINGBUF.CPP contain the class STOREBUF with the functions:

    void ResetBufIndex( void )
    STOREBUF(int, float *)
    ~STOREBUF()
    void StartMeasure( int, float, float, float, int ,float )
    void WriteValue( void )
    void SetOutChan(int)
    float ReadValue( void )
    int GetBufLen( void )
    float GetBufTa( void )
    int GetStatus( void )

The files DRSIGNAL.H, DRSIGNAL.CPP contain the classes AFBUF, TWOBUFFER, SIGNAL with the functions:

class AFBUF

    AFBUF()
    ~AFBUF()
    int NewFBuf( int )
    float ReadFBuf( void )
    int WriteFBuf( float )

class TWOBUFFER
void New2Buffer( int, int, int )
int Write2Buffer( float )
float Read2Buffer( void )

class SIGNAL

SIGNAL()
float InitTime( float )
int MakeSignal( int, float, float, float, int )
float ReadNextValue( void )
void SetRange( float, float )
void WriteBuffer( float )
int Stuetzstellen( float, int )

The file PLOT.CPP contains the functions:

class PLOT

int CALLBACK ReadPlot( char *lpfName )
int CALLBACK WritePlot( char *lpfName )
int CALLBACK Plot( int command, int channel )
int CALLBACK GetPlot( int start, char *lpzName )
int CALLBACK PrintPlot( int idx, HDC dcPrint, int iyOffset )
int CALLBACK GetPldInfo( int &controller, int &decoupled, double &p, double &i, double &decoup, char **s, int &n, int &c, double &d )

1.5 References of the DTSSRV32.DLL

Global Data:

typedef struct {
  WORD controller;  // controller structure (open loop, P or PI controller)
  WORD decoupled;  // flag for decoupling controller
  WORD sp1shape;  // shape of set point signal 1 (constant, rectangle, sine, etc.)
  WORD sp2shape;  // shape of set point signal 2
  double p;       // controller parameter proportional part
  double i;       // controller parameter integral part
  double decoup;  // decoupling parameter
  double sp1offset;  // offset of set point signal 1
}
```c
struct ServiceParameters {
    double sp1amplitude; // amplitude of set point signal 1
    double sp1periode;  // period of set point signal 1
    double sp1offset;      // offset of set point signal 1
    double sp2amplitude;      // amplitude of set point signal 2
    double sp2periode;        // period of set point signal 2
    double At1;        // effective cross section of tank 1
    double At2;         // effective cross section of tank 2
    double At3;         // effective cross section of tank 3
};

type struct {
    double t1setpoint;  // liquid level set point tank 1 or pump flow rate 1
    double t2setpoint;  // liquid level set point tank 2 or pump flow rate 2
    double t1level;   // measured liquid level tank 1
    double t2level;   // measured liquid level tank 2
    double t3level;   // measured liquid level tank 3
    double p1flow;   // control signal pump 1
    double p2flow;   // control signal pump 2
    double t1sensvolt;  // measured liquid level sensor voltage tank 1
    double t2sensvolt;  // measured liquid level sensor voltage tank 2
    double t3sensvolt;  // measured liquid level sensor voltage tank 3
    double p1contrvolt;  // control signal voltage for pump 1
    double p2contrvolt;  // control signal voltage for pump 2
};
```
1.5.1 The DTSSRV32.DLL Interface Functions

The functions LinMain, WEP, LockMemory and SetDriverHandle denoted by (WIN16) in the upper list are not explained here, because they are used only for 16-Bit applications.

InitDll

    BOOL InitDll( int hInstDLL, DWORD fdwReason )

Parameters

hInstDLL is a handle to the instance of the DLL.

fdwReason indicates the reason for the call.

Description

The function InitDll is the entry and exit function of the DLL. On entry new objects for the adapter card driver, the scope buffer, the signal generators and the controllers are created and assigned to the corresponding global parameters. These objects are deleted on exit.

Return

Is equal to TRUE in every case.

SetDriverHandle

    BOOL CALLBACK SetDriverHandle( HANDLE hDrv )

Parameters

hDrv is a handle for the IO-adapter card driver.

Description

The function SetDriverHandle sets the internal handle for the IO-adapter card driver equal to the actual parameter.

Attention: This function may only be called by the Timer32B.DLL!

Return

Always equal to 0.
DoService

BOOL CALLBACK DoService( DWORD counter )

Parameters

counter is a counter for the number of calls.

Description

The function DoService is the service routine for the controller interrupt. It is called at every sampling period when the interrupt is active. The following operations are carried-out in sequence:

- trigger the output stage (rectangle signal),
- read the liquid levels of 3 tanks,
- store the measured values to the measurement vector,
- determine the set points by means of the generators,
- store the set points to the measurement vector,
- calculate the pump control signals (closed/open loop control, none),
- calculate cross flow rates and nominal outflow,
- correct control signals in case of decoupling,
- store the pump control signals to the measurement vector,
- write the pump control signals,
- store the measurement-vector.

Attention: This function is to be called only by the TIMER.DLL!

SetParameter

BOOL CALLBACK SetParameter( WORD wSize, LPSTR lpData )

Parameters

wSize is the size (in bytes) of the data structure pointed to by lpData.

lpData is a pointer to a data structure of type ServiceParameter.

Description

The function SetParameter copies the data structure pointed to by lpData to the global structure par (type ServiceParameters) only when the size of the source structure is less or equal to the size of the destination structure. In this case the controller parameters of the two PI controllers, the set point generators as well as the cross sections of the tanks are set accordingly.

Return

Is equal to TRUE when the size of the source structure is less or equal to the size of the destination structure, else return is equal to FALSE.
GetData

BOOL CALLBACK GetData( WORD wSize, LPSTR lpData )

Parameters

wSize is the size (in bytes) of the data structure pointed to by lpData.

lpData is a pointer to a data structure of type ServiceData.

Description

The function GetData at first copies the content of the measurement-vector scopebuf to the global structure dat (type ServiceData). Then dat is copied to the data structure pointed to by lpData only when the size of the source structure is less or equal to the size of the destination structure.

Return

Is equal to TRUE when the size of the source structure is less or equal to the size of the destination structure, else return is equal to FALSE.

ResetPIController

void CALLBACK ResetPIController( void )

Description

The function ResetPIController resets the integral parts of the two PI controllers.

The group of functions

SenGetLevel

double CALLBACK SenGetLevel( int mode )

SenSetLevel

void CALLBACK SenSetLevel( int mode, double val )

SenEichLevel

void CALLBACK SenEichLevel( int mode )

SenGetVolt

double CALLBACK SenGetVolt( int s, double h )

OfcEichStart

void CALLBACK OfcEichStart( void )

OfcEichStop

void CALLBACK OfcEichStop( void )
OfcEichGet

double CALLBACK OfcEichGet( int i )

PfrEichStart

void CALLBACK PfrEichStart( int i )

PfrEichStop

void CALLBACK PfrEichStop( int i, double time )

PfrEichGet

double CALLBACK PfrEichGet( int p, int i )

LoadCalibration

int CALLBACK LoadCalibration( char *fname )

SaveCalibration

int CALLBACK SaveCalibration( char *fname )

directly call the functions of the class DACDRV with the same names (LoadCalibration ->> Load, SaveCalibration ->> Save). The description of the functions used for calibrating the tank system is to be found in the corresponding sections.

MeasureStart

int CALLBACK MeasureStart( double time, double trigger, double prestore, int tchannel, int slope )

Parameters

time is the total measuring time (in sec).

trigger is the trigger level for the trigger channel.

prestore is the time before the trigger condition is reached (in sec).

tchannel is the number of the trigger channel.

slope is a flag for the direction of the trigger condition.

float taint is the sampling period of the interrupt service routine.

Description

The function MeasureStart calls the function scope.StartMeasure to start a measured data recording. In advance the controller settings are copied to the global structure measctrlstatus.

See also

STOREBUF::StartMeasure
The group of functions

**MeasureLevel**

double CALLBACK MeasureLevel ( void )

**MeasureStatus**

int CALLBACK MeasureStatus ( void )

**SetDemo**

void CALLBACK SetDemo ( double se1, double se2, double se3, double sp1, double sp2, double sl1, double sl2, double sl3, double sv13, double sv32, double sv20 )

call directly the corresponding functions `scope.GetBufferLevel`, `scope.GetStatus` of the class `STOREBUF` and `drv.SetDemo` of the class `DACDRV`.

**See also**  `STOREBUF::GetBufferLevel`, `STOREBUF::GetStatus`, `DACDRV::SetDemo`

**IsDemo**

int CALLBACK IsDemo ( void )

**Description**

The function `IsDemo` returns a 1 only when the dtssrv32.dll is a DEMO version (generated with the macro `__DTS_DEMO__`, instead of the IO-adapter card a mathematical model is accessed). Otherwise the function returns 0.

**Return**

Is equal to 1 in case of a DEMO version, else equal to 0.
1.5.2 The DLL Interface PLOT

Included in the dtssrv32.dll, the functions of the file PLOT.CPP provide the interfaces for graphic output of measured data and for displaying information about the contents of documentation files (*.PLD).

Global Data:

HWND handlelist[100] is an array to store the handles of plot windows.
PROJECT project is a structure with data for the project identification.
CTRLSTATUS measctrlstatus is a structure containing the controller state, controller parameters as well as the measuring time at the time a measuring is started.
CTRLSTATUS ctrlstatus is a structure containing the controller state, controller parameters as well as the measuring time at the time a controller is started.
DATASTRUCT datastruct is structure containing the number of measurement-vectors, the number of its components as well as the sampling period of a measuring.
char FileName[60] is a string containing the name of a documentation file (*.PLD).
double **ppData is a pointer to a buffer containing measurements loaded from a documentation file (*.PLD).
int NumberOfCurvesInChannel is the number of curves of a plot depending on the "plot channels" (= selected groups of components of the measurement vector).
int ChannelToScope is the relation between curves (index) of the measurement buffer scope or the pointer **ppData and the "plot channels" (= selected groups of components of the measurement vector).
char *ScopeNames contains the curve descriptions (strings) for the linestyle table of the plot.
char *TitleNames contains the drawing titles for different "plot channels".
char *YAxisNames contains the description of the Y-axis for different "plot channels".
char *XAxisName contains the description of the X-axis for different "plot channels".
char *SensorYAxisName is the description of the Y-axis for the drawing with the sensor characteristics.
char *SensorXAxisName is the description of the X-axis for the drawing with the sensor characteristics.
char *SensorScopeNames contains the curve descriptions (strings) for the linestyle table of the drawing with the sensor characteristics.
char *SensorTitleName is the drawing title for the plot the sensor characteristics.
char *PumpYAxisName is the description of the Y-axis for the drawing with the pump characteristics.
char *PumpXAxisName is the description of the X-axis for the drawing with the pump characteristics.
char *PumpScopeNames contains the curve descriptions (strings) for the linestyle table of the drawing with the pump characteristics.
char *PumpTitleName is the drawing title for the plot of the pump characteristics.
**ReadPlot**

```c
int CALLBACK ReadPlot( char *lpfName )
```

**Parameters**

*lpfName* is a pointer to the name of a documentation file, from which measurements are to be read.

**Description**

The function *ReadPlot* reads the structures *project*, *ctrlstatus* and *datastruct* as well as the measurements from the documentation file with the given name *lpfName* and stores the measurements to a new global data array pointed to by **ppData**. Up to 59 characters of the file name *lpfName* are copied to the global file name *FileName*.

**Return**

The state of the file access: …

- 0, measurements read successfully,
- -1, file with the given name could not be opened,
- -2, the PROJECT structure from the file contains a wrong project number.

---

**WritePlot**

```c
int CALLBACK WritePlot( char *lpfName )
```

**Parameters**

*lpfName* is a pointer to the name of a documentation file, to which measurements are to be written.

**Description**

The function *WritePlot* writes the global structures *project*, *measctrlstatus*, the local structure *DATASTRUCT* *mydatastruct* as well as the content of the global measurement buffer *scope* to a documentation file with the given name *lpfName*. The local structure *mydatastruct* contains the number of measurement-vectors, the number of its components as well as the sampling period of a measuring.

**Return**

The state of the file access:

- 0, measurements written successfully,
- -1, file with the given name could not be created.
Plot

int CALLBACK Plot( int command, int channel )

Parameters

*command* defines the data source:

- 1, data from the global measurement buffer *scope*,
- 2, data from the global array **ppData**, 
- 3, at that time meaningless, return -4, 
- 4, data from the sensor characteristics, 
- 5, data from the plot characteristics.

*channel* defines the curves related to "plot channels":

- 0, Tank Levels [cm] (3 curves),
- 1, Tank Levels and Set points [cm] (5 curves),
- 2, Pump Flow rates [ml/s] (2 curves),
- 3, Pump Control Signals [ml/s] (2 curves),
- 4, Tank Levels [cm] and Pump Flow rates [ml/s] (5 curves),

Description

The function Plot represents the curves specified by *channel* with accompanying descriptions in a graphic inside a plot window. The data sources are the global measurement buffer *scope*, the global array **ppData** or the characteristics of the sensors or the pumps depending on the parameter *command*.

Return

The state of the graphic output:

- 0, successful graphic output of measured curves,
- -1, invalid values for *command*,
- -2, invalid values for *channel*,
- -3, length of the global array **ppData** is 0,
- -4, length of the global measurement buffer *scope* is 0.

See also

CreateSimplePlotWindow, SetCurveMode, AddAxisPlotWindow, AddXData, AddPlotTitle, ShowPlotWindow.
GetPlot

int CALLBACK GetPlot( int start, char *lpzName )

Parameters

start is a flag indicating the first plot window.

*lpzName is a pointer to the title of the plot window, the Windows handle of which was found.

Description

The function GetPlot determines the Windows handle of an existing plot window referenced by a local index index. The Windows handle is copied to the global list handlelist and the index is incremented. If the Windows handle is unequal to 0 up to 60 characters of the title of the corresponding plot window are copied to lpzName. With start=TRUE the Windows handle of the plot window with index=0 is determined.

Return

The state of the handle determination:

=0, handle = 0, plot window with current index could not be found,
=1, handle determined for current index, title copied.

See also

GetValidPlotHandle.

PrintPlot

int CALLBACK PrintPlot( int idx, HDC dcPrint, int iyOffset )

Parameters

idx is the index for the global list of handles referencing existing plot windows.

dcPrint is the device context of the output device.

iyOffset is the beginning of the printout in vertical direction as a distance in [mm] from the upper margin of a page.

Description

The function PrintPlot prints the content of the plot window with the Windows handle from the global list handlelist[ idx ] to the device with the device context dcPrint. The printout has a width of 180 mm and a height of 140 mm. It is located at the left margin with a distance of iyOffset mm from the upper margin of a (i.e. DIN A4) page.

Return

Is always equal to 0.

See also

PrintPlotWindow.
GetPldInfo

```c
int CALLBACK GetPldInfo( int &controller, int &decoupled, double &p, double &i, double &decoup, char **s, int &n, int &c, double &d )
```

**Parameters**

- `&controller` is a reference to the controller structure (open loop control, P controller decoupled, PI controller).
- `&decoupled` is a reference to the PI controller structure (with/without decoupling).
- `&p` is a reference to the proportional factor of the PI controller.
- `&i` is a reference to the integral factor of the PI controller.
- `&decoup` is a reference to the decoupling parameter.
- `**s` is a (double) pointer to the string containing date and time of the measuring.
- `&n` is a reference to the number of samples of each measured signal (curve).
- `&c` is a reference to the number of measured signals.
- `&d` is a reference to the sampling period of the measuring.

**Description**

The function `GetPldInfo` reads selected elements of the structures `ctrlstatus` as well as `datastruct` and stores these elements to the mentioned parameter references. It is assumed that the structures were filled previously with data from a loaded documentation file (*.PLD).

**Return**

Is the result:

- `=0`, the structure elements have been copied,
- `=-1`, the global data array `**ppData` does not exist, length = 0.

**See also**

`ReadPlot`. 
1.5.3 The Class LLS in the dtssrv32.dll

The class LLS provides the determination and handling of a sensor characteristic for one tank. The sensor characteristic is a straight line of the form \( h[cm] = u[Volt] \cdot m + b \) to calculate the liquid level \( h \) in cm with respect to the sensor voltage \( u \) in Volt.

**Public data:**

- double \( m \) is the gradient of the sensor characteristic.
- double \( b \) is the offset of the sensor characteristic.

**Private data:**

- double \( l1 \) is the liquid level for the lower calibration point (in cm).
- double \( l2 \) is the liquid level for the upper calibration point (in cm).
- double \( v1 \) is the liquid level for the lower calibration point (in Volt).
- double \( v2 \) is the liquid level for the upper calibration point (in Volt).
- int \( status \) is the status of the calibration data:
  - =0, default calibration data
  - =1, calibration data from file
  - =2, calibration data from measurement

---

**LLS::LLS()**

```cpp
void LLS( void )
```

**Description**

The constructor of the class LLS initializes values for the calibration data and calculates the gradient and the offset of the accompanying sensor characteristic (from 9 Volt correspond to 0 cm and -8.5 Volt correspond to 60 cm). The status of the calibration is reset to 0. The calibration points for the measurements are set to \( l1 = 20 \) and \( l2 = 50 \) cm.
### LLS::calccm

```c
double calccm( double volt )
```

**Parameters**

- `double volt` is the sensor value in Volt.

**Description**

The function `calccm` returns a liquid level in cm with respect to a sensor value (parameter `volt`) by determining the accompanying straight line equation.

**Return**

Liquid level (double) in cm for sensor value in Volt.

### LLS::calcv

```c
double calcv( double cm )
```

**Parameters**

- `double cm` is the liquid level in cm.

**Description**

The function `calcv` returns a sensor value in Volt with respect to a liquid level (parameter `cm`) by determining the accompanying straight line equation.

**Return**

Sensor value (double) in Volt for a liquid level in cm.

### LLS::SetL1

```c
void SetL1( double in )
```

**Parameters**

- `double in` is the liquid level of the lower calibration point.

**Description**

The function `SetL1` adjusts the liquid level in cm for the lower calibration point to the value of the parameter `in`.

### LLS::SetL2

```c
void SetL2( double in )
```

**Parameters**

- `double in` is the liquid level of the upper calibration point.

**Description**

The function `SetL2` adjusts the liquid level in cm for the upper calibration point to the value of the parameter `in`.
LLS::SetV1

void SetV1( double in )

Parameters
double in is the sensor value of the lower calibration point.

Description
The function SetV1 adjusts the sensor value in Volt for the lower calibration point to the value of the parameter in.

LLS::SetV2

void SetV2( double in )

Parameters
double in is the sensor value of the upper calibration point.

Description
The function SetV2 adjusts the sensor value in Volt for the upper calibration point to the value of the parameter in.

LLS::newsensor

void newsensor( void )

Description
The function newsensor calculates the gradient and the offset of the straight line equation of a sensor using the pair of values (level in cm / sensor in Volt) from two calibration points. The status of the calibration is set to 2.

LLS::GetLevel

double GetLevel( int which )

Parameters
int which selects the return value
=1, liquid level of the lower calibration point,
else, liquid level of the upper calibration point.

Description
The function GetLevel returns the liquid level in cm for the lower or upper calibration point depending on the parameter which.

Return
Liquid level (double) of the lower or upper calibration point.
LLS::Load

```c
int Load( char *name )
```

**Parameters**

char *name is a pointer to the file name.

**Description**

The function Load reads the gradient and offset for a straight line equation from a file with the name name (ASCII format). The status of the calibration is set to 1.

**Return**

State of the file access:

- 1 (TRUE), when reading from file was successful,
- 0 (FALSE), in any other case.

---

LLS::Save

```c
int Save( char *name )
```

**Parameters**

char *name is a pointer to the file name.

**Description**

The function Save stores the gradient and offset of a straight line equation to a file with the name name (ASCII format).

**Return**

State of the file access:

- 1 (TRUE), when writing to file was successful,
- 0 (FALSE), in any other case.

---

LLS::GetStatus

```c
int GetStatus( void )
```

**Description**

The function GetStatus returns the status of the calibration.

**Return**

Status (int) of the calibration data:

- 0, default calibration data
- 1, calibration data from file
- 2, calibration data from measurement
1.5.4 The Class OFC in the dtssrv32.dll

The class OFC determines the outflow coefficients of a three-tank-system. Upper liquid levels and lower liquid levels reached after a known period of time are substituted in the model equations to calculate the outflow coefficients.

**Private data:**

- **double ofc[3]** contains the outflow coefficients:
  - [0] = tank 1-3,
  - [1] = nominal outflow (tank 2),
- **double maxlevel[3]** are the upper liquid levels to determine the outflow coefficients.
- **double minlevel[3]** are the lower liquid levels to determine the outflow coefficients.
- **double dlevel[3]** are the differences between the upper and lower liquid levels.
- **double level[3]** are the mean values of the upper and lower liquid levels.
- **double time** is the time interval between the upper and lower liquid levels (switch-on time of the pumps).
- **int status** is the status of the outflow coefficients:
  - =0, default outflow coefficients
  - =1, outflow coefficients from file
  - =2, outflow coefficients from measurement

---

**OFC::OFC()**

```cpp
void OFC( void )
```

**Description**

The constructor of the class OFC sets initial values for the outflow coefficients (tanks 1-3 and 3-2 = 0.5 nominal outflow = 0.6). The status of the outflow coefficients is reset to 0.

---

**OFC::SetMaxLevel**

```cpp
void SetMaxLevel( double h1, double h2, double h3 )
```

**Parameters**

- **double h1** is the upper liquid level of tank 1.
- **double h2** is the upper liquid level of tank 2.
- **double h3** is the upper liquid level of tank 3.

**Description**

The function SetMaxLevel adjusts the upper liquid levels in cm to determine the outflow coefficients to the values of the given parameters.
OFC::SetMinLevel

```c
void SetMinLevel( double h1, double h2, double h3 )
```

**Parameters**
- `double h1` is the lower liquid level of tank 1.
- `double h2` is the lower liquid level of tank 2.
- `double h3` is the lower liquid level of tank 3.

**Description**
The function `SetMinLevel` adjusts the lower liquid levels in cm to determine the outflow coefficients to the values of the given parameters.

OFC::SetTime

```c
void SetTime( double t )
```

**Parameters**
- `double t` is the time interval between the liquid levels.

**Description**
The function `SetTime` adjusts the time interval the system needed to reach the lower liquid levels starting from the upper liquid levels to the value of the parameter `t`.

OFC::CalcOfc

```c
void CalcOfc( void )
```

**Description**
The function `CalcOfc` calculates the outflow coefficients with respect to the global variables for the upper and lower liquid levels and for the time interval using the mathematical model of the three-tank-system.

OFC::Load

```c
int Load( char *name )
```

**Parameters**
- `char *name` is a pointer to the file name.

**Description**
The function `Load` reads the outflow coefficients from a file with the name `name` (ASCII format). The status of the outflow coefficients is set to 1.

**Return**
- `1` (TRUE), when reading from file was successful,
- `0` (FALSE), in any other case.
OFC::Save

int Save( char *name )

Parameters
char *name is a pointer to the file name.

Description
The function Save stores the outflow coefficients to a file with the name name (ASCII format).

Return
State of the file access:

=1 (TRUE), when writing to file was successful,
=0 (FALSE), in any other case.

OFC::GetStatus

int GetStatus( void )

Description
The function GetStatus returns the status of the outflow coefficients.

Return
Status (int) of the outflow coefficients:

=0, default outflow coefficients
=1, outflow coefficients from file
=2, outflow coefficients from measurement
1.5.5 The Class PFR in the dtssrv32.dll

The class PFR provides determination and handling of a pump characteristic for a tank system. The pump characteristic is defined by 9 base points. Each base point contains the correlation between a known control signal in Volt and the resulting pump flow rate in ml/s. Values between the base points are obtained by linear interpolation. The control signal for the base points is preassigned in the range from -10 Volt to +10Volt with a step of 2,5 Volt. The pump flow rate is calculated from the changes in the liquid levels during a known time period.

**Public data:**

- `int i` is the index of a base point belonging to a pump characteristic.
- `int status` is the status of the pump characteristic:
  - `=0`, default pump characteristic
  - `=1`, pump characteristic from file
  - `=2`, pump characteristic from measurement
- `int calibration` is a flag for the pump characteristic.
- `int delay` is a flag for the pump characteristic.
- `double deltatime` is the time period between the initial and final value of a liquid level.
- `double level1` is the initial value of the liquid level.
- `double level2` is the final value of the liquid level.
- `double pump[9]` contains the base points of the pump characteristic (flow rate [ml/s]).

---

**PFR::PFR()**

```cpp
void PFR( void )
```

**Description**

The constructor of the class PFR initializes default values for the pump characteristic in the range of 0 to 100 ml/s with a step of 12,5 ml/s according to the base points of the control signal in the range of -10 V to +10 V with a step of 2,5 V. The status of the pump characteristic as well as the flags are reset to 0.

---

**PFR::pumpi2Q**

```cpp
double pumpi2Q( int k )
```

**Parameters**

- `int k` is the index for the base point of the pump characteristic.

**Description**

The function pumpi2Q returns the value of the base point of the pump characteristic referenced by an index (parameter `k`)

**Return**

Pump flow rate (double) in ml/s belonging to the index.
PFR::pumpQ2V

double pumpQ2V( double q )

Parameters
double q is the desired pump flow rate.

Description
The function pumpQ2V returns the control signal in Volt determined by interpolation of the pump characteristic which is required for a desired pump flow rate in ml/s (Parameter q).

Return
Control signal (double) in Volt required for a desired pump flow rate in ml/s.

PFR::GetVoltage

double GetVoltage( void )

Description
The function GetVoltage returns the control signal in Volt referenced by the current index (public data i).

Return
Control signal (double) in Volt referenced by an index.

PFR::SetOut

void SetOut( int in )

Parameters
int in is the index for the pump characteristic.

Description
The function SetOut adjusts the current index to the value of the parameter in and sets the flag calibration to 1.

PFR::SetMeasureLevel1

void SetMeasureLevel1( double l1 )

Parameters
double l1 is the initial liquid level for calibration.

Description
The function SetMeasureLevel1 adjusts the initial value of the liquid level used for the pump calibration to the value of the parameter l1.
PFR::SetMeasureLevel2

void SetMeasureLevel2( double l2 )

Parameters

double l1 is the final liquid level for calibration.

Description

The function SetMeasureLevel2 adjusts the final value of the liquid level used for the pump calibration to the value of the parameter l2.

---

PFR::SetMeasureTime

void SetMeasureTime( double t )

Parameters

double l1 is the time period between the initial and final value of the liquid levels.

Description

The function SetMeasureTime adjusts the time period, which was needed to reach the final value of the liquid level starting with the initial value, to the (measured) value of the parameter t.

---

PFR::CalcQzu

double CalcQzu( int i )

Parameters

int i is the base point index for the pump characteristic.

Description

The function CalcQzu returns the flow rate belonging to a base point of the pump characteristic referenced by an index (parameter i). The flow rate is calculated using the current values of the initial and final liquid levels as well as the time period. The status of the pump characteristic is set to 2 when the flow rate of the last base point is calculated.

Return

Pump flow rate (double) in ml/s referenced by an index.
PFR::Load

```c
int Load( char *name )
```

**Parameters**

char *name is a pointer to the file name.

**Description**

The function Load reads the base points of the pump characteristic from a file with the name name (ASCII format). The status of the pump characteristic is set to 1.

**Return**

State of the file access:

- 1 (TRUE), when reading from file was successful,
- 0 (FALSE), in any other case.

PFR::Save

```c
int Save( char *name )
```

**Parameters**

char *name is a pointer to the file name.

**Description**

The function Save stores the base points of the pump characteristic together with the accompanying control signal to a file with the name name (ASCII format).

**Return**

State of the file access:

- 1 (TRUE), when writing to file was successful,
- 0 (FALSE), in any other case.

PFR::GetStatus

```c
int GetStatus( void )
```

**Description**

The function GetStatus returns the status of the pump characteristic.

**Return**

The status (int) of the pump characteristic:

- 0, default pump characteristic
- 1, pump characteristic from file
- 2, pump characteristic from measurement.
1.5.6 The Class TUSTINPI in the dtssrv32.dll

The class TUSTINPI provides the calculation of a digital PI controller obtained by an approximation of an analog PI controller. The coefficients of the digital controller result from applying the Tustin relation on the transfer function of the analog controller:

analog controller: \[ G(s) = k_i \frac{1}{s} + k_p \]

with the Tustin relation: \[ s = \frac{2}{T_a} \frac{(z-1)}{(z+1)} \]

digital controller: \[ y(k+1) = y(k) + z_0 \cdot x_d(k) + z_1 \cdot x_d(k-1) \]

where \[ z_0 = k_i \frac{T_a}{2} + k_p, \]
\[ z_1 = k_i \frac{T_a}{2} - k_p \]

Private data:

double \( t_a \) is the sampling period of the controller.
double \( k_i \) is the integral portion of the analog controller.
double \( k_p \) is the proportional portion of the analog controller.
double online_zk0 \( z_0 \) coefficient of the active digital controller.
double \( zk0 \) \( z_0 \) coefficient of the digital controller.
double online_zk1 \( z_1 \) coefficient of the active digital controller.
double \( zk1 \) \( z_1 \) coefficient of the digital controller.
double \( y \) is the control signal from the previous sampling period.
double \( x_d_1 \) is the control error signal from the previous sampling period.
int \( newz \) is a flag indicating new coefficients for the active controller.

TUSTINPI::TUSTINPI()

```cpp
void TUSTINPI ( void )
```

Description

The constructor of the class TUSTINPI adjusts initial values for the coefficients of the analog controller (to 1 resp. 0), for the values of the digital controller belonging to the previous sampling period \( (y=x_d_1=0) \) and calculates the coefficients of the digital controller.
TUSTINPI::SetKP

void SetKP( double k )

Parameters
double k is the proportional amplification of the analog controller.

Description
The function SetKP adjusts the proportional portion of the analog PI controller to the value of the given parameter and calculates the coefficients of the accompanying digital controller.

TUSTINPI::GetKP
double GetKP( void )

Description
The function GetKP returns the adjusted proportional portion of the analog PI controller.

Return
The proportional portion (double) of the analog controller.

TUSTINPI::SetKI

void SetKI( double k )

Parameters
double k is the integral portion of the analog controller.

Description
The function SetKI adjusts the integral portion of the analog PI controller to the value of the given parameter and calculates the coefficients of the accompanying digital controller.

TUSTINPI::GetKI
double GetKI( void )

Description
The function GetKI returns the adjusted integral portion of the analog PI controller.

Return
The integral portion (double) of the analog controller.
TUSTINPI::SetTA

void SetTA( double t )

Parameters
double t is the sampling period of the digital controller.

Description
The function SetTA adjusts the sampling period in sec of the digital controller to the value of the given parameter and calculates the coefficients of the accompanying digital controller.

TUSTINPI::GetTA

double GetTA( void )

Description
The function GetTA returns the adjusted sampling period of the digital controller.

Return
The sampling period (double) of the digital controller.

TUSTINPI::ResetControl

void ResetControl( void )

Description
The function ResetControl resets the past values of the digital controller (control signal y and control error signal xd from the previous sampling period) to 0.

TUSTINPI::Control

double Control( double xd )

Parameters
double xd is the current control error signal of the digital controller.

Description
The function Control calculates the digital controller with respect to the given control error signal xd and returns the control signal limited to the range +/-100. In case the flag newz was set, at first the coefficients of the digital controller are copied to those of the active controller. The flag is reset after this operation.

Return
The control signal (double) of the digital controller.
TUSTINPI::CalcZCoef

void CalcZCoef( void )

Description

The function CalcZCoef calculates the coefficients of the digital controller by applying the Tustin relation on the transfer function of the analog controller. The Flag newz is set to 1.
1.5.7 The Class STOREBUF in the dtssrv32.dll

The instance of the class STOREBUF realizes the function of data buffering. The data buffer created dynamically looks like a matrix with a maximum of assignable rows, where each row contains an adjustable number of components (i.e. float values from measurements). The storage in the data buffer is performed row by row, where each row is represented by a data vector, which was filled by another routine from an upper level. In this case it is the interrupt service routine which fills the data vector, i.e. with the set point value, measurements and control signals, in every sampling period. An element function (StartMeasure) of STOREBUF starts and controls the storage (WriteValue) of this data vector in the data buffer. With respect to the measuring time at first those sampling periods are determined in which storage is to be performed (number of store operations * sampling periods = measuring time). Where the number of store operations is calculated at first such that it is always less than the maximum number of measurement vectors (= number of rows of the memory matrix). At the end of the measuring time the store operation is terminated in case no additional trigger conditions are set. In case of an activated trigger condition, a signal crosses a given value with a selected direction, the store operation is continued until the end of the measuring time after the trigger condition was met. In case the signal does not meet the trigger condition, the store operation is performed endless in a ring until the user interactively terminates this operation. In addition a time before the trigger condition (prestore time) is adjustable in which storage in the data buffer is performed. The time after the trigger condition is met is then the measuring time reduced by the preset time. The mentioned data vector will be named measurement-vector in the following.

Private Data:

float \textit{taplt} is the sampling period of the interrupt service routine.
int \textit{trigger\_channel} is the channel (index) of the measurement-vector used for triggering.
int \textit{startmessung} flag for starting new measuring.
int \textit{gomessung} flag for measuring is started.
int \textit{storedelay} is the number of sampling periods in between the storage of values.
int \textit{storedelayi} is the counter for \textit{storedelay}.
int \textit{MaxVectors} is the maximum number of storable measurement-vectors.
int \textit{anzahl} is the number of stored measurement-vectors.
int \textit{anzahli} is the counter for \textit{anzahl}.
int \textit{stopmeasureindex} is the index for normal end of measuring.
int \textit{triggerindex} is the trigger index.
int \textit{prestoreoffset} is the number of stored measurement-vectors previous to the trigger.
int \textit{nchannel} is the number of float values in the measurement-vector.
int \textit{outchannel} is the channel (index) of the component of the measurement-vector, which is to be read (for output).
int \textit{bufindex} is an internal index for the next storage location in the data buffer.
int \textit{triggered} flag for trigger condition is met.
int \textit{stored\_values} number of measurement-vector storages since the start of the measuring.
float \textit{trigger\_value} trigger float value.
float *\textit{fptr} is a pointer to the start address of the dynamic data buffer.
float *\textit{sourceptr} is a pointer to the measurement-vector.
float *\textit{inptr} is a pointer to the actual data buffer location.
int \textit{aktiv} flag for status of the dynamic data buffer.
int \textit{status} flag for storage control.
STOREBUF::ResetBufIndex

```c
void ResetBufIndex( void )
```

**Description**

The private element function `ResetBufIndex` sets `bufindex` to 0 and `inptr` equal to `fptr` meaning that the start conditions for the data buffer are set.

STOREBUF::STOREBUF

```c
STOREBUF( int nchannel, float *indata, int maxvectors )
```

**Parameters**

- `int nchannel` is the number of float values of the external measurement-vector.
- `float *indata` is the pointer to the start address of the measurement-vector.
- `int maxvectors` is the maximum number of measurement-vectors.

**Description**

The constructor of this class initializes flags (`gomessung, startmessung, aktiv = FALSE`) to control the storage as well as a pointer to the measurement vector (`sourceptr = indata`). The maximum number of the measurement-vectors is set (`MaxVectors = maxvectors`) where the minimum value is limited to 1.

STOREBUF::~STOREBUF()

```c
void ~STOREBUF( void )
```

**Description**

The destructor of this class frees the dynamically allocated memory `fptr` in case it was created.
STOREBUF::StartMeasure

```c
void StartMeasure( float meastime, float triggervalue, float prestoretime, int triggerdir,
float taint )
```

**Parameters**

- `triggerchannel` is the number of the trigger channel.
- `meastime` is the measuring time in seconds.
- `triggervalue` is the trigger level of the trigger channel.
- `prestoretime` is the time of storage previous to the trigger (in sec).
- `triggerdir` is the flag for direction (below/above) of the trigger condition.
- `taint` is the sampling period of the interrupt service routine.

**Description**

The function `StartMeasure` initializes a new storage operation. To a maximum of `maxvectors` measurement-vectors are stored. In case the adjusted measuring time `meastime` is longer than `maxvectors * taint` (sampling period) the number of interrupt executions without data storage is calculated. The arguments of this function are all the parameters required for the storage.

STOREBUF::WriteValue

```c
void WriteValue( void )
```

**Description**

The function `WriteValue` stores `nchannel` float values from the array `indata` (measurement-vector) to the current address of the dynamically allocated array.

STOREBUF::SetOutChan

```c
void SetOutChan(int in)
```

**Parameters**

- `int in` references the component of the measurement vector which is to be read (output).

**Description**

The inline function `SetOutChannel` sets the channel number (index in the measurement-vector) of the signal which is to be returned by the function `ReadValue`. 
STOREBUF::ReadValue

float ReadValue( void )

Description
The function ReadValue returns the value of the next storage location belonging to the channel selected by SetOutChannel.

Return
Value (float) read from measurement-vector.

STOREBUF::GetBufLen

int GetBufLen( void )

Description
The function GetBufLen interrupts a current storage operation and returns the number of stored measurement-vectors.

Return
Number (int) of stored measurement-vectors.

STOREBUF::GetBufTa

float GetBufTa( void )

Description
The inline function GetBufTa returns the time between storage, which was calculated with respect to the measuring time and the sampling period.

Return
Time (float) between storage depending on measuring time and sampling period.

STOREBUF::GetStatus

int GetStatus( void )

Description
The function GetStatus returns the status of the store operation.

Return
Status (int) of store operation

0  not initialized
1  storage before trigger
2  waiting for trigger condition
4  storage operation
5  storage complete
6  storage interrupted
STOREBUF::GetBufferLevel

double GetBufferLevel( void )

Description
The function GetBufferLevel returns the percentage of the former measurement time with respect to the given trigger condition (= filling ratio or level of the data buffer). The return value will stay at 0% until the valid trigger condition is reached even when prestoretime is unequal to zero. That means the return value will start with an initial value of prestoretime / meastime in % at the time of a valid trigger condition.

Return
The percentage of the filling ratio (double) of the data buffer.
1.5.8 The Class AFBUF in the dtssrv32.dll

An instance of the class AFBUF is an object that creates dynamically a data array for an assignable number of float values. Data can be stored in this array and can be read afterwards when the data array is filled completely. The array is handled like a ring buffer.

Private data:

float *fptr is the pointer to the start of the dynamically created data array.
float *inptr is the pointer to the current storage location ready to store a value (input).
float *outptr is the pointer to the current storage location ready to read a value (output).
int aktiv flag: dynamic memory is initialized.
int filled flag: data array is filled.
int abuflen is the number of float values in the data array.
int inbufindex is the index of the current input position.
int outbufindex is the index for the current output position.

AFBUF::AFBUF()

void AFBUF( void )

Description

The constructor of this class resets the flag aktiv, which indicates a dynamically created data array.

AFBUF::~AFBUF()

void ~AFBUF( void )

Description

The destructor of this class frees the initialized data memory in case it was created dynamically.

AFBUF::NewFBuf

int NewFBuf( int anzahl )

Parameters

anzahl is the size of the data array in float values.

Description

The function NewFBuf initializes a data array with anzahl float values. A value of 1 is returned after a successful initialization, otherwise 0 is returned.

Return

Status (int) of data array:

= 0, data array is not initialized,
= 1, data array is initialized.
AFBUF::ReadFBuf

float ReadFBuf( void )

Description
The function ReadFBuf returns the float value of the next storage location of the dynamically created data array in case this array was filled previously.

Return
Value (float) from data array.

AFBUF::WriteFBuf

int WriteFBuf( float fvalue )

Parameters
float fvalue is the value, which is to be stored.

Description
The function WriteFBuf stores the float value to the storage location.

Return
Number (int) of stored values (=0 in case no data array initialized).
1.5.9 The Class TWOBUFFER in the dtssrv32.dll

The class TWOBUFFER handles two instances of the class AFBUF. One instance (write-instance) can be used to store data while the other is used to read out data (read-instance). In case the data array of the write-instance is filled it is handled as a read-instance in the following. This condition guarantees that the interrupt service routine has always access to valid data.

Private objects:

AFBUF Buf1 is an instance of the class AFBUF
AFBUF Buf2 is an instance of the class AFBUF

Private data:

int readbuffer flag: data array is ready for read operation.
int buffer1 flag:
    0 = Buf1 write,
    1 = Buf1 read.
int buffer2 flag:
    0 = Buf2 write,
    1 = Buf2 read.
int newbuffer flag:
    0 = not a new output buffer,
    1 = Buf1 is a new output buffer,
    2 = Buf2 is a new output buffer.
int buf1len length of the data array of the instance Buf1
int buf1leni index of the instance Buf1.
int buf2len length of the data array of the instance Buf2.
int buf2leni index of the instance Buf2.
int inbufindex index for input data array.
int repw1 number of repeated values in Buf1
int repw1i index of repeated values in Buf1
int repw2 number of repeated values in Buf2
int repw2i index of repeated values in Buf2
int repb1 number of data array outputs of the instance Buf1.
int repb1i index of the array outputs of the instance Buf1.
int repb2 number of data array outputs of the instance Buf2.
int repb2i index of the array outputs of the instance Buf2.
TWOBUFFER::TWOBUFFER()

void TWOBUFFER( void )

Description
The constructor of this class initializes flags and counters as follows:
readbuffer = FALSE, buffer cannot be read,
buf1len = 1, length of the buffer Buf1,
buf2len = 1, length of the buffer Buf2,
buffer1 = 1, buffer Buf1 for read operation,
buffer2 = 0, buffer Buf2 for write operation,
newbuffer = 0, no buffer for read or write operation available.

TWOBUFFER::New2Buffer

void New2Buffer( int anzahl, int repeatwert, int repeatbuf )

Parameters
int anzahl is the number of float values of the new array.
int repeatwert defines how often a value is to be repeated during a read operation by Read2Buffer.
int repeatbuf defines how often the array is to be sent to the output.

Description
The function New2Buffer creates data arrays dynamically with anzahl float values. With buffer1 = 0 Buf1 is created and with buffer2 = 0 Buf2 is created.

TWOBUFFER::Write2Buffer

int Write2Buffer( float wert )

Parameters
float wert is the value, which is to be stored.

Description
The function Write2Buffer writes the argument value to the data array. In case the end of the array is reached, the array is used as a source for the function Read2Buffer.

Return
Total number (int) of stored (written) values.
**TWOBUFFER::Read2Buffer**

```c
float Read2Buffer( void )
```

**Description**

The function `Read2Buffer` returns the values of the read-array handling like a ring. In case the argument `repeatbuf` of the function `New2Buffer` was equal to `x`, the array is read `x` times. After `x` read operations zero is returned. In case `repeatbuf` is equal to 0, the read operation is cyclic.

**Return**

Value (float), which is read from the array.
1.5.10 The Class Signal in the dtssrv32.dll

An instance of the class SIGNAL is an object to create a data array, which represents a given signal shape in case it is read out with constant time intervals. To do this an instance of the class TWOBUFFER is used. Adjustable signal shapes are rectangle, triangle, sawtooth and sine. In addition the amplitude, an offset and the time period is adjustable.

Private Data:

- float `abtastzeit` sampling period to read out values.
- float `stuetzst` number of base points of a signal period.
- float `signaloffset` offset of the signal.
- float `signalamplitude` amplitude of the signal.
- float `minrange` minimum available return value.
- float `maxrange` maximum available return value.

Private objects:

TWOBUFFER `sign` is an instance of the class TWOBUFFER

```
SIGNAL::SIGNAL( )
```

```c
void SIGNAL( void )
```

Description

The constructor of this class initializes the variables `abtastzeit`, `minrange` and `maxrange`.

```
SIGNAL::InitTime
```

```c
float InitTime( float settime )
```

Parameters

- float `settime` is the sampling period of the read routine (in sec.)

Description

The function `InitTime` sets the sampling time, which is used to read out the values from the interrupt routine, equal to the given controller sampling period.

Return

Adjusted sampling period (float) in seconds.
**SIGNAL::MakeSignal**

```c
int MakeSignal( int form, float offset, float ampl, float periode, int repeatbuf)
```

**Parameters**

- `int form` is the signal shape indicator
  - konstform (constant) 0
  - rectform (rectangle) 1
  - triform (triangle) 2
  - saegeform (sawtooth) 3
  - sinusform (sine) 4
- `float offset` offset value of the signal.
- `float ampl` amplitude of the signal.
- `float periode` period of the signal (in sec).
- `int repeatbuf` defines how often the signal is to be read out (0 = continuously).

**Description**

The function `MakeSignal` generates a data array with a maximum of 1024 float values, in which the values of the selected signal shape are stored. The signal shape is adjusted by the argument `form`.

The absolute value of the signal \( f(t) \) is given by the sum \( offset + ampl \times f(t) \).

In case the number of base points determined by the division \( periode / \) sampling period is greater than 1024 the number of base points is halved and the repeat value `repw1` or `repw2` is doubled until the number is less than 1024.

After the generation of a data array it is assigned as a source to the function `ReadNextValue` (see class `TWOBUFFER`).

**Return**

Error status:

- =0, no error
- =1, illegal signal shape.

**SIGNAL::ReadNextValue**

```c
float ReadNextValue( void )
```

**Description**

The function `ReadNextValue` reads the data from the assigned array. The value is internally limited to the range `minrange` to `maxrange`. It is called by the interrupt service routine. Due to the locking mechanism in `TWOBUFFER`, new signal shapes can be created even in case the active interrupt outputs another one.

**Return**

Value (float) read from the data array.
SIGNAL::SetRange

void SetRange( float min, float max )

Parameters
float min is the minimum return value of the function ReadNextValue.
float max is the maximum return value of the function ReadNextValue.

Description
The function SetRange adjusts the range of the base points forming the signal, i.e. the minimum and maximum values returned by the function ReadNextValue.

SIGNAL::WriteBuffer

void WriteBuffer( float value )

Parameters
float value is the value which has to be stored.

Description
The private element function WriteBuffer writes the argument value to the data array of the instance TWOBUFFER.

SIGNAL::Stuetzstellen

int Stuetzstellen( float Periodenzeit, int form )

Parameters
float Periodenzeit is the time period of the signal.
int form is the indicator for the adjusted signal shape.

Description
The private element function Stuetzstellen calculates the number of base points and with this the length of the data arrays of the instance TWOBUFFER depending on the time period and the signal shape. The number of the base points is determined by the division Periodenzeit / sampling period. In case of a constant signal shape the minimum number of base points is 1.

Return
Calculated number (int) of base points.
1.6 Driver Functions in the DTSSRV32.DLL

1.6.1 The Class DACDRV

The class DACDRV provides the interface between the DTS200 controller program and the driver functions of the PC plug-in card. The class WDAC98 containing the driver functions is the basic class of DACDRV. In addition this class contains the mathematical model of the Three-Tank-System when it is compiled with '#define __DTS_DEMO__' (see file DTSDEFIN.H). With this all program functions except for the calibration can be carried out for a simulated Three-Tank-System. The PC plug-in card is no longer required in this case. This program version will be called 'Demo-Version' in the following.

Basic Class:

WDAC98 driver functions of the PC adapter card (file WDAC98.CPP)

The files DACDRV.H, DACDRV.CPP contain the class DACDRV with the functions:

DACDRV( void )
~DACDRV(){
  
  void SetTankDims( double at1, double at2, double at3 )
  
  double ReadTank1( void )
  
  double ReadTank2( void )
  
  double ReadTank3( void )
  
  void SetPumpFlow1( double q )
  
  void SetPumpFlow2( double q )
  
  #ifdef __DTS_DEMO__
    void SetDemo( double se1, double se2, double se3, double sp1, double sp2, double sl1, double sl2, double sl3, double sv13, double sv32, double sv20)
  
    void GetSensorErrors( double& SensErr1, double& SensErr2, double& SensErr3 )
  
    void GetPumpErrors( double& PumpErr1, double& PumpErr2 )
  
    void ResetSimSystem( void )
  
    void SetNoise( double SignalNoise )
  
    double SensNoise( void )
  
  #endif
  
  double SenGetLevel( int mode )
  
  void SenSetLevel( int mode, double val )
  
  void SenEichLevel( int mode )
  
  double SenGetVolt( int mode, double val )
  
  void OfcEichStart( void )
  
  void OfcEichStop( void )

double OfcEichGet(int i)

void PfrEichStart(int i)

void PfrEichStop(int i, double time)

double PfrEichGet(int p, int i)

int Save(char* FileName)

int Load(char* FileName)

int eichok(void)

int CheckSystem(void)

int CheckFree(void)

virtual void StartInterrupt(void)

virtual void TriggerEndstufe(void)

Private Data:

LLS DTS200sen1 is an object to determine and handle the characteristic of level sensor 1
LLS DTS200sen2 is an object to determine and handle the characteristic of level sensor 2
LLS DTS200sen3 is an object to determine and handle the characteristic of level sensor 3
OFC DTS200ofc is an object to determine and handle the outflow coefficients
PFR DTS200pump1 is an object to determine and handle the characteristic of the pump 1
PFR DTS200pump2 is an object to determine and handle the characteristic of the pump 2
double Level1 is the liquid level of tank 1 in [cm]
double Level2 is the liquid level of tank 2 in [cm]
double Level3 is the liquid level of tank 3 in [cm]
double Volt1 is the liquid level of tank 1 in [Volt]
double Volt2 is the liquid level of tank 2 in [Volt]
double Volt3 is the liquid level of tank 3 in [Volt]
int Tick is a flag for the sensor calibration
unsigned long EichTick is a time counter for the pump calibration

#define __DTS_DEMO__

double q1 is the flow rate of the pump 1
double q2 is the flow rate of the pump 2
double q13 is the cross flow from tank 1 to tank 3
double q32 is the cross flow from tank 3 to tank 2
double q20 is the outflow of tank 2
double q11 is the leakage flow from tank 1
double q12 is the leakage flow from tank 2
double q13 is the leakage flow from tank 3
double SensError1 is the error of sensor 1
double SensError2 is the error of sensor 2
double SensError3 is the error of sensor 3
double PumpError1 is the error of pump 1
double PumpError2 is the error of pump 2
double l1 is a measure for the leakage at tank 1
double l2 is a measure for the leakage at tank 2
double l3 is a measure for the leakage at tank 3
double v13 is a measure for the clog between tank 1 and tank 3
double v32 is a measure for the clog between tank 3 and tank 2
double v20 is a measure for the clog in the outflow from tank 2
int AddNoise is a flag for the noise signal
double noise is the amplitude of the sensor noise signal
#endif

DACDRV::DACDRV

DACDRV( void )

Description

The constructor of the class DACDRV initializes an object of the class WDAC98 as well as all objects required to determine and handle the characteristics and outflow coefficients (classes LLS, PFR, OFC). The liquid levels in [cm] and [Volt] and for the Demo-Version the error signals as well as the real pump control signals are reset to 0. Finally the file DE-FAULT.CAL containing the calibration data (characteristics and outflow coefficients) of the tank system is loaded. A message box with the error message "Can't open file: DE-FAULT.CAL" will appear, when this file does not exist.
DACDRV::~DACDRV

~DACDRV( void )

Description
The destructor of the class DACDRV resets the real pump control signals to 0.

DACDRV::SetTankDims

void SetTankDims(double at1, double at2, double at3 )

Parameters
at1 is the effective cross section of tank 1
at2 is the effective cross section of tank 2
at3 is the effective cross section of tank 3

Description
The function SetTankDims copies the given values to the variables for the effective cross sections of the tanks only when they are limited to the range >0 and <1000.

DACDRV::ReadTank1

double ReadTank1( void )

Description
The function ReadTank1 reads the sensor for the liquid level in tank 1 and calculates the accompanying liquid level in [cm] with respect to the corresponding sensor characteristic. This value is returned by the function.

For the DEMO-Version the return value is equal to the liquid level \( h \) calculated by means of the mathematical model. This value may be superimposed by an error and noise signal according to the following relation:

\[
\text{level[cm]} = h \times (1 - \text{sensor error}) + \text{noise signal}
\]

The noise signal is added only when the corresponding flag is set. Negative liquid levels are limited to 0,0.

Return:
The liquid level of tank 1 (in DEMO-Version with errors).
**Source Files of the DTS200 Controller Program**

**Three – Tank – System DTS200**

---

**DACDRV::ReadTank2**

double **ReadTank2** ( void )

**Description**

The function **ReadTank2** reads the sensor for the liquid level in tank 2 and calculates the accompanying liquid level in [cm] with respect to the corresponding sensor characteristic. This value is returned by the function.

For the DEMO-Version the return value is equal to the liquid level \( h \) calculated by means of the mathematical model. This value may be superimposed by an error and noise signal according to the following relation:

\[
\text{level[cm]} = h \times (1 - \text{sensor error}) + \text{noise signal}
\]

The noise signal is added only when the corresponding flag is set. Negative liquid levels are limited to 0,0.

**Return:**

The liquid level of tank 2 (in DEMO-Version with errors).

---

**DACDRV::ReadTank3**

double **ReadTank3** ( void )

**Description**

The function **ReadTank3** reads the sensor for the liquid level in tank 3 and calculates the accompanying liquid level in [cm] with respect to the corresponding sensor characteristic. This value is returned by the function.

For the DEMO-Version the return value is equal to the liquid level \( h \) calculated by means of the mathematical model. This value may be superimposed by an error and noise signal according to the following relation:

\[
\text{level[cm]} = h \times (1 - \text{sensor error}) + \text{noise signal}
\]

The noise signal is added only when the corresponding flag is set. Negative liquid levels are limited to 0,0.

**Return:**

The liquid level of tank 3 (in DEMO-Version with errors).
DACDRV::SetPumpFlow1

```c
void SetPumpFlow1( double q )
```

**Parameters**

$q$ is the desired flow rate of pump 1

**Description**

The function `SetPumpFlow1` determines the control signal in [Volt] required for a desired flow rate $q$ of the pump 1 by evaluation of the pump characteristic. This control signal is provided at the D/A converter output. During the determination of the pump characteristic ($\text{calibration} = 1$) the accompanying base point values are provided at the output.

The D/A converter output is omitted in the DEMO-Version. The flow rate for the mathematical model is calculated as follows

\[
\text{Flow rate [ml/s]} = q \ast (1 - \text{pump error})
\]

---

DACDRV::SetPumpFlow2

```c
void SetPumpFlow2( double q )
```

**Parameters**

$q$ is the desired flow rate of pump 2

**Description**

The function `SetPumpFlow2` determines the control signal in [Volt] required for a desired flow rate $q$ of the pump 2 by evaluation of the pump characteristic. This control signal is provided at the D/A converter output. During the determination of the pump characteristic ($\text{calibration} = 1$) the accompanying base point values are provided at the output.

The D/A converter output is omitted in the DEMO-Version. The flow rate for the mathematical model is calculated as follows

\[
\text{Flow rate [ml/s]} = q \ast (1 - \text{pump error})
\]

The flow rates of both pumps are limited to the allowed range. The mathematical model of the Three-Tank-System is calculated with respect to the possibly adjusted leakage flows and clogs.

\[
\begin{align*}
\text{coef1} &= \text{outflow coefficient1} \ast \text{valve section} \\
\text{coef2} &= \text{outflow coefficient2} \ast \text{valve section} \\
\text{coef3} &= \text{outflow coefficient3} \ast \text{valve section} \\
\end{align*}
\]

\[
\begin{align*}
\text{leakage flows:} \\
q_1 &= \text{coef2} \ast l_1 \ast \text{sqrt}(2 \ast g \ast \text{old LEVEL1}) \\
q_2 &= \text{coef2} \ast l_2 \ast \text{sqrt}(2 \ast g \ast \text{old LEVEL2}) \\
q_3 &= \text{coef2} \ast l_3 \ast \text{sqrt}(2 \ast g \ast \text{old LEVEL3}) \\
\end{align*}
\]

\[
\begin{align*}
\text{cross flows:} \\
\text{delta_h13} &= \text{old LEVEL1} - \text{old LEVEL3} \\
\text{delta_h32} &= \text{old LEVEL3} - \text{old LEVEL2} \\
q_{13} &= \text{coef1} \ast (1 - \nu13) \ast \text{sqrt}(2 \ast g \ast \text{delta_h13}) \\
\end{align*}
\]
\[ q_{32} = \text{coef}_2 \times (1 - v_{32}) \times \sqrt{2g \times \delta_{h32}} \]
\[ q_{20} = \text{coef}_3 \times (1 - v_{20}) \times \sqrt{2g \times \text{old}_\text{Level}_2} \]

\[ T = \text{sampling period} \]
\[ A = \text{tank cross section} \]

\[ \text{Level}_1 = (\text{flowrate}_1 - q_{13} - q_{l1}) \times T/A + \text{old}_{\text{level}1} \]
\[ \text{Level}_2 = (\text{flowrate}_2 + q_{32} - q_{20} - q_{l2}) \times T/A + \text{old}_{\text{level}2} \]
\[ \text{Level}_3 = (q_{13} - q_{32} - q_{l3}) \times T/A + \text{old}_{\text{level}3} \]

The calculated liquid levels are limited to the range 0 to 60 cm.

**DACDRV::SetDemo**

```
void SetDemo (double se1, double se2, double se3, double sp1, double sp2, double sl1,
             double sl2, double sl3, double sv13, double sv32, double sv20)
```

**Parameters**

- `se1` is the error of sensor 1
- `se2` is the error of sensor 2
- `se3` is the error of sensor 3
- `sp1` is the error of pump 1
- `sp2` is the error of pump 2
- `sl1` is a measure for the leakage at tank 1
- `sl2` is a measure for the leakage at tank 2
- `sl3` is a measure for the leakage at tank 3
- `sv13` is a measure for the clog between tank 1 and tank 3
- `sv32` is a measure for the clog between tank 3 and tank 2
- `v20` is a measure for the clog in the outflow of tank 2

**Description**

The function **SetDemo** adjusts the sensor and pump errors as well as the measures for leakages and clogs for the simulation of the Three-Tank-System. The errors are considered as follows:

- liquid level [cm] = \( h \times (1 - \text{sensor error}) \)
- flow rate [ml/s] = \( q \times (1 - \text{pump error}) \)
- leakage flow [ml/s] = \( \text{coef} \times \text{leakage error} \times \sqrt{2g \times \text{level}} \)
- cross flow [ml/s] = \( \text{coef} \times (1 - \text{clog}) \times \sqrt{2g \times \text{delta_level}} \)
DACDRV::GetSensorErrors

void GetSensorErrors ( double& SensErr1, double& SensErr2, double& SensErr3 )

Parameters
SensErr1 is a pointer reference to the error of sensor 1
SensErr2 is a pointer reference to the error of sensor 2
SensErr3 is a pointer reference to the error of sensor 3

Description
The function GetSensorErrors returns the adjusted sensor errors referenced by pointers.

DACDRV::GetPumpErrors

void GetPumpErrors ( double& PumpErr1, double& PumpErr2 )

Parameters
PumpErr1 is a pointer reference to the error of pump 1
PumpErr2 is a pointer reference to the error of pump 2

Description
The function GetPumpErrors returns the adjusted pump errors referenced by pointers.

DACDRV::ResetSimSystem

void ResetSimSystem( void )

Description
The function ResetSimSystem resets the liquid levels of the simulated Three-Tank-System to 0.

DACDRV::SetNoise

void SetNoise( double SignalNoise )

Parameters
SignalNoise is the amplitude of the noise signal in [cm].

Description
The function SetNoise adjusts the amplitude of the noise signal which is superimposed to the liquid level sensor signal of the simulated Three-Tank-System and activates the addition of the noise signal.
**DACDRV::SensNoise**

```c
double SensNoise( void )
```

**Description**

The function `SensNoise` returns the noise signal for the sensors of the liquid levels of the simulated Three-Tank-System. The pseudo binary noise signal is calculated as follows:

\[
\text{noise signal [cm]} = \text{noise} \times (1 - \text{random number}[0...2])
\]

**Return:**

The noise signal (double) in [cm] for the liquid levels.

---

**DACDRV::SenGetLevel**

```c
double SenGetLevel( int mode )
```

**Parameters**

- `mode` determines the return value:
  
  =1, liquid level for the lower calibration point,  
  else, liquid level for the upper calibration point.

**Description**

The function `SenGetLevel` returns the liquid level in [cm] for the lower or upper calibration point of tank 1 depending on the parameter `mode`.

**Return:**

Liquid level (double) of the lower or upper calibration point.

---

**DACDRV::SenSetLevel**

```c
void SenSetLevel( int mode, double val )
```

**Parameters**

- `mode` determines the calibration point:
  
  =1, liquid level for the lower calibration point,  
  else, liquid level for the upper calibration point.

- `val` is the liquid level for the lower or upper calibration point.

**Description**

The function `SenSetLevel` adjusts the liquid level in [cm] for the lower or upper calibration point for all tanks depending on the parameter `mode` to the value of `val`. 
DACDRV::SenEichLevel

```c
void SenEichLevel( int mode )
```

**Parameters**

- `mode` determines the calibration point:
  
  - 1, liquid level for the lower calibration point,
  - else, liquid level for the upper calibration point.

**Description**

The function `SenEichLevel` reads successively in 8 sampling periods (synchronized by `Tick`) the sensor values in [Volt] for the liquid levels (in the calibration points), calculates the corresponding mean values and takes these values for the lower or upper calibration point depending on the parameter `mode`.

The controller interrupt has to be active. Its service routine has to read the sensor values and has to reset `Tick` to 0 (here by the function `TriggerEndstufe`).

DACDRV::SenGetVolt

```c
double SenGetVolt( int mode, double h )
```

**Parameters**

- `mode` determines the return value:
  
  - 1, sensor value in [Volt] for tank 1
  - 2, sensor value in [Volt] for tank 2
  - 3, sensor value in [Volt] for tank 3
  - else return value of 0

- `h` is the liquid level in [cm] of a tank

**Description**

The function `SenGetVolt` returns the sensor value in [Volt] for the sensors of tank 1, 2 or 3 depending on the parameter `mode` for a given liquid level `h` in [cm] by evaluation of the accompanying straight line equation (sensor characteristic).

**Return:**

The sensor value (double) in [Volt] for a liquid level in [cm].

DACDRV::OfcEichStart

```c
void OfcEichStart( void )
```

**Description**

The function `OfcEichStart` adjusts the initial values of the liquid levels for the determination of the outflow coefficients to the current values of the liquid levels in all tanks and stores the counter content of the interrupt service routine to `EichTick`. 
DACDRV::OfcEichStop

void OfcEichStop( void )

Description
The function **OfcEichStop** adjusts the final values of the liquid levels for the determination of the outflow coefficients to the current values of the liquid levels in all tanks and reads the counter content of the interrupt service routine. The time since calling the function **OfcEichStart** is calculated by comparison with *EichTick*. The three outflow coefficients are calculated using this time and the differences of the initial and final liquid levels.

DACDRV::OfcEichGet

double OfcEichGet( int i )

Parameters

i determines the outflow coefficient:

i=0, tank 1-3
i=1, nominal outflow tank2
i=2, tank 3-2

Description
The function **OfcEichGet** returns the outflow coefficient referenced by the parameter *i*.

Return:
The outflow coefficient (double) referenced by *i*.

DACDRV::PfrEichStart

void PfrEichStart( int i )

Parameters

i determines the base point (control signal in [Volt]) for the pump characteristic.

Description
The function **PfrEichStart** adjusts the initial values for the liquid levels used to determine the pump characteristics to the current values of the liquid levels in the tanks 1 and 2. The control signal for both pumps is set to the value of the base point referenced by the parameter *i*. The counter content of the interrupt service routine is stored to *EichTick*. 
DACDRV::PfIEichStop

void PfIEichStop( int i, double time )

Parameters

i determines the base point (pump flow rate in [ml/s]) for the pump characteristic.

time is the delay time for settling of the liquid levels.

Description

The function PfIEichStop resets the control signal for both pumps to 0 and calculates the time since calling the function PfIEichStart by comparing the counter content of the interrupt service routine with EichTick. After passing the delay time time for the settling of the liquid levels in the tanks 1 and 2 these values are taken as the final liquid levels to determine the pump flow rates. The pump flow rates are calculated for the base point referenced by the parameter i. After the determination of the last base point (i=8) the flag indicating pump calibration is reset again.

DACDRV::PfIEichGet

double PfIEichGet( int p, int i )

Parameters

p determines the pump (=1- pump 1, =2- pump 2).
i determines the base point for the pump flow rate (=0,...,8).

Description

The function PfIEichGet returns the pump flow rate for the pump referenced by p and for the base point referenced by the parameter i.

Return:

Pump flow rate (double) referenced by i for pump p.

DACDRV::Save

int Save( char* FileName )

Parameters

*FileName pointer to the name of the file to which the calibration data are to be written.

Description

The function Save stores the sensor characteristics (offset and gradient of the straight line equation), the outflow coefficients as well as the pump characteristics (9 base points for each characteristic) to the file with the given name. The function returns 1 when the file exists, otherwise 0.

Return:

File status:

=1, file exists,
=0, file doesn't exist.
DACDRV::Load

    int Load( char* FileName )

Parameters
    *FileName pointer to the name of the file from which the calibration data are to be read.

Description
    The function Load reads the sensor characteristics (offset and gradient of the straight line equation), the outflow coefficients as well as the pump characteristics (9 base points for each characteristic) from the file with the given name. The function returns 1, when the file exists, otherwise 0.

Return:
    File status:
    =1, file exists,
    =0, file doesn't exist.

DACDRV::eichok

    int eichok( void )

Description
    The function eichok is a dummy routine reserved for future applications.

Return:
    Is always equal to 1

DACDRV::CheckSystem

    int CheckSystem( void )

Description
    The function CheckSystem is a dummy routine reserved for future applications.

Return:
    Is always equal to 1

DACDRV::CheckFree

    int CheckFree( void )

Description
    The function CheckFree is a dummy routine reserved for future applications.

Return:
    Is always equal to 1
DACDRV::StartInterrupt

virtual void StartInterrupt( void )

Description
The function StartInterrupt activates the output stage release by sending the necessary trigger pulse and starting the rectangle signal. The interrupt behaviour is retained.

DACDRV::TriggerEndstufe

virtual void TriggerEndstufe( void )

Description
The function TriggerEndstufe toggles the level of the rectangle signal for the output stage release and resets Tick to 0.
1.6.2 The Class WDAC98

The class WDAC98 realizes the interface between the class DACDRV and the driver functions of the DACSHNTB.DLL, which is a shell for the WDM-driver of the PC adapter card.

The files WDAC98.CPP and WDAC98.H contain the class WDAC98 with the functions:

- `double ReadAnalogVolt(int channel)`
- `void WriteAnalogVolt(int channel, double val)`
- `int ReadDigital(int channel)`
- `void WriteDigital(int channel, int value)`
- `unsigned int GetCounter(void)`
- `unsigned long GetTimer(void)`
- `unsigned int ReadDDM(int channel)`
- `void ResetDDM(int channel)`
- `void ReadAllDDM(unsigned int &cnt0, unsigned int &cnt1, unsigned int &cnt2)`
- `void ResetAllDDM(void)`

### WDAC98::ReadAnalogVolt

```cpp
double ReadAnalogVolt(int channel)
```

**Parameters**

`channel` is the number of the analog input channel, which is to be read.

**Description**

The function `ReadAnalogVolt` reads the analog input channel specified by `channel` and returns the corresponding voltage value. The value is in the range from -10.0 to +10.0 with the assumed unit [Volt].

**Return:**

The input voltage of the analog channel in the range from -10.0 to +10.0.
WDAC98::WriteAnalogVolt

```c
void WriteAnalogVolt( int channel, double val )
```

**Parameters**
- `channel` is the number of the analog output channel, to which a value is to be written.
- `val` is the value for the analog output.

**Description**
The function `WriteAnalogVolt` writes the value `val` in the range from -10.0 to +10.0 (with the assumed unit [Volt]) as an analog voltage to the specified analog output channel. Values outside of the mentioned range are limited internally.

WDAC98::ReadDigital

```c
int ReadDigital( int channel )
```

**Parameters**
- `channel` is the number of the digital input channel, which is to be read.

**Description**
The function `ReadDigital` reads the state (0 or 1) of the specified digital input channel and returns this value.

**Return:**
The state (0 or 1) of the specified digital input.

WDAC98::WriteDigital

```c
void WriteDigital( int channel, int val )
```

**Parameters**
- `channel` is the number of the digital output channel, to which a value is to be written.
- `value` is the new state of the digital output.

**Description**
The function `WriteDigital` writes the value `val` (0 or 1) to the specified digital output channel and with this sets its state.

WDAC98::GetCounter

```c
unsigned int GetCounter( void )
```

**Description**
The function `GetCounter` returns the content of 16-bit-counter register.

**Return:**
The content of the 16-bit-counter register.
WDAC98::GetTimer

```c
unsigned long GetTimer( void )
```

**Description**
The function `GetTimer` returns the content of the 32-bit-timer register.

**Return:**
The content of the 32-bit-timer register.

WDAC98::ReadDDM

```c
unsigned int ReadDDM( int channel )
```

**Parameters**

`channel` is the number of the DDM device, which is to be read.

**Description**
The function `ReadDDM` returns the content of the counter register of the specified DDM device (incremental encoder).

**Return:**
The content of the specified DDM counter register.

WDAC98::ResetDDM

```c
unsigned int ResetDDM( int channel )
```

**Parameters**

`channel` is the number of the DDM device, which is to be reset.

**Description**
The function `ResetDDM` resets the content of the counter register of the specified DDM device (incremental encoder).
WDAC98::ReadAllDDM

void ReadAllDDM( unsigned int &cnt0, unsigned int &cnt1, unsigned int &cnt2 )

Parameters

&cnt0 is a reference to the content of the counter register of the DDM device No. 0, which is to be read.

&cnt1 is a reference to the content of the counter register of the DDM device No. 1, which is to be read.

&cnt2 is a reference to the content of the counter register of the DDM device No. 2, which is to be read.

Description

The function ReadAllDDM should read the contents of the counter registers of the DDM devices 0, 1 and 2 at the same time and return the values by references.

This function is still not realized but reserved for future applications.

WDAC98::ResetAllDDM

void ResetAllDDM( void )

Description

The function ResetAllDDM resets the contents of the counter register of all DDM devices (incremental encoders) at the same time.
# Interface Functions of the PLOTBC32.DLL

List of the functions of the standard interface:

```c
int Version( void ),

HWND CreateSimplePlotWindow(HWND parentHWnd, WORD NumberOfCurves,
                              WORD NumberOfPoints, double far** data )

void ShowPlotWindow(HWND HWnd, BOOL bflag )
void ClosePlotWindow(HWND HWnd)
void UpdatePlotWindow(HWND HWnd)
HWND GetValidPlotHandle( int index )
void AddPlotTitle(HWND HWnd, int Position, LPSTR title)
WORD AddAxisPlotWindow( HWND HWnd, WORD AxisID, LPSTR title,
                         WORD Position, WORD ScalingType,
                         double ScalMin, double ScalDelta, double ScalMax )
void AddXData(HWND HWnd, WORD XCount, double far* XData )
void AddTimeData(HWND HWnd, WORD XCount, double StartTime, double SamplingPeriod )
WORD AddYData(HWND HWnd, WORD nYCount, double far* YData )
void SetAxisPosition(HWND HWnd, WORD AxisID, WORD Position)
int SetCurveMode(HWND HWnd, WORD idCurve, LPSTR title,
                 WORD AxisId, WORD LineStyle, DWORD Colour,
                 WORD MarkType )
int SetPlotMode(HWND HWnd, WORD TitlePosition, DWORD TitleColour, LPSTR Title,
                WORD WithLineStyleTable, WORD WithAxisFrame, WORD WithPlotFrame,
                DWORD FrameColour, WORD WithDate, long OldDate, LPSTR FontName, int MaxCharSize )
void PrintPlotWindow( HWND HWnd, HDC printerDC, int xBegin, int yBegin, int xWidth,
                     int yHeight, BOOL scale )
HWND CreateEmptyPlotWindow(HWND parentHWnd)
```

List of the functions of the extended interface:

```c
BOOL ValidPlotWindow(HWND HWnd)
WORD GetNumberOfCurves(HWND HWnd)
WORD GetidCurvePlotWindow(HWND HWnd, int indexCurve)
WORD SavePlotWindow(HWND HWnd, const char* filename)
WORD RestorePlotWindow(HWND HWnd, const char* filename)
```
Table of the macros in use:

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X_AXIS</td>
<td>1</td>
<td>reference AxisID for the X-axis</td>
</tr>
<tr>
<td>Y_AXIS</td>
<td>2</td>
<td>reference AxisID for the Y-axis</td>
</tr>
<tr>
<td>Y_AXIS</td>
<td>4</td>
<td>reference AxisID for the Y2-axis</td>
</tr>
<tr>
<td>AXE_BOTTOM</td>
<td>1</td>
<td>X-axis bottom to axis frame</td>
</tr>
<tr>
<td>AXE_LEFT</td>
<td>1</td>
<td>Y/Y2-axis left to axis frame</td>
</tr>
<tr>
<td>AXE_RIGHT</td>
<td>2</td>
<td>Y/Y2-axis right to axis frame</td>
</tr>
<tr>
<td>AXE_TOP</td>
<td>2</td>
<td>X-axis top to axis frame</td>
</tr>
<tr>
<td>AXE_MIDDLE</td>
<td>4</td>
<td>X/Y-axis in the middle of the axis frame</td>
</tr>
<tr>
<td>TITLETEXT_TOP</td>
<td>1</td>
<td>drawing title top position</td>
</tr>
<tr>
<td>TITLETEXT_BOTTOM</td>
<td>2</td>
<td>drawing title bottom position</td>
</tr>
<tr>
<td>TITLETEXT_APPEND</td>
<td>4</td>
<td>drawing title appended to the window title</td>
</tr>
<tr>
<td>LINEAR_SCALING</td>
<td>0</td>
<td>linear scaling of the min/max-values of an axis</td>
</tr>
<tr>
<td>LOG_SCALING</td>
<td>1</td>
<td>logarithmic scaling of the min/max-values of an axis</td>
</tr>
<tr>
<td>INTERN_SCALING</td>
<td>0</td>
<td>automatic internal scaling of the min/max-values of an axis</td>
</tr>
<tr>
<td>EXTERN_SCALING</td>
<td>2</td>
<td>external adjustment of the min/max/delta-scaling values of an axis</td>
</tr>
<tr>
<td>NO_MARK</td>
<td>0</td>
<td>without marking a Y-curve</td>
</tr>
<tr>
<td>CROSS</td>
<td>1</td>
<td>marking by a laying cross</td>
</tr>
<tr>
<td>TRIANG_UP</td>
<td>2</td>
<td>marking by a triangle top oriented</td>
</tr>
<tr>
<td>TRIANG_DOWN</td>
<td>3</td>
<td>marking by a triangle bottom oriented</td>
</tr>
<tr>
<td>QUAD</td>
<td>4</td>
<td>marking by a square</td>
</tr>
<tr>
<td>CIRCLE</td>
<td>5</td>
<td>marking by a circle</td>
</tr>
</tbody>
</table>

**Version**

```c
int Version( void )
```

**Description:**

The function `Version` returns the version number (at this time = 150 for the version 1.5 dated 21. November 2002) of this DLL.

**Return**

The version number of this DLL.
CreateSimplePlotWindow

HWND CreateSimplePlotWindow (HWND parentHWnd,
WORD NumberOfCurves, WORD NumberOfPoints, double far** data)

Parameters

parentHWnd is the windows handle of the parent window.
NumberOfCurves is the number of curves in the plot object.
NumberOfPoints is the number of points of each curve in the plot object.
data is a pointer to the value matrix of the curves.

Description

The function CreateSimplePlotWindow creates a window containing a standard plot object. This plot object contains the value matrix data consisting of NumberOfCurves Y-curves (rows of the value matrix) with NumberOfPoints points (columns of the value matrix) for each curve with respect to a common X-axis. The X-axis is interpreted as a time axis with NumberOfPoints steps to be drawn at the top of the axes frame including labels and a standard axis title. All Y-curves correspond to one common Y-axis to be drawn left to the axes frame including a standard axis title and labels determined by an automatic internal scaling. A grid net with dashed lines is added to the axes frame. A linestyle table is located in the upper part of the window containing a short piece of a straight line for each Y-curve with the accompanying attributes linestyle, colour and marking type followed by a short describing text ("Curve #xx"). Each curve is displayed with attributes according to the following table.

<table>
<thead>
<tr>
<th>Curve No.</th>
<th>Text</th>
<th>Linestyle</th>
<th>Colour</th>
<th>Marking Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Curve # 1</td>
<td>PS_SOLID</td>
<td>BLACK</td>
<td>None</td>
</tr>
<tr>
<td>2</td>
<td>Curve # 2</td>
<td>PS_DASH</td>
<td>RED</td>
<td>Cross</td>
</tr>
<tr>
<td>3</td>
<td>Curve # 3</td>
<td>PS_DOT</td>
<td>GREEN</td>
<td>Triangle top</td>
</tr>
<tr>
<td>4</td>
<td>Curve # 4</td>
<td>PS_DASHDOT</td>
<td>BLUE</td>
<td>Triangle bottom</td>
</tr>
<tr>
<td>5</td>
<td>Curve # 5</td>
<td>PS_DASHDOTDOT</td>
<td>MAGENTA</td>
<td>Square</td>
</tr>
<tr>
<td>6</td>
<td>Curve # 6</td>
<td>PS_SOLID</td>
<td>CYAN</td>
<td>Circle</td>
</tr>
<tr>
<td>7</td>
<td>Curve # 7</td>
<td>PS_DASH</td>
<td>YELLOW</td>
<td>None</td>
</tr>
<tr>
<td>8</td>
<td>Curve # 8</td>
<td>PS_DOT</td>
<td>GRAY</td>
<td>Cross</td>
</tr>
</tbody>
</table>

The 5 different linestyles, 6 marking types and 8 colours are repeated serially. The curve handles (identifiers) are set automatically equal to the curve numbers. A standard drawing title will be added below the axes frame.

Return

The Windows handle of the plot object window for a successful windows creation. Otherwise NULL is returned.
ShowPlotWindow

```
void ShowPlotWindow(HWND HWnd, BOOL bflag);
```

**Parameters**

- **HWnd** is a Windows handle of a plot object window.
- **bflag** is a flag to control the visibility of a plot object window (=TRUE - visible, else invisible).

**Description**
The function `ShowPlotWindow` displays a previously created plot object window with the Windows handle `HWnd` when the flag `bflag` is set equal to TRUE. Otherwise the plot object window is hidden.

ClosePlotWindow

```
void ClosePlotWindow(HWND HWnd)
```

**Parameters**

- **HWnd** is a Windows handle of a plot object window.

**Description**
The function `ClosePlotWindow` closes a previously created plot object window with the Windows handle `HWnd` and removes all the corresponding objects from the memory.

UpdatePlotWindow

```
void UpdatePlotWindow(HWND HWnd)
```

**Parameters**

- **HWnd** is a Windows handle of a plot object window.

**Description**
The function `UpdatePlotWindow` updates the drawing of a previously created plot object window with the Windows handle `HWnd`.

GetValidPlotHandle

```
HWND GetValidPlotHandle(int index)
```

**Parameters**

- **index** is an index to reference a plot object window.

**Description**
The function `GetValidPlotHandle` determines the Windows handle `HWnd` of that plot object window which is referenced by the given `index`. Starting with an index of 0 the handle of each previously created plot object window is determinable. The function returns the value 0, when a plot object window with the given `index` does not exist.

**Return**
The handle `HWnd` of the plot object window referenced by `index` if it exists else 0.
AddPlotTitle

```c
void AddPlotTitle( HWND HWND, int Position, LPSTR title)
```

**Parameters**

- `HWND` is a Windows handle of a plot object window.
- `Position` is the position of the drawing title (TITLETEXT_TOP or TITLETEXT_BOTTOM + possibly TITLETEXT_APPEND).
- `title` is a pointer to the new drawing title with a maximum of 255 characters.

**Description:**
The function `AddPlotTitle` inserts a new drawing title `title` at the position `Position` in a previously created plot object window with the Windows handle `HWND`. The position is either the upper part of the drawing frame (TITLETEXT_TOP) or the lower part (TITLETEXT_BOTTOM). If the macro TITLETEXT_APPEND is defined in addition the `title` is appended also to the window’s title. However the overall length of this window’s title is limited to 79 characters. The drawing title must not exceed 255 characters. Line wrapping is carried-out automatically if necessary but the drawing title will be truncated if it exceeds a third of the drawing height.

AddAxisPlotWindow

```c
WORD AddAxisPlotWindow( HWND HWND, WORD AxisID, LPSTR title,
                     WORD Position, WORD ScalingType, double ScalMin, double ScalDelta,
                     double ScalMax )
```

**Parameters**

- `HWND` is a Windows handle of a plot object window.
- `AxisID` is a reference to the axis (X-axis = X_AXIS, Y_axis = Y_AXIS, second Y-axis = Y2_AXIS).
- `title` is a pointer to the new axis title with a maximum of 255 characters.
- `Position` is the position of the axis inside the axes frame:
  - X-axis at the bottom (AXE_BOTTOM), at the top (AXE_TOP) or in the middle (AXE_MIDDLE),
  - a Y-axis left (AXE_LEFT), right (AXE_RIGHT) or in the middle (AXE_MIDDLE) of the frame.
- `ScalingType` is the scaling mode for the new axis:
  - LINEAR_SCALING | INTERN_SCALING - internal, linear
  - LINEAR_SCALING | EXTERN_SCALING - external, linear
  - LOG_SCALING | INTERN_SCALING - internal, logarithmic
  - LOG_SCALING | EXTERN_SCALING - external, logarithmic
- `ScalMin` is the minimum external scaling value for the axis.
- `ScalDelta` is the external scaling step for the axis.
- `ScalMax` is the maximum external scaling value for the axis.
Description

The function \texttt{AddAxisPlotWindow} adds a new axis with the reference \textit{AxisID} (X\_AXIS, Y\_AXIS or Y2\_AXIS) to a previously created plot object window with the Windows handle \texttt{HWnd}. Any existing axis in this plot object with the same reference will be replaced by the new one. The axis title \textit{title}, the position \textit{Position} inside the axes frame (AXE\_BOTTOM / AXE\_LEFT, AXE\_RIGHT / AXE\_TOP or AXE\_MIDDLE) as well as the scaling mode \textit{ScalingType} (LOG\_SCALING / LINEAR\_SCALING and EXTERN\_SCALING / INTERN\_SCALING) are to be defined for the new axis. The scaling values \textit{ScalMin}, \textit{ScalDelta} and \textit{ScalMax} are considered only when the macro EXTERN\_SCALING is defined for the scaling mode. Otherwise the scaling values are determined automatically.

Return

The axis reference \textit{AxisID} when the axis was created successfully, else 0.

\textbf{AddXData:}

\begin{verbatim}
void AddXData(HWND HWnd, WORD XCount, double far* XData)
\end{verbatim}

Parameters:

\texttt{HWnd} is a Windows handle of a plot object window.
\texttt{XCount} is the number of points for the X-axis in the plot object.
\texttt{*Xdata} is a pointer to the data of the X-axis in the plot object.

Description:

The function \textbf{AddXData} adds new data \textit{XData} with a number of \textit{XCount} values for the X-axis to a previously created plot object window with the Windows handle \texttt{HWnd}. Any existing data of a X-axis in this plot object are replaced by the new data.

\textbf{AddTimeData:}

\begin{verbatim}
void AddTimeData(HWND HWnd, WORD XCount, double StartTime, double SamplingPeriod)
\end{verbatim}

Parameters:

\texttt{HWnd} is a Windows handle of a plot object window.
\texttt{XCount} is the number of points (time values) for the X-axis in the plot object.
\texttt{StartTime} is the initial value for the time axis (=X-axis).
\texttt{SamplingPeriod} is the sampling period, the time distance between two successive values for the time axis (=X-axis).

Description:

The function \textbf{AddTimeData} adds new data with a number of \textit{XCount} time values for the X-axis to a previously created plot object window with the Windows handle \texttt{HWnd}. The time values start with \texttt{StartTime} and end with \texttt{(XCount - 1) * SamplingPeriod}. Any existing data of a X-axis in this plot object are replaced by the new data.
AddYData:

WORD AddYData(HWND hWnd, WORD nYCount, double far* YData)

Parameters:

HWnd is a Windows handle of a plot object window.

nYCount is the number of points for the Y-curve in the plot object.

*Ydata is a pointer to the data for the Y-curve in the plot object.

Description:

The function AddYData adds a new Y-curve given by the data YData with a number of YCount values to a previously created plot object window with the Windows handle hWnd. The function returns an automatically generated reference (handle) for the Y-curve when a valid plot object window exists. The standard values for the curve-attributes linestyle, colour, marking type and describing text ("Curve #xx") are set automatically as described with the function CreateSimplePlotWindow with respect to the returned reference value. In case no data are defined for a X-axis, a standard time axis from 1.0 to nYCount*1.0 is generated in addition.

Return

Is equal to the automatically generated reference (idCurve) of the added Y-curve, when the plot object window exists, else equal to 0.

See also

CreateSimplePlotWindow.

SetCurveMode

int SetCurveMode(HWND hWnd, WORD idCurve, LPSTR title, WORD AxisId, WORD LineStyle, DWORD Colour, WORD MarkType)

Parameters

HWnd is a Windows handle of a plot object window.

idCurve is the reference (handle) of the Y-curve.

title is a pointer to the new describing text of the Y-curve used for the linestyle table with a maximum of 255 characters. The current describing text is retained when the length of this string is equal to 0.

AxisId is the assignment to the Y-axis (Y_AXIS) or Y2-axis (Y2_AXIS).

LineStyle is the linestyle of the Y-curve (see CreateSimplePlotWindow). The current linestyle is retained when this parameter is equal to 0xFFFF.

Colour is the (RGB-) colour of the Y-curve. The current colour is retained when this parameter is equal 0xFFFFFFFFL.

MarkType is the marking type of the Y-curve. The current marking type is retained when this parameter is equal 0xFFFF.
The function **SetCurveMode** changes the attributes of a Y-curve referenced by *idCurve* belonging to a previously created plot object window with the Windows handle *HWnd*. The describing text *title* for the linestyle table, the assignment *AxisId* to the Y- or Y2-axis, the linestyle *LineStyle*, the colour *Colour* as well as the marking type *MarkType* are assignable.

**Remark:** When a Y2-axis is not existing but a curve is assigned to this axis the linestyle table demonstrates this fact by displaying only the describing text for this curve without the short piece of a straight line. The number of characters in the describing text should be short with respect to the number of curves.

**Return**

Is equal to 1, when the Y-curve with *idCurve* exists, else equal to 0.

---

**SetPlotMode**

```c
int SetPlotMode( HWND HWnd, WORD TitlePosition, DWORD TitleColour, LPSTR Title,
                 WORD WithLineStyleTable, WORD WithAxisFrame, WORD WithPlotFrame,
                 DWORD FrameColour, WORD WithDate, long OldDate, LPSTR FontName,
                 int MaxCharSize)
```

**Parameters**

- **HWnd** is a Windows handle of a plot object window.
- **TitlePosition** is the position of the drawing title (TITLETEXT_TOP or TITLETEXT_BOTTOM + possibly TITLETEXT_APPEND).
- **TitleColour** is the (RGB-) colour for the drawing title. The current colour is retained when this parameter is equal 0xFFFFFFFFL.
- **Title** is a pointer to the new drawing title with a maximum of 255 characters. The current drawing title text is retained when the length of this string is equal to 0.
- **WithLineStyleTable** enables (=TRUE) or disables (=FALSE) the display mode of the linestyle table.
- **WithAxisFrame** is a flag determining if a frame is to be drawn around the axes crossing (=TRUE) or not (=FALSE).
- **WithPlotFrame** is a flag determining if a frame is to be drawn around the drawing (=TRUE) or not (=FALSE) only during output to a Windows Meta File or a raster device.
- **FrameColour** is the (RGB-) colour for the axes frame. The current colour is retained when this parameter is equal 0xFFFFFFFFL.
- **WithDate** is a parameter determining if no date (=FALSE), the current date (=NEW_DATE) or a given 'old' date (=OLD_DATE) is to be inserted in the upper left part of the drawing.
- **OldDate** is the 'old' date, which is considered only when **WithDate** is set to OLD_DATE.
- **FontName** is the font name of the character set used for all text outputs (titles, date, linestyle table, labels). If the length of this name is equal to 0, the default character set will be used.
MaxCharSize is the maximum character height for all text outputs used with a maximum window size. Reducing the plot window size will scale down the character height to a minimum of 12 pixels. If this parameter is equal to 0xFFFF, the default maximum character size will be used.

Description:
The function SetPlotMode changes the general layout of a plot object window with the Windows handle hWnd previously created i.e. by CreateSimplePlotWindow. As described with the function AddPlotTitle a new drawing title title is inserted at the position Position. The position is either the upper part of the drawing frame (TITLETEXT_TOP) or the lower part (TITLETEXT_BOTTOM). If the macro TITLETEXT_APPEND is defined in addition the title is appended also to the windows title. However the overall length of this windows title is limited to 79 characters. The drawing title must not exceed 255 characters. Line wrapping is carried out automatically if necessary but the drawing title will be truncated if it exceeds a third of the drawing height. The drawing title is displayed using the colour TitleColor and the character set FontName with a maximum character height MaxCharSize (for a maximum size of the plot window). The character set as well as the maximum character height are also used for the other text outputs.

If the flag WithLineStyleTable is set to TRUE a linestyle table is inserted above the axes frame containing a short piece of a straight line for each Y-curve with the accompanying attributes linestyle, colour and marking type followed by a short describing text in the standard form "Curve #xx" or defined by the function SetCurveMode.

When the flag WithAxisFrame is set to TRUE, a frame is drawn around the axes crossing using the colour FrameColour only at those margins, which are not occupied by an axis.

When the flag WithPlotFrame is set to TRUE, an additional frame is drawn around the complete drawing using the colour FrameColour only in case the plot window is output to a Windows Meta File or to a raster device.

The parameter WithDate determines the display mode of the date in the upper left part of the drawing. With With Date set to FALSE the date output is missing. With WithDate set to NEW_DATE the current date (day, month, year and time during drawing the plot) is inserted while WithDate set to OLD_DATE will display the date given by the parameter OldDate.

Return

Is equal to 1, when the plot window with the handle hWnd exists, else equal to 0.

See also

CreateSimplePlotWindow, AddPlotTitle, SetCurveMode.
PrintPlotWindow

```c
void PrintPlotWindow(HWND HWND, HDC printerDC, int xBegin, int yBegin, int xWidth,
                        int yHeight, BOOL scale )
```

**Parameters**

- `HWND` is a Windows handle of a plot object window.
- `printerDC` is the device context of the output device.
- `xBegin` is the left margin of the hardcopy (mm/ Pixel)
- `yBegin` is the upper margin of the hardcopy (mm/ Pixel)
- `xWidth` is the width of the hardcopy (mm/ Pixel)
- `yHeight` is the height of the hardcopy (mm/ Pixel)
- `scale` = TRUE, position and size of the hardcopy in [mm], else position and size of the hardcopy in pixels.

**Description:**

The function `PrintPlotWindow` generates an output (typically a hardcopy) of a previously created plot object window with the Windows handle `HWND`. The output device is defined by its device context handle `printerDC`. The position and the size of the hardcopy are determined by the parameters `xBegin`, `yBegin`, `xWidth` and `yHeight`. These parameters are interpreted as [mm], if the parameter `scale` is set to TRUE. Otherwise these parameters are taken as pixel numbers.

CreateEmptyPlotWindow

```c
HWND CreateEmptyPlotWindow(HWND parentHWND)
```

**Parameters**

- `parentHWND` is the windows handle of the parent window.

**Description:**

The function `CreateEmptyPlotWindow` creates a window with an 'empty' plot object. This plot object only contains the current date, an empty axes frame as well as a standard drawing title above this frame.

**Return**

The Windows handle of the plot object window for a successful windows creation. Otherwise NULL is returned.
ValidPlotWindow

HWND ValidPlotWindow(HWND HWnd)

Parameters

HWnd is a Windows handle of a plot object window.

Description:

The function ValidPlotWindow returns TRUE if the specified Windows handle is a valid handle of an active plot window object. Otherwise it returns FALSE.

Return

TRUE if the Windows handle is a valid handle of a plot object window, else FALSE.

GetNumberOfCurves

WORD GetNumberOfCurves(HWND HWnd)

Parameters

HWnd is a Windows handle of a plot object window.

Description:

The function GetNumberOfCurves returns the number of curves belonging to the currently active plot window object with the specified Windows handle.

Return

The number of curves of the active plot window object.

GetidCurvePlotWindow

WORD GetidCurvePlotWindow(HWND HWnd, int indexCurve)

Parameters

HWnd is a Windows handle of a plot object window.

indexCurve is the number of the curve (>=0, <= number of curves).

Description:

The function GetidCurvePlotWindow returns the reference handle (identification) of a curve with the number indexCurve belonging to the currently active plot window object with the specified Windows handle. The number indexCurve specifies the index of the internal list of curves. It has to be greater than or equal to zero and less than the number of curves of this plot window object.

Return

The number of curves of the active plot window object.
SavePlotWindow

HWND  SavePlotWindow(HWND HWnd)

Parameters

*HWnd* is a Windows handle of a plot object window.

*filename* is a pointer to the name of a file to which the content of a plot object window is to be written.

Description:

The function *SavePlotWindow* returns TRUE if the file with the specified *filename* exists and the content of the active plot window object with the given Windows handle could be written to the file successfully. Otherwise it returns FALSE. The content of the plot window object is defined by the general plot layout, the axis attributes, the curve attributes and its data. These information are written to the file with separating comment lines in an ASCII format.

Return

TRUE if the file exists and the content of the plot object window could be written to it successfully, else FALSE.

---

RestorePlotWindow

HWND  RestorePlotWindow(HWND HWnd)

Parameters

*HWnd* is a Windows handle of a plot object window.

*filename* is a pointer to the name of a file from which the content of a plot object window is to be read.

Description:

The function *RestorePlotWindow* returns TRUE if the file with the specified *filename* exists and the data could be read from this file successfully. Otherwise it returns FALSE. The file data are interpreted as the attributes and data of a plot window object which are stored to the active plot window object with the given Windows handle. The file format must follow the specifications mentioned with *SavePlotWindow*.

Return

TRUE if the file exists and the content of the plot object window could be read from it successfully, else FALSE.
3 Interface Functions of the TIMER32B.DLL

The TIMER32B.DLL (version 2.0) supports the cyclic call of specific functions of a 'SERVICE.DLL' which realizes a sampled data control with a constant sampling period.

The exported functions are:

- `UINT SetService(LPSTR lpServiceName)`
- `UINT SelectDriver(LPSTR lpDriverName, USHORT CardNo)`
- `UINT StartTimer(double Time)`
- `UINT IsTimerActive(void)`
- `UINT StopTimer(void)`
- `void GetMinMaxTime(DWORD *min, DWORD *max, BOOL res)`
- `float GetSimTime(void)`
- `UINT GetPCTimings(WORD wSize, PCTimings *PCtim)`
- `UINT SetupDriver(USHORT *newCardNumber)`

The TIMER32B.DLL operates only with a 32 bit operating system (Win98SE, ME, 2000, XP). In case of a 16 bit operating system the message 'Windows 98/2000 required !!!' appears on the screen and none of the functions can be called in advance. Before any cyclic call can be started by `StartTimer` the functions `SetService` and `SelectDriver` have to be called to define the name of the 'SERVICE.DLL' and the name of the adapter card driver.

---

**SetService**

`UINT SetService(LPSTR lpServiceName)`

**Parameters**

`lpServiceName` is a pointer to the name of a 'SERVICE.DLL'.

**Description**

The function `SetService` copies the given library name (including the extension "DLL") to the global variable `szServiceName` and tries to load the DLL with the given name. In case the DLL cannot be loaded the message 'SetService <<ServiceName>> LoadLibrary failed' is sent to the screen and the function returns 0. In the other case the function tries to get the addresses of the functions `DoService`, `SetDriverHandle` and `IsDemo` from the loaded library. If any of the addresses cannot be obtained a corresponding error message 'SetService - GetProcAddress <<function name>> failed!' appears on the screen and the function returns 0. Otherwise the function returns TRUE (=1). The DLL is freed in any case.

**Attention:** It is strongly recommended to call this function as the first function of the TIMER32B.DLL. Furthermore it has to be called before any function of the "Service"-DLL is called!

**Return**

TRUE (=1) in case the DLL could be loaded and the addresses of some functions could be obtained, otherwise FALSE (=0).
SelectDriver

UINT SelectDriver( LPSTR lpDriverName )

Parameters

lpDriverName is the file name of a new driver for the PC adapter card.

Description

The function SelectDriver copies the given file name to the global variable szDriverName which adjusts the driver for the PC adapter card only when the 'SERVICE.DLL' is not a demo version and no multi-media timer is running. Valid driver names are: DAC98.DRV, DAC6214.DRV, DIC24.DRV, MF614.DRV, MF624.DRV.

In case of invalid driver names an error message 'Missing WDM driver for : <DriverName>' appears on the screen and the function returns TERR_FAIL.

Return

Error state:

TERR_OK (0) on successful operations,
TERR_RUNNING (1), if a timer is still running,
TERR_FAIL (99), invalid driver name.

StartTimer

UINT StartTimer( double Time )

Parameters

Time is the sampling period in seconds (minimum 0.001 s).

Description

The function StartTimer opens and initializes the PC adapter card driver with the name given by the global variable szDriverName (see also SelectDriver). If the driver could be opened and the adapter card could be initialized successfully the driver handle is sent to the 'SERVICE.DLL' by calling SetDriverHandle and a multi-media timer is programmed according to the given sampling period greater equal 1 millisecond. A value of 0 (TERR_OK) is returned only when all of the operations were carried-out successfully. Possible error messages (and returns) are:

'StartTimer - OpenDriver failed.' (TERR_DRV_LOAD_FAIL)
'StartTimer - InitDriver failed.' (TERR_DRV_INIT_FAIL)
'StartTimer - timeSetEvent failed.' (TERR_NO_TIMER)

Return

Error state:

TERR_OK (0) on successful operations,
TERR_RUNNING (1), if a timer is still running,
TERR_TOOFAST (2), if the selected sampling period is too small,
TERR_DRV_LOAD_FAIL (5), if the card driver opening fails,
TERR_DRV_INIT_FAIL (6), if the adapter card could not be initialized,
TERR_NO_TIMER (7), if the multi-media timer could not be started.
IsTimerActive

UINT IsTimerActive( void )

Description
The function IsTimerActive returns the state of the timer controlling the sampling period.

Return
Timer state:
0: timer is not running,
1: timer is running.

StopTimer

UINT StopTimer( void )

Description
The function StopTimer stops the currently running multi-media timer, resets the driver handle and closes the current adapter card driver.

Return
Error state:
TERR_OK (0) on successful operations,
TERR_RUNNING (1), if no timer is running

GetMinMaxTime

GetMinMaxTime( DWORD &min , DWORD &max, BOOL res)

Parameters
&min is a reference to the minimum sampling period in ms.
&max is a reference to the maximum sampling period in ms.
res is a flag to reset the minimum and maximum value of the sampling period.

Description
The function GetMinMaxTime returns the minimum and maximum value of the real sampling period determined up to this time. With res=1 the minimum and maximum value are set to the nominal sampling period. The timing values are taken from the multimedia timer.
GetSimTime

float GetSimTime( void )

Description

The function GetSimTime returns the (simulation) time passed since the last start of a multimedia timer. This value is calculated by the product of the nominal sampling period and the number of calls of the function DoService.

Return

Time in seconds since the last call of StartTimer.

GetPCTimings

UINT GetPCTimings( WORD wSize , PCTimings *PCtim )

Parameters

wSize is the size of the structure of type PCTimings in bytes.

*PCtim is a pointer to the structure of type PCTimings containing:

long n, number of sampling periods since start or last reset

double minCalc, minimum run-time in microseconds of the 'SERVICE.DLL'

double maxCalc, maximum run-time in microseconds of the 'SERVICE.DLL'

double meanCalc, mean value of run-time in microseconds of the 'SERVICE.DLL'

double minSample, minimum sampling period in microseconds

double maxSample, maximum sampling period in microseconds

double meanSample, mean value of sampling period in microseconds

double tmax, maximum sampling period for statistic array in milliseconds

double varCalc, reserved for future use

double varSample, reserved for future use

long vCalc[20], statistic array for run-time of the 'SERVICE.DLL'

long vSample[20], statistic array for sampling period

long status, status of high-resolution timing

Description

The function GetPCTimings returns the minimum, maximum and mean values of the real sampling period and of the run-time of the 'SERVICE.DLL' determined up to this time. The statistic arrays vCalc and vSample contain the frequency of occurrence of the timing values with respect to tmax in addition. That means any timing value greater or equal to tmax results in incrementing vCalc[19] resp. vSample[19] whereas any timing value less or equal to tmax/20 results in incrementing vCalc[1] resp. vSample[1]. The components vCalc[0] resp. vSample[0] should never contain a value greater 0. With PCtim->n=-1 the minimum, maximum, mean values as well as the statistic arrays are reset. Depending on the availability the timing values are either taken from the high-resolution counter (bit 0 of status is set) or from the multi-media timer (bit 0 of status is reset).

Result:

0: no timing values copied due to structure size error,
1: correct timing values copied.
**SetupDriver**

UINT *newCardNumber*

**Parameters**

*newCardNumber* is a pointer to a new card number.

**Description**

The function `SetupDriver` closes the PC adapter card driver and then tries to open and close up to four different drivers for up to four cards of the same type only when no multi-media timer is running.

If no driver could be opened the error message 'It's not possible to configure this driver' appears on the screen and the function returns TERR_FAIL.

In case only one card driver could be opened the message 'Only one card/driver is present and selected !' is sent to the screen and the function returns TERR_OK. Only in case more than one card driver is present a dialog is started to select a new card number which is stored in *newCardNumber*.

An error message 'Could not open Dialog !' appears if the dialog could not be started.

**Return**

Error state :

- TERR_OK (0) on successful operations,
- TERR_RUNNING (1), if a timer is still running,
- TERR_FAIL (99), if no driver could be opened.
4 Interface Functions of the DACSHNTB.DLL

The dynamic-link library DACSHNTB.DLL provides an interface between a 32 bit application and the WDM drivers for the adapter cards (ISA bus: DAC98, DIC24, DAC6214, PCI bus: MF614 and MF624 at this time) from amira’s product range. Handling the different I/O functions of these adapter cards is simplified by using the exported functions of the DACSHNTB.DLL instead of calling CreateFile and DeviceIoControl to operate directly with the WDM drivers (see also short documentation of the WDM drivers).

Any application must follow the series

OpenDACdrv,
InitDACdevice,
call I/O functions and then
CloseDACdrv.

As can be seen from the file DACSHNT.H the exported (extern "C") functions are:

**Standard control functions:**
BOOL _stdcall CloseDACdrv( HANDLE hDAC )

UULONG _stdcall GetDACError( void )

void _stdcall GetDACStatus( HANDLE hDAC, PDEVICESTATUS pdevstatus )

void _stdcall InitDACDevice( HANDLE hDAC )

HANDLE _stdcall OpenDACdrv( USHORT CardType, USHORT CardNumber )

**Digital input/output functions:**
USHORT _stdcall ReadDigital( HANDLE hDAC, USHORT channel )

USHORT _stdcall ReadDigitalMultiple( HANDLE hDAC, USHORT channel, USHORT numbers, USHORT matches, USHORT wait )

USHORT _stdcall ReadDigitalMultMask( HANDLE hDAC, USHORT channelmask, USHORT expvalue, USHORT numbers, USHORT matches, USHORT wait )

USHORT _stdcall ReadAllDigital( HANDLE hDAC )

void _stdcall WriteDigital( HANDLE hDAC, USHORT channel, USHORT value )

void _stdcall WriteDigitalSeries( HANDLE hDAC, USHORT channel, USHORT numbers, PUSHORT values )

void _stdcall WriteAllDigital( HANDLE hDAC, USHORT value )

**Analog input/output functions:**
SHORT _stdcall ReadAnalogInt( HANDLE hDAC, USHORT channel, USHORT mode )

double _stdcall ReadAnalogVolt( HANDLE hDAC, USHORT channel, USHORT mode )

void _stdcall WriteAnalogInt( HANDLE hDAC, USHORT channel, USHORT value )

void _stdcall WriteAnalogVolt( HANDLE hDAC, USHORT channel, double value )

**Counter/timer input/output functions:**
void _stdcall SetCounter( HANDLE hDAC, USHORT value )

void _stdcall SetGateCounter( HANDLE hDAC, USHORT value )
USHORT _stdcall GetCounter( HANDLE hDAC )

void _stdcall SetDACTimer( HANDLE hDAC, ULONG value)

void _stdcall SetDACTimerMs( HANDLE hDAC, double timeMs )

void _stdcall SetGateTimer( HANDLE hDAC, USHORT value )

ULONG _stdcall GetDACTimer( HANDLE hDAC )

Incremental encoder input/output functions:

void _stdcall ResetDDM( HANDLE hDAC, USHORT channel )

void _stdcall ResetAllDDM( HANDLE hDAC )

LONG _stdcall ReadDDM( HANDLE hDAC, USHORT channel )

void _stdcall ReadDDMSet( HANDLE hDAC, USHORT channel, PDIC_DDM_SETL pddm_set )

Special control functions:

void _stdcall SetClock( HANDLE hDAC, USHORT mode )

void _stdcall SetDDMDir( HANDLE hDAC, USHORT channel, USHORT value )

void _stdcall SetDDMFilter( HANDLE hDAC, USHORT channel, USHORT value )

void _stdcall SetINT( HANDLE hDAC, USHORT channel, USHORT mode )

void _stdcall WaitDAC( HANDLE hDAC, USHORT time )
OpenDACdrv

HANDLE _stdcall OpenDACdrv( USHORT CardType, USHORT CardNumber)

Parameters

CardType is the type of the adapter card defined by the macro (see DACSHNT.H) or value of the following table

<table>
<thead>
<tr>
<th>Macro</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAC98</td>
<td>98</td>
</tr>
<tr>
<td>DIC24</td>
<td>24</td>
</tr>
<tr>
<td>DAC6214</td>
<td>6214</td>
</tr>
<tr>
<td>MF614</td>
<td>614</td>
</tr>
<tr>
<td>MF624</td>
<td>624</td>
</tr>
</tbody>
</table>

CardNumber is the number of the card installed in the PC(=1...4 for cards of same type).

Description

The function OpenDACdrv opens the WDM driver for the adapter card with the given type and number and returns the corresponding handle if this driver exists and if it is not opened by another application. An error value is returned in any other case. A CardNumber less than 1 is automatically set to 1 whereas a value greater 4 is set to 4.

Return

Driver handle or INVALID_HANDLE_VALUE

CloseDACdrv

BOOL _stdcall CloseDACdrv(HANDLE hDAC)

Parameters

hDAC is the handle of the WDM driver opened previously for a PC adapter card.

Description

The function CloseDACdrv closes the WDM driver with the given handle and returns a value of non-zero if the function succeeds. Otherwise the return value is zero.

Return

Error state :

non-zero on successful operations,
zero in case of an error.
InitDACDevice

void _stdcall InitDACDevice( HANDLE hDAC )

Parameters

hDAC is the handle of the WDM driver opened previously for a PC adapter card.

Description

The function InitDACDevice initializes the registers of the adapter card and reads the identification mask from the adapter card. Only in case this mask is equal to the expected value the initialization procedure is taken as valid which is followed by resetting the global error flag DACError. Otherwise this error flag is set to 1.

This function is always to be called directly after opening the WDM driver and before any other function to access the adapter card is called!

The DACError is set to 1 by all of the following functions when the handle hDAC is invalid or when the WDM driver could not return a valid result.

The initial settings are:
All digital outputs to zero.
All analog outputs to 0.0V
Internal clock for 16 bit counter and 32 bit timer set to 2MHz (50MHz for MF624).
Gates for 16 bit counter and 32 bit timer disabled…
Initial settings for 16 bit counter and 32 bit timer are 0xFFFF resp. 0xFFFFFFFF…
Reset all DDM’s (incremental encoders) and the internal interrupt source.

Further details see description of WDM drivers

GetDACError

ULONG _stdcall GetDACError( void )

Description

The function GetDACError returns the value of DACError.
GetDACStatus

```c
void _stdcall GetDACStatus(HANDLE hDAC, PDEVICESTATUS pdevstatus )
```

**Parameters**

- `hDAC` is the handle of the WDM driver opened previously for a DAC98 adapter card.
- `pdevstatus` is a pointer to a structure of type DEVICE_STATUS containing:
  - `USHORT size;` // 0: size of this structure
  - `USHORT address;` // 1: base address of the adapter card
  - `USHORT id;` // 2: identification of adapter card
  - `USHORT INT;` // 3: internal interrupt source channel
  - `ULONG clock_value;` // 4, 5: internal clock in Hz (for timer & counter)
  - `USHORT clock_intern;` // 6: reserved
  - `USHORT timer_installed;` // 7: reserved
  - `USHORT gate_counter;` // 8: last setting of gate for counter input
  - `USHORT counter_set;` // 9: last setting of counter register
  - `ULONG timer_set;` // 10, 11: last setting of timer register
  - `USHORT gate_timer;` // 12: last setting of gate for timer input
  - `USHORT ISAinterrupt;` // 13: current ISA interrupt vector
  - `USHORT status1;` // 14: reserved
  - `USHORT status2;` // 15: reserved
  - `UCHAR AINs;` // 16L: number of analog input channels
  - `UCHAR AOUTs;` // 16H: number of analog output channels
  - `UCHAR DINs;` // 17L: number of digital input channels
  - `UCHAR DOUTs;` // 17H: number of digital output channels
  - `UCHAR DDMs;` // 18L: number of incremental encoder (DDM) channels
  - `UCHAR COUNTERs;` // 18H: number of counters, timers
  - `UCHAR hDAC;` // 19L: handle of adapter card driver
  - `UCHAR flag;` // 19H: status flag of adapter card driver

**Description**

The function `GetDACStatus` reads the complete status information of the adapter card and stores the results in the corresponding structure elements.
ReadDigital

BOOL _stdcall ReadDigital(HANDLE hDAC, USHORT channel)

Parameters

hDAC is the handle of the WDM driver opened previously for a PC adapter card.
channel identifies a single input channel (=bit number of the digital inputs).

Description

The function ReadDigital reads a single digital input channel and returns its value.

Return

Value 0 or 1 of the specified digital input channel.

ReadDigitalMultiple

BOOL _stdcall ReadDigitalMultiple(HANDLE hDAC, USHORT channel, USHORT numbers, USHORT matches, USHORT wait)

Parameters

hDAC is the handle of the WDM driver opened previously for a PC adapter card.
channel identifies a single input channel (=bit number of the digital inputs).
numbers specifies the number of the desired read operations.
matches specifies how many read operations must return a true result.
wait is reserved for the future.

Description

The function ReadDigitalMultiple reads a single digital input channel multiple (numbers of) times and counts the true results. It returns a true value when the sum of the results is greater than or equal to the value of matches. Otherwise the return value is zero.

Return

Value true (=1), when numbers of read operations returned at least matches of true results, otherwise 0.

ReadDigitalMultMask

BOOL _stdcall ReadDigitalMultMask(HANDLE hDAC, USHORT channelmask, USHORT expvalue, USHORT numbers, USHORT matches, USHORT wait)

Parameters

hDAC is the handle of the WDM driver opened previously for a PC adapter card.
channelmask identifies the input channels, which are to be read.
expvalue specifies the value, which is expected from the identified digital input channels.
numbers specifies the number of the desired read operations.
matches specifies how many read operations must return a true result.
wait is reserved for the future.

**Description**

The function **ReadDigitalMultMask** reads all of the digital input channels multiple *(numbers of)* times and compares the masked *(by channelmask)* results with the expected value *expvalue*. It returns a true value when at least a number of *matches* comparisons delivered a true result. Otherwise the return value is zero.

**Return**

Value true (1), when *numbers* of read operations returned at least *matches* of true results for the comparisons between the masked channels and the expected value, otherwise 0.

**ReadAllDigital**

```c
USHORT _stdcall ReadAllDigital(HANDLE hDAC)
```

**Parameters**

*hDAC* is the handle of the WDM driver opened previously for a PC adapter card.

**Description**

The function **ReadDigital** reads all digital input channels and returns their value.

**Return**

Value of the digital input channels.

**WriteDigital**

```c
void _stdcall WriteDigital(HANDLE hDAC, USHORT channel, USHORT value)
```

**Parameters**

*hDAC* is the handle of the WDM driver opened previously for a PC adapter card.

*channel* identifies a single output channel (=bit number of the digital outputs).

*value* is the new state (0 or 1) of the specified digital output channel.

**Description**

The function **WriteDigital** sets the state of the specified digital output *channel* to the given *value*.
WriteDigitalSeries

void _stdcall WriteDigitalSeries(HANDLE hDAC, USHORT channel, USHORT numbers, PUSHORT pvalues )

Parameters

hDAC is the handle of the WDM driver opened previously for a PC adapter card.
channel identifies a single or all digital output channels.
numbers specifies the number (size of series not greater than 254) of values for the digital output channel(s).
pvalue is a pointer to an array (series) of values.

Description

The function WriteDigitalSeries writes a series of numbers values referenced by the pointer pvalue to the specified digital output channel (single:0-7 or all:0xFE digital output channels). The timing behaviour depends on the computing power of the PC.

WriteAllDigital

void _stdcall WriteAllDigital(HANDLE hDAC, USHORT value )

Parameters

hDAC is the handle of the WDM driver opened previously for a PC adapter card.
value is the new state of all digital output channels.

Description

The function WriteAllDigital sets the states of all digital output channels to the given value.

ReadAnalogInt

SHORT _stdcall ReadAnalogInt(HANDLE hDAC, USHORT channel,USHORT mode )

Parameters

hDAC is the handle of the WDM driver opened previously for a PC adapter card.
channel identifies a single analog input channel (=number of the analog inputs).
mode specifies the analog input range only for the DAC98 and MF614 adapter card according to:
0: 0 ... 5V…
1: 0 ... 10V…
2: -5 ... +5V
3: -10 ... +10V

Description

The function ReadAnalogInt reads a single analog input channel and returns its value as an 16 bit integer in the range from -2048 to +2047 (-8192 to +8191 for MF624).
Value in the range from -2048 to +2047 (-8192 to +8191 for MF624) from the specified analog input channel.

### ReadAnalogVolt

```
double _stdcall ReadAnalogVolt(HANDLE hDAC, USHORT channel, USHORT mode )
```

**Parameters**

- `hDAC` is the handle of the WDM driver opened previously for a PC adapter card.
- `channel` identifies a single analog input channel (=number of the analog inputs).
- `mode` is reserved for future use.

**Description**

The function `ReadAnalogVolt` reads a single analog input channel and returns its value taken as an input voltage in the range from -10 to +10.

**Return**

Value in the range from -10 to +10 from the specified analog input channel.

### WriteAnalogInt

```
void _stdcall WriteAnalogInt(HANDLE hDAC, USHORT channel, USHORT value )
```

**Parameters**

- `hDAC` is the handle of the WDM driver opened previously for a PC adapter card.
- `channel` identifies a single analog output channel (=number of the analog outputs).
- `value` specifies the value for the analog output channel.

**Description**

The function `WriteAnalogInt` writes the given value to the specified analog output channel. The value has to be in the range from -2048 to +2047 (-8192 to +8191 for MF624) to obtain an output voltage range from -10 to +10V.
Interface Functions of the DACSHNTB.DLL

Three – Tank – System DTS200

WriteAnalogVolt

void _stdcall WriteAnalogVolt(HANDLE hDAC, USHORT channel, double value )

Parameters

hDAC is the handle of the WDM driver opened previously for a PC adapter card.

channel identifies a single analog output channel (=number of the analog outputs).

value specifies the value for the analog output channel.

Description

The function WriteAnalogVolt writes the given value to the specified analog output channel. The value has to be in the range from -10 to +10 to obtain an output voltage range from -10 to +10V. Any other value is limited automatically to the mentioned range.

SetCounter

void _stdcall SetCounter(HANDLE hDAC, USHORT value )

Parameters

hDAC is the handle of the WDM driver opened previously for a PC adapter card.

value specifies the initial value for the 16 bit counter.

Description

The function SetCounter sets the initial value for the 16 bit counter. The counter will decrement this value.

SetGateCounter

void _stdcall SetGateCounter(HANDLE hDAC, USHORT value )

Parameters

hDAC is the handle of the WDM driver opened previously for a PC adapter card.

value enables (=1) or disables (=0) the clock input for the 16 bit counter.

Description

The function SetGateCounter enables the gate for the clock signal input of the 16 bit counter when value is set to 1, this gate is disabled when value is equal to zero.
GetCounter

```c
USHORT _stdcall GetCounter(HANDLE hDAC )
```

**Parameters**

`hDAC` is the handle of the WDM driver opened previously for a PC adapter card.

**Description**

The function `GetCounter` returns the content of the 16 bit counter register ('on-the-fly').

**Return**

Content of the 16 bit counter register.

SetDACTimer

```c
void _stdcall SetDACTimer(HANDLE hDAC, ULONG value )
```

**Parameters**

`hDAC` is the handle of the WDM driver opened previously for a PC adapter card.

`value` specifies the initial value for the 32 bit timer.

**Description**

The function `SetDACTimer` sets the initial value for the 32 bit timer. The upper 16 bit of `value` are written to the upper cascade of the timer register and the lower 16 bit are written to the lower cascade. **Any of the cascade parts must not be set to zero!** The resulting time period is the product of the lower and upper cascade and the time period of the clock signal (default 1/2000000 sec).

SetDACTimerMs

```c
void _stdcall SetDACTimerMs(HANDLE hDAC, double timeMs )
```

**Parameters**

`hDAC` is the handle of the WDM driver opened previously for a PC adapter card.

`timeMs` specifies the timing period in [ms] for the 32 bit timer.

**Description**

The function `SetDACTimerMs` programs the 32 bit timer such that it operates with a timing period of `timeMs` milliseconds. The timing period is valid only when the input clock of the timer is equal to 2MHz.
SetGateTimer

void _stdcall SetGateTimer(HANDLE hDAC, USHORT value)

Parameters

hDAC is the handle of the WDM driver opened previously for a PC adapter card.

value enables (=1) or disables (=0) the clock input for the 32 bit timer.

Description

The function SetGateTimer enables the gate for the clock signal input of the 32 bit timer when value is set to 1, this gate is disabled when value is equal to zero.

GetDACTimer

ULONG _stdcall GetDACTimer(HANDLE hDAC)

Parameters

hDAC is the handle of the WDM driver opened previously for a PC adapter card.

Description

The function GetDACTimer returns the content of the 32 bit timer register ('on-the-fly').

Return

Content of the 32 bit timer register.

ResetDDM

void _stdcall ResetDDM(HANDLE hDAC, USHORT channel)

Parameters

hDAC is the handle of the WDM driver opened previously for a PC adapter card.

channel specifies the number of the DDM device on the adapter card.

Description

The function ResetDDM resets a single DDM device (for incremental encoder signal) specified by the given channel number. Valid channel numbers are 0 for the DAC6214, 0-2 for the DAC98 and 0-3 for the DIC24, MF614 and MF624 adapter card.
ResetAllDDM

void _stdcall ResetAllDDM(HANDLE hDAC)

Parameters
hDAC is the handle of the WDM driver opened previously for a PC adapter card.

Description
The function ResetAllDDM resets all DDM devices (for incremental encoder signals) from the adapter card at the same time.

ReadDDM

LONG _stdcall ReadDDM(HANDLE hDAC, USHORT channel)

Parameters
hDAC is the handle of the WDM driver opened previously for a PC adapter card.
channel specifies the number of the DDM device on the adapter card.

Description
The function ReadDDM reads the content of the counter register of a single DDM device (for incremental encoder signal) specified by the given channel number. Valid channel numbers are 0 for the DAC6214, 0-2 for the DAC98 and 0-3 for the DIC24, MF614 or MF624 adapter card.

Return
Content of the 16 or 24 bit DDM counter register.

ReadDDMSet

void _stdcall ReadDDMSet(HANDLE hDAC, USHORT channel, PDIC_DDM_SETL pddm_set)

Parameters
hDAC is the handle of the WDM driver opened previously for a PC adapter card.
channel specifies the number of the DDM device on the adapter card.
pddm_set is a pointer to a structure of type DIC_DDM_SETL containing

ULONG timer, DDM timer or adapter card timer
LONG counter, DDM counter
USHORT status, DDM status (only for DIC24)

Description
The function ReadDDMSet reads the contents of the timer register, the counter register and the status register of a single DDM device (for incremental encoder signal) specified by the given channel number only for the DIC24 adapter card and stores the results in the corresponding structure elements referenced by pddm_set. In case of DAC6214, DAC98 and MF614 the 32 bit timer register of the adapter card is read instead and a status value of 0 is re-
turned. Valid channel numbers are 0 for the DAC6214, 0-2 for the DAC98 and 0-3 for the DIC24, MF614 and MF624 adapter card.

### SetDDMFilter

```c
void _stdcall SetDDMFilter(HANDLE hDAC, USHORT channel, USHORT value )
```

**Parameters**

- `hDAC` is the handle of the WDM driver opened previously for a DIC24 adapter card.
- `channel` specifies the number of the DDM device on the DIC24 adapter card.
- `value` enables (=0) or disables (=1) the input filter of the DDM device.

**Description**

The function `SetDDMFilter` enables the input filter function of a single DDM device (for incremental encoder signal) specified by the given channel number when value is set to zero. A value of 1 disables the filter function. **This function is valid only for a DIC24 adapter card!**

### SetDDMDir

```c
void _stdcall SetDDMDir(HANDLE hDAC, USHORT channel, USHORT value )
```

**Parameters**

- `hDAC` is the handle of the WDM driver opened previously for a DIC24 adapter card.
- `channel` specifies the number of the DDM device on the DIC24 adapter card.
- `value` specifies the timer count direction (increment=0, decrement=1) of the DDM device.

**Description**

The function `SetDDMDir` adjusts the count direction of the timer of a single DDM device (for incremental encoder signal) specified by the given channel number. The timer is incremented when value is set to zero and decremented when value is set to 1. **This function is valid only for a DIC24 adapter card!**
SetINT

```c
void _stdcall SetINT(HANDLE hDAC, USHORT channel, USHORT value )
```

**Parameters**

- **hDAC** is the handle of the WDM driver opened previously for a PC adapter card.
- **channel** specifies the internal interrupt source.
- **value** enables (=1) or disables (=0) the interrupt source.

**Description**

The function `SetINT` enables the internal interrupt source specified by the given `channel` number when `value` is set to 1. The interrupt source is disabled for a `value` of zero. The selectable interrupt sources depend on the type of the ISA Bus adapter card as follows:

<table>
<thead>
<tr>
<th>channel</th>
<th>DAC98</th>
<th>DIC24</th>
<th>DAC6214</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>32 bit timer overflow</td>
<td>new increment DDM0</td>
<td>32 bit timer overflow</td>
</tr>
<tr>
<td>1</td>
<td>16 bit counter overflow</td>
<td>new increment DDM1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>End of A/D conversion</td>
<td>new increment DDM2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>new increment DDM3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>32 bit timer overflow</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>16 bit counter overflow</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>External interrupt 6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>External interrupt 7</td>
<td></td>
</tr>
</tbody>
</table>

The interrupt itself will be active only when the hardware jumper (see hardware manual) is set according to the interrupt resource selected during installation of the adapter card.

SetClock

```c
void _stdcall SetClock(HANDLE hDAC, USHORT mode )
```

**Parameters**

- **hDAC** is the handle of the WDM driver opened previously for a DAC98 adapter card.
- **mode** specifies the internal clock of the DAC98 adapter card:
  - =0, 8 MHz
  - =1, 4 MHz
  - =2, 2 MHz
  - =3, 1 MHz
  - =4, 500 kHz
  - =5, 250 kHz
  - =6, 125 kHz
  - =7, 62.5 kHz.

**Description**

The function `SetClock` adjusts the internal clock for the 16 bit counter and the 32 bit timer only for the DAC98 adapter card.
WaitDAC

    void _stdcall WaitDAC(HANDLE hDAC, USHORT time)

Parameters

hDAC is the handle of the WDM driver opened previously for a PC adapter card.

time specifies the wait time in [ms].

Description

The function WaitDAC operates as a delay time in milliseconds.
5 WDM Drivers for DAC98, DAC6214, DIC24, MF614 and MF624

The drivers are installable 32-Bit drivers (WDM Version 1.0) applicable to 32-Bit programs with Windows 98SE/ME/2000/XP.

It is strictly recommended to use the functions of DACSHNTB.DLL instead of controlling the drivers directly!

To exchange data with the drivers directly the function DeviceIoControl is to be used with a device handle returned by the function CreateFile, i.e.

```
HANDLE hDev = CreateFile( (PSTR) szDeviceName, GENERIC_WRITE | GENERIC_READ | FILE_SHARE_WRITE | FILE_SHARE_READ, NULL, OPEN_EXISTING, 0, NULL);
```

where szDeviceName is determined by the system during the driver installation procedure. Any of the installed drivers may be opened only by one single application. To control one of the mentioned adapter cards the function DeviceIoControl may be called as follows:

```
DWORD dwRet = DeviceIoControl( hDev, DRVCMD_xxx, ioBuff, IOBUFF_SIZE, ioBuff, IOBUFF_SIZE, &dwBytesRet, NULL);
```

where DRVCMD_xxx defines one of the user control codes (see file DRVIOCTL.H) which will be described in the following. Instead of declaring two different buffers for input and output operations the single buffer ioBuff of size IOBUFF_SIZE may be used. The required size of this buffer depends on the type of the user control code as can be seen from the following tables.

**Table of device or channel macros**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL_CHANNELS</td>
<td>0x00FE</td>
<td>All channels of digital input or output</td>
</tr>
<tr>
<td>DDM0</td>
<td>0</td>
<td>Incremental encoder input channel 0</td>
</tr>
<tr>
<td>DDM1</td>
<td>1</td>
<td>Incremental encoder input channel 1</td>
</tr>
<tr>
<td>DDM2</td>
<td>2</td>
<td>Incremental encoder input channel 2</td>
</tr>
<tr>
<td>DDM3</td>
<td>3</td>
<td>Incremental encoder input channel 3</td>
</tr>
<tr>
<td>COUNTER</td>
<td>10</td>
<td>16 Bit counter device</td>
</tr>
<tr>
<td>TIMER</td>
<td>11</td>
<td>32 Bit timer device</td>
</tr>
<tr>
<td>DDM_SET</td>
<td>24</td>
<td>Read data structure with counter, timer, status from DIC24 DDM</td>
</tr>
<tr>
<td>DDM_FILTER</td>
<td>25</td>
<td>Set input filter of DIC24 DDM</td>
</tr>
<tr>
<td>DDM_DIR</td>
<td>26</td>
<td>Set count direction of DIC24 DDM</td>
</tr>
</tbody>
</table>
### Table of user control codes

<table>
<thead>
<tr>
<th>User control code</th>
<th>Size of ioBuff (WORD) on return</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRVCMD_DREAD</td>
<td>3</td>
<td>Read a single or all digital input channels</td>
</tr>
<tr>
<td>DRVCMD_DREAD_MULTIPLE</td>
<td>5</td>
<td>Read a single input channel multiple times</td>
</tr>
<tr>
<td>DRVCMD_DREAD_MULT_MASK</td>
<td>6</td>
<td>Read masked input channels multiple times</td>
</tr>
<tr>
<td>DRVCMD_DWRITE</td>
<td>2</td>
<td>Write a value to a single or all digital output channels</td>
</tr>
<tr>
<td>DRVCMD_DWRITE_SERIES</td>
<td>Variable</td>
<td>Write a series of values to a single or all digital output channels</td>
</tr>
<tr>
<td>DRVCMD_AREAD</td>
<td>3</td>
<td>Read a single analog input channel</td>
</tr>
<tr>
<td>DRVCMD_AWRITE</td>
<td>2</td>
<td>Write a value to a single analog output channel</td>
</tr>
<tr>
<td>DRVCMD_COUNT</td>
<td>4-7</td>
<td>Read the content of a single DDM counter (from quadrature encoder input), a 16/32 Bit counter/timer or timer and status of a single DDM</td>
</tr>
<tr>
<td>DRVCMD_SCOUNT</td>
<td>2-3</td>
<td>Set filter operation or count direction of a single DDM (DIC24) or initial value for 16/32 Bit counter/timer</td>
</tr>
<tr>
<td>DRVCMD_RCOUNT</td>
<td>1</td>
<td>Reset a single DDM or all DDM’s or a 16/32 Bit counter/timer</td>
</tr>
<tr>
<td>DRVCMD_SGATE</td>
<td>2</td>
<td>Set the gate of a 16/32 Bit counter/timer</td>
</tr>
<tr>
<td>GET_DRIVER_INFO</td>
<td>variable</td>
<td>Get a string containing the symbolic driver name, the version, the card address and the interrupt vector</td>
</tr>
<tr>
<td>DRVCMD_INIT</td>
<td>5</td>
<td>Initialize the adapter card and return its address, ID, interrupt vector and handle</td>
</tr>
<tr>
<td>DRVCMD_SCLOCK</td>
<td>1</td>
<td>Set the clock rate for counter and timer</td>
</tr>
<tr>
<td>DRVCMD_SINT</td>
<td>2</td>
<td>Set internal interrupt source</td>
</tr>
<tr>
<td>DRVCMD_RSTsetStatus</td>
<td>19</td>
<td>Read the status of the adapter card</td>
</tr>
<tr>
<td>DRVCMD_WAIT</td>
<td>1</td>
<td>Start a system timer for a given period of ms</td>
</tr>
<tr>
<td>DRVCMD_TEST_WAIT_END</td>
<td>1</td>
<td>Check if the system timer reached time-out</td>
</tr>
</tbody>
</table>
# Table of user control codes with IObuffer layout

<table>
<thead>
<tr>
<th>IObuffer (WORD) index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Digital IO</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DRVCMD_DREAD</td>
<td>Channel</td>
<td>reserved</td>
<td>Returned digitalIN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DRVCMD_DREAD_MULTIPLE</td>
<td>Channel</td>
<td>Number of reads</td>
<td>Number of matches</td>
<td>reserved</td>
<td>Returned result</td>
<td></td>
</tr>
<tr>
<td>DRVCMD_DREAD_MULT_MASK</td>
<td>Channel</td>
<td>Value mask</td>
<td>Number of reads</td>
<td>Number of matches</td>
<td>reserved</td>
<td>Returned result</td>
</tr>
<tr>
<td>DRVCMD_DWRITE</td>
<td>Channel</td>
<td>DigitalOUT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DRVCMD_DWRITE_SERIES</td>
<td>Channel</td>
<td>Number of values</td>
<td>First digitalOUT</td>
<td>Second digitalOUT</td>
<td>…next digitalOUT</td>
<td>…next digitalOUT</td>
</tr>
<tr>
<td><strong>Analog IO</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DRVCMD_AREAD</td>
<td>Channel</td>
<td>Mode</td>
<td>Returned analogIN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DRVCMD_AWRITE</td>
<td>Channel</td>
<td>analogOUT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Counter, Timer, DDM’s</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DRVCMD_COUNT</td>
<td>DDMi</td>
<td>DDM_SET</td>
<td>Returned DDM counter</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DDMi</td>
<td>DDM_SET</td>
<td>Returned DDM counter</td>
<td>Returned DDM timer + status in index 6</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>COUNTER</td>
<td>reserved</td>
<td>Returned 16bit counter</td>
<td>Returned counter gate</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TIMER</td>
<td>reserved</td>
<td>Returned 32bit timer</td>
<td>Returned timer gate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DRVCMD_SCOUNT</td>
<td>DDMi</td>
<td>DDM_FILTER</td>
<td>value</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DDMi</td>
<td>DDM_DIR</td>
<td>value</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>COUNTER</td>
<td>Counter value</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TIMER</td>
<td>32bit timer value</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DRVCMD_RCOUNT</td>
<td>DDMi</td>
<td>ALL_CHANNELS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>COUNTER</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TIMER</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DRVCMD_SGATE</td>
<td>COUNTER</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TIMER</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table of special user control codes with IObuffer layout

<table>
<thead>
<tr>
<th>IObuffer (WORD) index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_DRIVER_INFO</td>
<td>Returned string with IO ports or memory mapped ports, ISA-Interrupt-Number</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DRVCMD_INIT</td>
<td>Returned base address</td>
<td>Returned CardId</td>
<td>Returned CARD_ID</td>
<td>Returned interrupt vector</td>
<td>Returned handle hDAC</td>
<td></td>
</tr>
<tr>
<td>DRVCMD_SCLOCK</td>
<td>Mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DRVCMD_SINT</td>
<td>channel</td>
<td>value</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DRVCMD_RSTATUS</td>
<td>Base address</td>
<td>CardId</td>
<td>Internal interrupt source</td>
<td>Clock setting</td>
<td>reserved</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>reserved</td>
<td>Counter gate</td>
<td>Counter setting</td>
<td>Timer setting</td>
<td>Timer gate</td>
<td></td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
</tr>
<tr>
<td>Interrupt flag</td>
<td>Internal status</td>
<td>Analog channels</td>
<td>Digital channels</td>
<td>DDM’s, counter</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>18</td>
<td>19</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DRVCMD_WAIT</td>
<td>Handle hDAC</td>
<td>Internal flags</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DRVCMD_TEST_WAIT_END</td>
<td>Returned value</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Where CardId returned with the command DRVCMD_RSTATUS is defined as follows:

<table>
<thead>
<tr>
<th>Adapter card</th>
<th>CardId</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAC98</td>
<td>0x55</td>
</tr>
<tr>
<td>DIC24</td>
<td>0x24</td>
</tr>
<tr>
<td>DAC6214</td>
<td>0x01</td>
</tr>
<tr>
<td>MF614</td>
<td>0x614</td>
</tr>
<tr>
<td>MF624</td>
<td>0x624</td>
</tr>
</tbody>
</table>

and CARD_ID returned with the command DRVCMD_INIT should have the same value.
### Table of valid parameters depending on the type of the adapter card

<table>
<thead>
<tr>
<th></th>
<th>DAC98</th>
<th>DIC24</th>
<th>DAC6214</th>
<th>MF614</th>
<th>MF624</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Digital input channels</strong></td>
<td>0-7</td>
<td>0-7</td>
<td>0-3</td>
<td>0-7</td>
<td>0-7</td>
</tr>
<tr>
<td><strong>Digital output channels</strong></td>
<td>0-7</td>
<td>0-7</td>
<td>0-3</td>
<td>0-7</td>
<td>0-7</td>
</tr>
<tr>
<td><strong>Analog input channels, #bits</strong></td>
<td>0-7, 12bits +/- 2048</td>
<td>none</td>
<td>0-5, 12bits +/- 2048</td>
<td>0-7, 12bits +/- 2048</td>
<td>0-7, 14bits +/- 8192</td>
</tr>
<tr>
<td><strong>Analog output channels, #bits</strong></td>
<td>0-1, 12bits +/- 2048</td>
<td>0-1, 12bits +/- 2048</td>
<td>0-1, 12bits +/- 2048</td>
<td>0-3, 12bits +/- 2048</td>
<td>0-7, 14bits +/- 8192</td>
</tr>
<tr>
<td><strong>DDM channels, #bits</strong></td>
<td>DDM0-DDM2, 16bits</td>
<td>DDM0-DDM3, 16bits</td>
<td>DDM0, 12bits</td>
<td>DDM0-DDM3, 24bits</td>
<td>DDM0-DDM3, 24bits</td>
</tr>
<tr>
<td><strong>DDM timer and status</strong></td>
<td>none</td>
<td>available</td>
<td>none</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td><strong>DDM filter and direction</strong></td>
<td>none</td>
<td>available</td>
<td>none</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td><strong>Clock setting</strong></td>
<td>Clk62kHz - Clk8MHz</td>
<td>none</td>
<td>none</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td><strong>Internal interrupt source</strong></td>
<td>0=timer overflow 1=counter overflow 2=End of conversion</td>
<td>0-3=DDMi counter changed 4=timer overflow 5=counter overflow 6,7=external interrupt</td>
<td>Timer overflow</td>
<td>none</td>
<td>none</td>
</tr>
</tbody>
</table>

### Table of initial settings after using DRVCMD_INIT

<table>
<thead>
<tr>
<th></th>
<th>DAC98</th>
<th>DIC24</th>
<th>DAC6214</th>
<th>MF614</th>
<th>MF624</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Digital outputs</strong></td>
<td>Reset to zero</td>
<td>Reset to zero</td>
<td>Reset to zero</td>
<td>Reset to zero</td>
<td>Reset to zero</td>
</tr>
<tr>
<td><strong>Analog outputs</strong></td>
<td>Reset to 0.0V</td>
<td>Reset to 0.0V</td>
<td>Reset to 0.0V</td>
<td>Reset to 0.0V</td>
<td>Reset to 0.0V</td>
</tr>
<tr>
<td><strong>16 Bit counter, setting</strong></td>
<td>Counter2 of 8253 device, 0xFFFF</td>
<td>Counter2 of 8253 device, 0xFFFF</td>
<td>Counter0 of 8253 device, 0xFFFF</td>
<td>Counter3 of CTS9513 device, 0xFFFF</td>
<td>Counter1 (32 Bit), 0xFFFF</td>
</tr>
<tr>
<td><strong>32 Bit timer, setting</strong></td>
<td>Cascaded counter0+1 of 8253 device, 0xFFFF</td>
<td>Cascaded counter0+1 of 8253 device, 0xFFFF</td>
<td>Cascaded counter1+2 of 8253 device, 0xFFFF</td>
<td>Cascaded counter1+2 of CTS9513 device, 0xFFFF</td>
<td>Counter0, 0xFFFF</td>
</tr>
<tr>
<td><strong>Counter/timer clock input</strong></td>
<td>2 MHz</td>
<td>2 MHz</td>
<td>2 MHz</td>
<td>2 MHz</td>
<td>50 MHz</td>
</tr>
<tr>
<td><strong>Counter/timer gates</strong></td>
<td>disabled</td>
<td>disabled</td>
<td>disabled</td>
<td>disabled</td>
<td>disabled</td>
</tr>
<tr>
<td><strong>Quadrature encoder inputs</strong></td>
<td>Reset to zero</td>
<td>Reset to zero</td>
<td>Reset to zero</td>
<td>Reset to zero</td>
<td>Reset to zero</td>
</tr>
<tr>
<td><strong>Internal interrupt</strong></td>
<td>disabled</td>
<td>disabled</td>
<td>disabled</td>
<td>Not available</td>
<td>Not available</td>
</tr>
</tbody>
</table>
Extension Kit Electrical Control Valve and Adapter Box

for

Laboratory Setup DTS200 Three - Tank - System
1 Mounting Instructions for Electrical Control Valves 1-1

1.1 Control Valve as Connection Valve between the Tanks 1-1
1.2 Control Valve as Drain Valve 1-2
1.3 Electrical Connections 1-3

2 The Adapter Box 2-1

2.1 The Rear Panel 2-1
2.1.1 PC-Connector 1 2-1
2.1.2 PC-Connector 2 2-1
2.2 The Front Panel 2-1
2.2.1 Power Supply 1 2-1
2.2.2 Power Supply 2 (POWER Modul) 2-1
2.2.3 Converter Module 2-1
2.3 Technical Data 2-2
2.3.1 Adapter Box 2-2
2.3.2 Converter Module 2-2
2.4 Pin Reservation of the 37-pol. Sockets 2-2
1 Mounting Instructions for Electrical Control Valves

1.1 Control Valve as Connection Valve between the Tanks

1. Drain off all three tanks.

2. Put an absorbent cloth under the connection valves between the tanks because some remaining water may flow out by dismantling the valves.

3. Loosen the mounting screws of all tanks.

4. Loosen both spigot nuts of the connection valves and move them towards the tanks. The valve can be radially pulled out.

Steps 5 and 6 are only necessary if the delivery of the control valve contains additional fittings. If not, continue with step 7.

5. Now unscrew the remaining fittings from the tanks. Remove the sealing material which may be left in the threaded holes of the tanks.

6. Mount the fittings which were delivered together with the electrical control valve (use a spanner). Don’t forget to insert the fittings into the spigot nuts first. Screw the fittings into threaded holes of the tanks carefully.

7. Mount the control valve now by screwing the spigot nuts. The electrical connectors should be directed to the back of the tank system.

Steps 8-11 are only necessary if there is no fastening for the valves located on the base plate.

8. Now mark the mounting holes through the base of the valves.

9. Dismantle the control valves again.

10. Now drill the threaded holes at the marked locations on the base plate (6 mm metric screw-thread). The holes may not be drilled through the base plate.

11. Mount the control valve now by screwing the spigot nuts. Afterwards fix the valves to the base plate by using the delivered hexagon head screws (6 mm metric screw-thread, 15 mm length) with the corresponding plain washers. Use a spanner with 10 mm size.

12. Finally the tanks are screwed down to the base plate.

Now the system is ready for work. Check the closeness of all screwed connections. In case of possible leakage the corresponding connections have to be tightened.

The electrical connections are displayed on the control valves.
1.2 Control Valve as Drain Valve

1. Drain off all three tanks.

2. Put an absorbent cloth under the drain valve because some remaining water may flow out by dismantling the valve.

3. Loosen the spigot nut of the drain valve which has to be replaced and move the spigot nut towards the tank. The valve can be radially pulled out.

Steps 4 and 5 are only necessary if the delivery of the control valve contains additional fittings. If not, continue with step 6.

4. Now unscrew the remaining fitting from the tank. Remove the sealing material which may be left in the threaded holes of the tank.

5. Mount the fitting which was delivered together with the electrical control valve (use a spanner). Don’t forget to insert the fitting into the spigot nut first. Screw the fitting into threaded hole of the tank carefully.

6. Mount the control valve now by screwing the spigot nut. The electrical connectors should be directed to the right side of the tank system.

Steps 7-10 are only necessary if there is no fastening for the valves located on the base plate.

7. Now mark the mounting holes through the base of the valve.

8. Dismantle the control valve again.

9. Now drill the threaded holes at the marked locations on the base plate (6 mm metric screw-thread). The holes may not be drilled through the base plate.

10. Mount the control valve now by screwing the spigot nuts. Afterwards fix the valve to the base plate by using the delivered hexagon head screws (6 mm metric screw-thread, 15 mm length) with the corresponding plain washers. Use a spanner with 10 mm size.

Now the system is ready for work. Check the closeness of all screwed connections. In case of possible leakage the corresponding connections have to be tightened.

The electrical connections are displayed on the control valves.
1.3 Electrical Connections

Depending on the type of the control valve there are two or three electrical connectors mounted on one side of the valve’s actuator.

**ST1** is the electrical connector of the valve motor. The pins 2 and 3 have to be supplied with a voltage of 24VDC/1A. The upper pin is PE (protection earth). The external switch SW1 (pole reversal circuit) should demonstrate how to open and to close the valve.

**ST2 (optional)** is the electrical connector of the internal potentiometer (5 kOhm). In the closed position the pins 2 and 3 are of high impedance. In the open position the pin 2 and 3 are of low impedance. Depending on the adjustment of the potentiometer inside the actuator the limit values are not exactly 5 kOhm (close) and 0 Ohm (open). This doesn’t matter because in normal operation the potentiometer range will be calibrated either by external hardware or software.

**ST3** is the electrical output of the internal limit switches.

- Pin 1: Common
- Pin 2: indicates that the valve is closed
- Pin 3: indicates that the valve is open

The diagram shows an example circuit where the limits are indicated by means of two lamps L1 and L2.
2 The Adapter Box

The Adapter Box serves as an interface between electrical control valves (with or without potentiometer output) and the PC adapter card i.e. MF624 from Humusoft.

2.1 The Rear Panel

The mains input unit is located on the right above the type plate. It contains a fuse holder, the mains inlet and the power switch. Two 37-pol. sockets labeled "PC-Connector 1" and "PC-Connector 2" are located on the rear panel in addition.

2.1.1 PC-Connector 1

This socket is connected to the module slots with the numbers 1 to 4 of the adapter box.

2.1.2 PC-Connector 2

This socket is connected to the module slots with the numbers 5 to 6 of the adapter box.

2.2 The Front Panel

The modules of the adapter box are described from left to right as follows.

2.2.1 Power Supply 1

This module provides the 24 V AC power supply for the six module slots.

2.2.2 Power Supply 2 (POWER Modul)

This module provides the DC power supplies for the six module slots. Three LEDs indicate the availability of the DC voltages.

- +15V(green): +15V power supply is available
- -15V(green): -15V power supply is available

2.2.3 Converter Module

Up to six converter modules may be mounted into the adapter box. Any unused module slot will be covered by a blind plate.

A converter module allows for operating an electrical control valve. It is connected to the control valve by means of three connectors. 5 LEDs indicate different states of the valve. A key allows for selecting either the external operating mode of the valve or the direct operating mode (open/close).

- LED "Power"(green): 24V DC power supply is available.
- LED "Valve Open"(red): Indicates a completely opened valve and that its drive is switched off.
- LED "Open"(green): Indicates that the valve is moving to the open position. This operation mode is achieved either by an external signal connected to 37-pol. socket or by pressing the key to its upper position.
- Key "Extern Control": If this key is in its middle position the valve may be controlled by an external signal connected to 37-pol. socket. The required signals are described further below. When the key is pressed upwards the valve is moving to the open position. Pressing the key downwards will move the valve to the close position accordingly. Operating the key as described above overwrites the function of external signals.
- LED "Close"(green): Indicates that the valve is moving to the close position. This operation mode is achieved either by an external signal connected to 37-pol. socket or by pressing the key to its lower position.
- LED "Valve Closed"(red): Indicates a completely closed valve and that its drive is switched off.
2.3 Technical Data

2.3.1 Adapter Box

Sizes and weight:
- Length: 471 mm
- Depth: 340 mm
- Height: 152 mm
- Weight: 6 kg

Mains supply:
- Input voltage: 230 V AC
- Frequency: 50/60 Hz
- Fuse: 1A M

The leads coming from the front panel of the converter module are directly to be plugged to the control valve. The connectors are to be fixed by screws.

2.3.2 Converter Module

Inputs:
- Supply voltage: 24V AC, 1A
- +15V, -15V, +5V
- Open valve: TTL signal
- Close valve: TTL signal

All inputs are active high (5V).

Outputs:
- Limit switch valve opened: TTL signal
- Limit switch valve closed: TTL signal
- Analog signal valve position: Range +/- 10 V (from potentiometer)

The logic states of the signals to open or close a valve are described in table 2.1. The pin reservation of the 37-pol. connector is described in table 2.2.

<table>
<thead>
<tr>
<th>Open control valve logic</th>
<th>Close control valve logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 2.1: Logic of the input signals

<table>
<thead>
<tr>
<th>PC-Connector 1</th>
<th>PC-Connector 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>module 1</td>
<td>module 2</td>
</tr>
<tr>
<td>Input open control valve</td>
<td>30</td>
</tr>
<tr>
<td>Input close control valve</td>
<td>31</td>
</tr>
<tr>
<td>Output limit switch &quot;Opened&quot;</td>
<td>12</td>
</tr>
<tr>
<td>Output limit switch &quot;Closed&quot;</td>
<td>13</td>
</tr>
<tr>
<td>Digital Ground</td>
<td>29</td>
</tr>
<tr>
<td>Measured valve position (potentiometer)</td>
<td>1</td>
</tr>
<tr>
<td>Analog ground for measured position</td>
<td>9</td>
</tr>
</tbody>
</table>

Table 2.2: Pin reservation of the 37-pol. sockets