ColSim - The Manual

Dr. Christiane Kettner, University of Karlsruhe, FBTA.

31st October 2001

Contents

1 Introduction **3**
1.1 Preface . . . 3
1.2 Characteristics of ColSim . . . 3

2 Installation Guide **4**
2.1 General remarks . . . 4
2.2 Linux/Unix . . . 4

3 Starting the simulation **6**
3.1 Menu based simulation . . . 6
3.2 Manual simulation . . . 10

4 Designing a new system **12**
4.1 Introduction . . . 12
4.2 Editing parameters . . . 12
4.3 Construction of a new system . . . 15
4.4 Runtime errors . . . 22

5 The ColSim types **24**
5.1 General remarks . . . 24
5.2 Fluid types . . . 25
5.3 Ventilation types . . . 36
5.4 Building types . . . 41
5.5 General types . . . 41
5.6 Type related kernel routines . . . 49

6 Structure of ColSim **52**
6.1 Installation . . . 52
6.2 System configuration . . . 53
6.3 Conversion of the simulation script . . . 55
6.4 The simulation run . . . 56

7 Programmer's section **62**
7.1 How to integrate new types into the system . . . 62
7.2 How to write new types . . . 62

Chapter 1

Introduction

1.1 Preface

How ColSim came into being, who helped, who sponsored, etc.

1.2 Characteristics of ColSim

What ColSim can do, what it does better than competing programs.

Modular structure, plug flow modeling, small time steps. Forward integration of differential equation. Simulation environment for developping and testing controlling algorithms

Chapter 2

Installation Guide

2.1 General remarks

In principle, ColSim is designed to run under Linux or Unix. It can be used under Windows NT with some restrictions concerning the graphical in-and output. However, at the current date, this installation chapter deals only with the installation into a Linux system. With SuSE Linux, it is most easy to identiy the needed auxiliary program packages, but there should be no serious problems with other Linux versions.

2.2 Linux/Unix

A compressed ColSim program package can be obtained from ? The file, e.g. colsim.11.tgz, depending on the version number, should be placed in the home directory. Then, it's advantageous to check that the following programs and packages are available in the computer system:

- gawk: Program used mainly for quick data evaluation, i.e. summing up of data files, etc.
- gmake: For compiling and linking source code files and producing an executable code.
- gcc: Gnu C Compiler.
- SuSE package tcl_new (the version based upon tclsh8.0 or newer version) or corresponding package from other Linux version.
- SuSE package tk_new (the version based upon wish8.0 or newer version) or corresponding package. This and the above package are used e.g. for the ColSim menu.

- xfig: Interactive drawing tool which is used to create a graphical representation of the hydraulic system serving as input for the simulation. However, in ColSim versions later than 0.57, there is a customized xfig version included which simplifies the editing of parameters.
- vim or vi: Simple text editor which can be invoked by the ColSim menu panel. Optional.
- gnuplot: graphic program which is used for monitoring the simulation progress.
- tar: Program to unpack the ColSim program package.
- gzip: Program to unzip the ColSim program package.

If all programs listed above are present, type `tar xfz colsim.tgz` in some window. Hereby, the subdirectory `ColSim` is created.In the following, this directory will be called the "ColSim directory" and will be referenced as ./. Subdirectories of the ColSim directory like e.g. `cnv` are referenced as `./cnv/`.

To finish the installation, change into the ColSim directory with `cd ColSim` and type INSTALL.This shell script only fails if the path to one's own home directory is not set and direct execution of programs present in the home directory by typing their name only does not work, rather `./name` must be typed. If this is the case, add the path to your PATH variable or ask the system administrator to do it.

Chapter 3

Starting the simulation

3.1 Menu based simulation

3.1.1 Start the simulation

Figure 3.1:

In order to check whether the installation was successful, we try a first simulation run with a very simple system. Type `ColSim` in the directory `ColSim`. In the following, this will be our reference directory, called the " ColSim directory", and all paths given are assumed to start from here.
Now all availabe ColSim systems are listed in a small grey window titled `project_ organizer`. The steps explained in the following apply to each system, of course, but now we choose a concrete example: The demonstration system "Demo" . With a double click on "Demo", the system is selected. Note that this procedure can be abbreviated by typing `ColSim Demo` directly.
The first thing which is done now is the configuration of the chosen system. This means, all the modules needed are collected, translated, linked together and an exectuable program called `sim` is created. See section 6.2 for details. Then, the ColSim menu[1] pops up, like shown in fig.3.1). To select an item click on it once with

[1]If the menu doesn't appear, there are probably problems with the program `wish`. Make sure that the packages tcl_new and tk_new are installed and try to locate `wish` in your system. Maybe you have to correct the path to wish in the shell script `ColSim`. If nothing helps, go on with the manual simulation described in section 3.2.

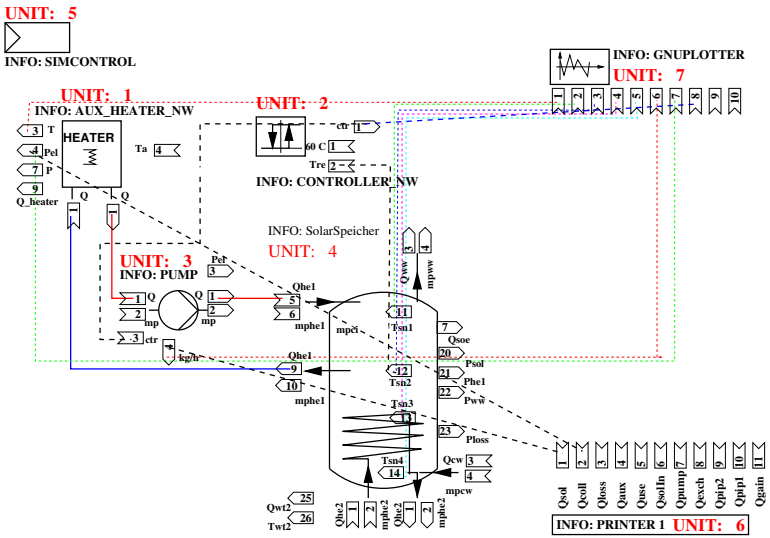


Figure 3.2: Demo system

the left mouse button.
The next step is to select **fig2dek,** which starts the reading in of information about the system's modules, their interconnections and parameters. When this process is finished, the simulation can be started by choosing `sim`.

3.1.2 Step by step

You may want to know what is happening now. Here, we give a short overview, a more detailed explanation can be found in chapter 6.

Conversion of the xfig-object

Activating **fig2dek** started the conversion of the xfig-graphics object `Demo.fig` into a so-called simulation dek `Demo.dek`,containing all the information for the simulation program in more compact form. In ColSim, the input for a simulation run -the arrangement of pipes, storages, pumps, etc., and all their parameters - is defined by a graphical object drawn with the public domain graphics software `xfig`. Take a look at the xfig-picture of the example system `Demo` by selecting **xfig** in the menu. What you see is something like fig.˜3.2.

Though it may appear confusing at first sight, it's in fact easy to survey after a while due to its modular structure. Our demo system consists of the following components, called units in ColSim:

1. Hydraulic components:
   A storage, a heater (called auxiliary heater since a solar collector is considered as the main heat source) and a pump, forming a hydraulic cycle in which fluid is circulating. Since in ColSim the system energy is balanced accurately at every timestep, only closed hydraulic cycles can be calculated.
   The hydraulic ports of the units in such a cycle are interconnected by solid blue and red lines indicating hot and cold water flow.
   Note that the color and the shape of the lines are only chosen for better survey! The only thing really necessary for a correct connection between two units is that one endpoint of the line lies within the outlet box of one unit and the other endpoint within the inlet box of the other unit. In-and outlet boxes can be distinguished by their inwardly and outwardly facing arrow tips, respectively.
   If during the simulation run the pump is turned on, the heater heats up the incoming water to 50 C. The warm water is fed into the storage at its top, while the water to be warmed up leaves the storage at a position specified to be at 70 % of the storage's height in this example.

2. The controlling components:
   A controller unit and the unit simcontrol. The latter must be present in every system to be simulated though it is not connected to any other unit: It contains information about the start and end date of the simulation and the lenght of one time step.
   The controller unit is connected with dashed black lines to the pump and to one or two temperature sensors in the tank. It compares the storage's temperature at the top with a set temperature and sends an "on" signal to the pump if reheating is necessary.

3. The output components:
   A so called gnuplotter which serves for the observation of variables during the simulation run: All quantities connected to inlets of the gnuplotter device are plotted as a function of time with the help of the public domain program `gnuplot`.
   There also is a unit called `printer` which writes all data arriving at its inlets to the file `sim_out0.dat` in the ColSim directory.

More about the individual types in ColSim can be found in chapter 5. Generally, the modules like storages, pipes, controllers, etc. belonging to a ColSim system are called "units". They can appear more than once in the system and are assigned a temporary unit number at the beginning of every simulation run to distinguish them. However, the generic objects like e.g. the `storage` are unique. They are called "types" and are associated with a source code file mostly bearing the same name and describing the physical behaviour of the type. Each type has a specific type number given as well in its source code file as in its `xfig`-representation.

How the unit's parameters are hidden in the graphics file and how new systems can be designed by editing parameters and rearranging the units will be discussed in chapter 4.

To summarize, the `xfig`-representation of the system contains all information about which types are used in the system, how they are interconnected and how their parameters are set. The direct input for the simulation program, however, is the simulation "dek" containing the same information as the graphics file, but in more comprised form.

Choosing **vim sim.dek** in the ColSim menu opens a simple editor[2] showing the simulation dek of the system under consideration. For each unit of the system, all parameters are listed, and also the interconnections with other units: The block which is titled "INPUTS" lists for every input the number of the unit and the outlet it is connected to. If an input is not connected, both entries are zero.

The file `./sim.dek` is actually a link to the file `./projects/Demo/Demo.dek`, but a copy can be found in `./cnv/sim_new.dek`.

Simulation run

Selecting **sim** in the ColSim menu starts the simulation run. First, some information is put out about the units contained in this system and the order in which they will be called.[3] The day of the year currently being simulated is printed and two gnuplot windows are opened. Hit the button **sleep** in order to halt the program execution at the end of the current day.
In the upper window, several system temperatures are plotted as a function of the simulation time in hours:

- The red curve corresponds to the temperature at the top of the storage. Here, the hot water from the heater enters. Obviously, the temperature approaches 50 C when the pump is turned on.
- The green curve is the temperature at 70 % of the storage's height. Here, the temperature sensor for the controller is placed. The pump is turned on if this temperature falls below 42 C.
- The blue curve is the temperature at 30 % of the storage's height. It is not influenced by the heating intervals, but it rises steadily due to heat conduction.
- The pink curve displays the temperature at the storage's bottom. It also rises only due to heat exchange with the upper layers via heat conduction.

In the upper right corner of the upper window, information about the curves is given which is read off the xfig-representation of the storage.
Choose the button **xfig** and see how the plotted curves arise from the connections of the gnuplotter's inlet boxes with the units outlet boxes. For the gnuplotter's connections, colored thin dashed lines are chosen which match the colors in which the connected quantities will be plotted latter.

[2] A few tips for the editor vim: By typing `i` the insert mode is started, with the `Esc` key, it is left again. Only outside of the insert mode, the file can be saved with `:w` and the editor left with `:q`.

[3] To study this output at the beginning it is recommendable to start the simulation manually by typing `sim | more` in the ColSim directory.

In the second gnuplot window, the red curve corresponds to the control signal sent from the controller unit to the pump and takes the values of 1 or 0, depending on whether the pump is on or off.
The green line visualizes the power output of the heater. Note that this curve is multiplied with a scaling factor given in the legend of the plot. The blue curve shows the mass flow through the pump in kg/h, also with a scaling factor.

When the button **wake up** is hit, the program execution continues. When it is finished, an output data file can be found in the ColSim directory. Type `less sim_out0.dat` to scroll in the file. The first column contains the simulation time in hours, the second the mass flow through the pump in kg/h, the third the heater power in W, the fourth the pump power in W, the fifth the power arriving in the storage, the sixth the storage's losses to the ambient in W, the seventh the temperature at sensor 2 in C. All data are averaged over one minute. The printer output can also be changed to integral values or current values, see section 5.5.6.

3.2 Manual simulation

ColSim can also be used without the menu, i.e. without the shell script `ColSim`. Note, however, that this script also configures an environment for the special system given as an argument in the call `ColSim SystemName`, which means that special links are created and paths are set. Therefore, in the following, it is assumed that both the installation script as well as the ColSim script were executed once without problems. If this is not the case, you should take a look at the scripts e.g. with the help of an editor and try to execute one line after the other until the problem is detected.

If the scripts were executed once before, only a few commands are needed to configure the system.

3.2.1 Manual configuration

In the following, all necessary steps of the configuration process are listed for the example system `Demo` (which can be replaced by any other system), but for a detailed explanation please see section 6.2 about the structure of ColSim. These steps need to be executed if another system was simulated before.

- Change into the directory `./cnv` and type `rm running_config.cfg` (this is not really necessary, but sometimes advantageous).
- Type `config ../projects/Demo/Demo.cfg`.
- Change into the directory `./src`, (type `touch *.c` if you want to be one the safe side) and then `make` or `m` which creates the executable program `sim` in the ColSim directory.
- Now change into the directory `./cnv`, type `rm sim.fig` and create a link with `ln -s ../projects/Demo/Demo.fig sim.fig`).

- Change back into the ColSim directory, type `rm load.dat` and set the following link: `ln -s projects/Demo/load.Demo.dat load.dat`.

3.2.2 Manual start of simulation

To start the simulation of the example system `Demo`:

- Change into the directory `./cnv` and check that `sim.fig` is a link to `../projects/Demo/Demo.fig`. If this is not the case, set the link like explained in the paragraph above.
- Type `cnv.exe`, which is the pendant to the button **fig2dek** of the menu. When the conversion of `sim.fig` into `sim_new.dek` is finished, change back into the ColSim directory.
- If a link is set from `sim.dek` to a file in the `./projects` directory, it must be removed (`rm sim.dek`) and reset with the following command: `ln -s cnv/sim_new.dek sim.dek`.
- With the command `sim`, the simulation can be started now. For an explanation of the system and the simulation output, please see section 3.1.2.

The xfig-file can be viewed by typing `xfig &` and loading the xfig file `Demo.fig` in the folder named above. Note, however, that the unit numbers displayed in this xfig file don't necessarily correspond to those used during the simulation run, since the actual unit numbers are always set by `cnv.exe`. The file `./cnv/sim_new.fig`, which is created by `cnv.exe` is a copy of the original system, but with actual unit numbers.

In order to halt the simulation, type `Ctrl s` in the window where the process sim was started. With `Ctrl q`, the execution continues. `Ctrl C` terminates the program run.

The executable program `sim` can also be started in the background with `nohup sim_copy &`, where `sim_copy` is a copy of `sim`, since then another simulation can run simultaneously. However, the two processes shouldn't write to the same output files, i.e. not the same printers (there are printer 0..9) should be used in the systems.

Chapter 4

Designing a new system

4.1 Introduction

The xfig graphics of a system serves as the input file for the simulation. Herein, all information about the implemented units, their interconnections, their parameters and their output variables is contained. With the help of an example system, the necessary xfig commands to design a new ColSim system are explained in the following[1].

In the first subsection , the most easy modification of a system is demonstrated, namely to modify a unit's parameters. In the second subsection , we demonstrate how to construct a new ColSim system from scratch, i.e. putting together the units, defining the right interconnections, editing the load profile, etc.

In order to exercise these steps immediately, a new system will be created in order not to modify the reference systems. The following commands should be executed now:

- Change into the folder `./projects` and copy the demonstration system `Demo` onto the new system `Example` with the command: `cp -r Demo Example`.
- Change into the new subdirectory `Example` and rename the following files: `mv Demo.cfg Example.cfg`, `mv Demo.fig Example.fig`, `mv load.Demo.dat load.Example.dat`.

4.2 Editing parameters

4.2.1 With the ColSim Menu

1. Type `ColSim Example` in the ColSim directory, then choose **xfig** in the ColSim menu (if nothing else is specified, always the left mouse button is meant). By this, an xfig window is opened, showing the actual system `Example`.
2. The most comfortable way to edit the parameters is with the ColSim-customized xfig program which is recognized by a button which reads **Update ColSim** in the left menu panel of xfig. Choosing this button makes little boxes appear everywhere in the picture, marking the corner points of the individual graphical objects. To grep the storage, click on one of the two lowest little boxes in the middle of the canvas with the RIGHT mouse button. The storage vanishes and the `info_edit` window appears directly.
   If you dont't have the ColSim-customized version but only the normal xfig:
   (a) Choose the **Copy** button in the left panel of the xfig window, the with the right mouse button onto one of the storaget's corner points. If you were successful, the text line right under the top panel confirms that the object was copied to a scrapfile called `.xfig` in the home directory.
   (b) select **info_edit** in the ColSim menu which makes the `info_edit` window appear.
3. The `info_edit` window, shown in Fig.4.1 with a pumpt's parameters, displays the part of the information about a unit which is contained but not visible in the xfig graphics of the system:
   - At the top of the `info_edit` window, the unit number and the type number of the object copied to the buffer are displayed. The unit number serves to identify the unit in the system and helps to divide several units of the same type, whereas the type number is needed to find the right source code file describing the unit, e.g. `pipe.c` for the pipe. The type number appears in the upper comment part of the source code files, see section 6.2 for details.
   - Then, the parameter values of the unit are listed, together with their physical units and a short explanation. A more detailed explanation can be found in chapter 5. Here, the parameters of the unit can be edited.
   - By choosing **entries** in the top panel of the `info_edit` window, the input inital values can be shown. In section 3.1, it was explained that via the lines in the xfig graphics of a system, connections are established between the outlet of one unit and the inlet of another unit. However, if an inlet is not connected and consequently receives no values from another unit, the input inital value given here is used instead. Also at the beginning of the simulation, the inital values are used for all inlets where no actual input values are accessible because the connected unit wasn't called yet.
   - In the same way, the output variables can be listed. Here, only the comments can be modified. Note that it is advantageous to insert comments, since they are printed in the gnuplot legends.
   - Finally, the inital elements of the array `deriv_o` can be listed.This array contains the temperatures at each node of the unit, if it is divided into nodes for the calculation like e.g. the storage (see section 6.4.2 for details). Here, the inital temperature at each node can be adjusted. Note, that sometimes these values are corrected by the program if the values dont't seem reasonable (e.g. if they are 0).

[1] For a general introduction into the Public Domain program xfig, please use the help function of this program.

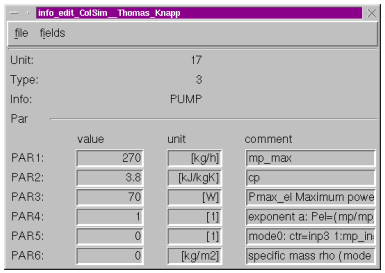


Figure 4.1: The `info_edit` window, showing the pump's parameters.

4. Edit the values and leave the `info_edit` window with **Save and Exit**.

5. If using the customized xfig, the unit is put back in place in the xfig graphics automatically. With the normal xfig, you have to first delete the old unit with the **Delete** button (left mouse button), then insert the edited unit with the **Paste** function (in the `Edit` menu). Using the right mouse button for the **Paste** function makes the unit find its old place on its own, with the left button you have to steer it. Make sure that the inlet and outlet connections to the other units fit, otherwise use the **Move** function to adjust the position.

6. When finished editing parameters,choose **Save** in the `File` menu of xfig. Try the new configuration by activating **fig2dek** in the ColSim menu and **sim** afterwards. If error messages appear, probably the unit wasn't placed right and some lines don't reach their in-or outlet boxes anymore. Try the **Move** function again, or use the error message explanation given in section 4.3.4.

A faster way to edit the parameters of a unit without the customized ColSim xfig is to use the **Delete** function with the right mouse button. function in step 2 a), instead of the **Copy** function. However, if the xfig grafic is somewhat more crowded than our example here, make sure that you dont't accidentally delete the lines instead of the unit. If succesful, the unit is deleted from the grafic, but stored in `.xfig` again. After editing it with `info_edit`, it can be pasted back directly into the grafic.

A useful feature is the **Sort .xfig** function of the ColSim menu. It can be used to acualize the comments to parameters, in-and outlets which appear in the info_edit window by rereading them from the source code file belonging to the unit under consideration. To use it, the unit must be in the scrap file `.xfig` (use the Copy or Delete function of xfig with the right mouse button). Then select **Sort .xfig** and open the `info_edit` window to check for changes. Sometimes the number and meaning of parameters change in the course of improving ColSim.

4.2.2 Without the ColSim Menu

1. Start `xfig` in the ColSim directory by typing `xfig &`.

2. Select **File** in the top panel of the xfig window and load the file `Example.fig` from the directory `./projects/Example`.

3. With the **Copy** button in the left panel copy the storage into the scrapfile `.xfig` like in step 2a) of the paragraph above.

4. `Now` use an editor to look at `.xfig` in the ColSim directory (during the installation of ColSim, a link was set from .xfig in the home directory to the ColSim directory.) After some data concerning the graphics, a block with readable text appears: First, the type number of the unit is given. Then, the parameters follow, after the big letters PAR. Then the input inital values, following the letters INP, after them the output variables and finally the node temperature inital values after the letters DERIV. See the paragraph above about the menu based parameter editing for a more detailed explanation.

5. Edit the values, save the file and leave the editor. Proceed according to step 5 of the previous paragraph.

6. To try the new configuration, the xfig graphics must be converted into a dek file and the simulation started according to the instructions given in section 3.2.2.

See the comment at the end of the previous subsection for the features of the **Delete** function of xfig. Instead of the **Sort .xfig** button, the shell script `xfig.ssc` in `./etc` can be used.

4.3 Construction of a new system

4.3.1 Putting together the units

In the following, there won't be individual instructions for menu based and manual operation, since all necessary commands were explained already for both approaches.

First, load `Example.fig` into a xfig window, adjust the scale with the **Zoom** function (lowest button in the left panel) to find a convenient scale and delete all units, lines and text, except for the unit `sim control`. This unit is needed in every system, since it sets beginning and end of the simulation, see section 5.5.1.

In the following, a simple system with two hydraulic cycles including a solar collector will be constructed as an example. The units needed will be taken from the library in the folder `./cnv/lib`, which contains the graphical representations of all units belonging to the `SchichtSpeicherSystem`, our reference system for the fluid systems. If this library is not available or not complete in your ColSim version, simply copy the units directly from `SchichtSpeicherSystem.fig` in the folder `./projects/SchichtSpeicherSystem`. Of course, the chosen units can also be copied from other xfig system graphics if present,

like e.g. `StandardKollektorAnlage.fig` in `./projects/StandardKollektorAnlage`. Note, however, that the parameter settings have to be checked more carefully when combining units from completely different systems: E.g. the heat capacities of the fluids must be chosen uniformly for all units passed by the same fluid. Otherwise, the simulation results will be peculiar, though no explicit error will be reported.

Note that if a clear orientation of the units is wanted, the **Grid** mode of xfig is helpful (button in the bottom panel).

1. Open a second xfig window and load the file `collector.fig` from the library and copy and paste it into the first xfig window.
2. Do the same with the following units: The units: `weather`, `storage`, `heat_exchanger`, two pipes and two `pumps`, `preferably` one from the solar cycle and one from the heatexchanger-storage cycle. Of course, the graphical appearance and the type number of the two pumps are identical, but their (in the graphics not visible) parameters are different. For convenience, those units are selected with suitable parameters for the actual purpose.
3. Place the unit `weather` in the upper left corner, e.g. below the unit `sim control`.
4. Place the collector next to the unit `weather`, then arrange the two pipes in such a way that they can be easily connected with the collector's OUT1 and INP1. In the follwing, we address output boxes of the xfig graphics as OUT and input boxes as INP.
5. Place the pump taken from the solar cycle and the heatexchanger next to the other ends of the pipes.

4.3.2 Hydraulic connections

Now the hydraulic cycle will be established by connecting the mass- and heatflow ports of all hydraulic units with lines. Originally, mass- and heatflow had separate ports in the graphic, since they are balanced separately by the program also. However, since mass- and heatflow always come together, it is now sufficient to connect only the heatflow port (always the small odd numbers), whereas the corresponding mass flow ports (always the small even numbers) are connected automatically during the conversion of the system graphics to the dek file.

To draw the connecting lines, select the zigzag line symbol from xfig's left panel (the comment **POLYLINE** drawing appears at the top left corner). Then, a color can be selected with the button **PenColor** from the bottom panel. For best clarity of the picture choose blue lines for cold water and red lines for hot water, line width 3, and let the lines follow the grid lines with as few bending points as possible.

Note that the way the line takes is not important as long as it begins in an input box and ends in an output box, or vice versa. Input boxes have an inwardly facing arrow tip, output boxes an outwardly facing one. The exact placing of the lines is easier, when the mode **Point Posn** is used in finest resolution (button in the bottom panel),

then all bending points and endpoints fall onto a 1/2 mm grid. To choose beginning and bending points of the line, use the left mouse button, to finish it use the middle mouse button.

- Connect the following units with a red line, always going from OUT 1 to INP 1: Collector - pipe_1 - heat_exchanger.
- Connect the following units with a blue line, also from OUT 1 to INP 1: Heat exchanger -pump - pipe_3 - collector.

Now the second hydraulic cycle is established: Place the second pump and the storage to the right of the heatexchanger and

- connect with a red line: OUT 3 of the heatexchanger with INP 1 of the pump, OUT 1 of the pump with INP 1 of the storage.
- Connect with a blue line: OUT 1 of the storage with INP 3 of the heatexchanger.

Now the hydraulic connections are complete. It's recommendable to adjust some parameters for a more interesting output: Set the simulation start in the unit `simcontrol` to 940701 and the end e.g. to 940801. Adjust the storage's start temperatures (the `DERIV` values) to e.g.10 C.

4.3.3 Data and controller cables

Next, the data input units are connected. Here, it's only the weather unit but mostly there's also a `load_profile` enabling user defined input for various units.. Use yellow lines to

- connect OUT 4 of the weather unit - the global radiation onto a horizontal plance - to INP 4 of the collector.
- Connect OUT 5 - the diffuse radiation to a horizontal plance - with INP 5 of the collector.
- Connect the ambient temperature data on OUT 1 of the weather unit with the collector (INP 3) and both pipes (also INP 3). These two pipes are considered to be outside of the house or in unisolated parts of it, e.g. under the roof, since their ambient temperature is the air temperature outside. In the reference system `SchichtSpeicherSystem`, there are two more pipes, representing pipes inside the house, which are at room temperature. There, INP 3 is not connected, but it's input inital value is set to 20 degrees.

The solar system still lacks a controller which decides when fluid should be pumped through the collector and the heatexchanger to transfer solar heat to the storage. Copy and paste the controller device, preferably that from the solar cycle, into the graphics below the pumps. Use thin dashed black lines to

- connect the control signal at OUT 1 of the controller with INP 3 of both pumps.
- Connect INP 1 of the controller with the absorber temperature at OUT 3 of the collector.
- Connect INP 2 of the controller with the storage temperature at temperature sensor 2, i.e. storage's OUT 12.
- Connect INP 3 of the controller with the temperature at the storage's top, OUT 11.

The controller compares the storage temperature with the collector temperature and sends an on or off signal to the pumps. See section 5.5.2 for a detailed explanation of the controller and its various modes.

Fig.4.3.3 shows how your system could look like now. In order to facilitate the comparison of the lines, they were drawn next to each other here. However, more complicated system graphics are better to survey when parallel lines of the same type (e.g. controller connections) are drawn onto each other. In the xfig graphics you can always use **Move** and afterwards **Undo** to distinguish the lines.

4.3.4 Error messages of fig2dek (cnv.exe)

After saving `Example.fig`, the conversion to a dek file can be started. However, when trying to convert a newly drawn system, very often error messages like the following appear:

- `no input-box found for output_unit:[5,3] !!`
  `no connection found for a line-symbol (return read_connections()=-1.`

Here, a line beginning in unit 5, output number 3 does not end in an input box.

- `no outputbox for a line-symbol found: !! (no double out_no's!) current line from [-4.6][3.0] to [22.4][1.5] cm`
  `no connection found for a line-symbol (return read_connections()=-1`

Here, a line was found which either meets no output box or no boxes at all. The line's coordinates (x,y) in cm in the xfig graphics are given.

When evaluating error messages, be aware that the unit numbers can change during the conversion process, when a system with newly added or differently arranged units is converted for the first time (see section 6.3). When working with the ColSim menu, it is sufficient to close the old xfig window after the conversion process and reload it by pressing the **xfig** button.

When working without the ColSim menu, the file `./cnv/sim_new.fig` can be used to identify the unit numbers corresponding to the error messages. Once the conversion runs without errors[2]. this file should be copied onto the system xfig file in the projects folder.

[2]If the conversion process breaks off due to severe error, some units may be missing in `sim_new.fig`.

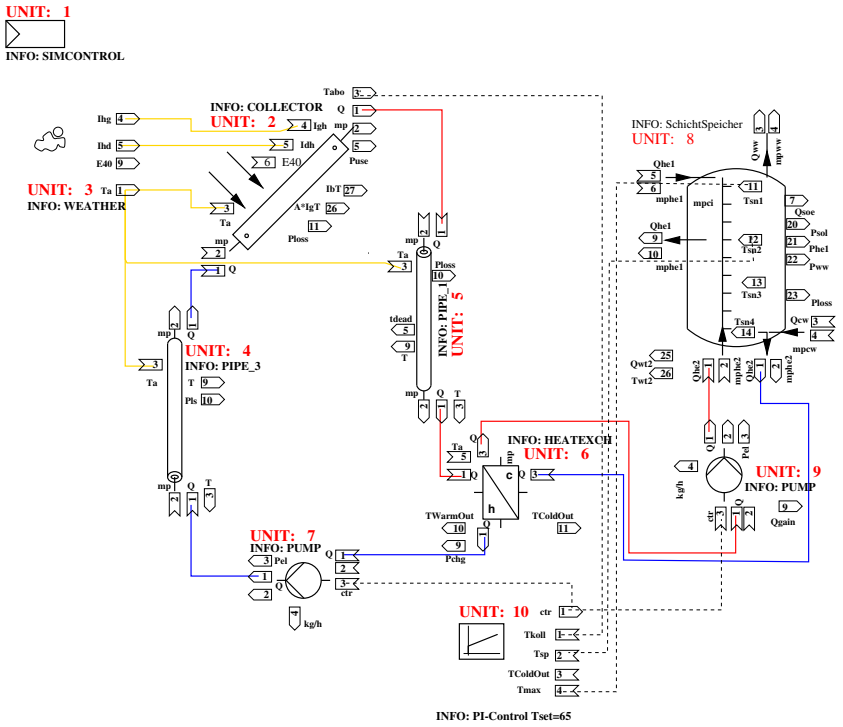


Figure 4.2: This is how your Example.fig should look like approximately.

4.3.5 Connecting plotting and printing devices

If the conversion process was successful, the simulation can be started. However, there is no output yet informing about the proceedings in the system. For this purpose, the plotting and printing devices were designed. Copy the `gnuplotter` and one of the printers into `Example.fig` and connect the output variables of your choice to the plotter's and printer's input boxes.

The gnuplotter:

Here's a suggestion for the connection of the plotter:

1. Connect e.g. the temperatures of the collector's absorber plate (OUT 3), of the ambient (OUT 1 of weather) and some of the various temperature sensors of the storage (OUT 11-14) to INP1..5 of the gnuplotter.
2. Connect e.g. the control signal of the controller (OUT1) and the power of the solar energy entering the storage (OUT20 of storage) to INP6, 7 of the gnuplotter.
3. Make sure that suitable scaling factors are chosen in the gnuplotter: For the above choice of connections, all input inital values of the gnuplotter should be 0 except that of INP7: Set it to `1e-3`. Set `PAR6` (y-axis of second diagram) to 3.

After saving `Example.fig`, converting it with **fig2dek** or `cnv.exe` and starting the simulation run, the two gnuplot windows should appear and show the connected quantities as a function of time. If no such windows appear, make sure that PAR7 of the gnuplotter is set to 1. If no lines appear or some are missing, check the connections and the scaling factors.

For each connected input, an entry appears in the upper right corner of one of the gnuplot window and also the chosen scaling factor is displayed here.

The printer:

Connect some interesting output variables of various units to the printing device. Depending on what printer you copied into the drawing -printer 0,1 or 2 - the output file will be sent to the files `sim_out0.dat`, `sim_out1.dat` or `sim_out2.dat`, all in the ColSim directory.

Keep in mind that the printer is able to process the data it receives by summing it up over a given time span, integrating it or finding out the mean value. These modes can be set for each printer input individually with its input inital value. See section 5.5.6 for details.

4.3.6 Adding output boxes to units

Not all output variables of a unit appearing in the `info_edit` window or in the source code of the type can be displayed in the system xfig picture. However, these boxes can easily be added to the respective unit in the following way:

1. Open a second, empty xfig window by typing `xfig`. Use the **Copy** or **Delete** functions with the right mouse button together with the **Paste** function to copy the unit to be modified from the system graphics into the new xfig window.
2. Use the **Break Compound** button approximately in the middle of the panel of xfig. It's the one where the explanation: `Break compound object` appears, when the mouse pointer is placed inside the button. Then double click on one of the little boxes marking the corners of the unit. Now the compound is opened, and instead of one xfig object, the unit consists of many of them, all marked with little boxes.
3. Copy one of the output boxes to an empty place in the drawing. Open the compound of the output box by repeating the steps listed above. Then use the **Delete** function to delete the little number inside the copied output box.
4. Activate the text mode by selection the big **T** in the left panel of xfig. Choose text size 20 and text font Times-Bold from the lower panel. Then insert the new output number into the copied output box. The insertion of text must be finished by hitting the Return or Enter key.
5. Put the output box and its number into a compound by using the **Glue Compound** button next to the **Break Compound** button. The explanation `Glue objects into compound object` must appear. Then use the MIDDLE mouse button two times to define a small rectangle enclosing the new output box. When this operation was successful, the new compound object is surrounded by four black little boxes. Now click on one of them with the right mouse button and the box and the text are defined as a new compound object.
6. **Move** the new output box to a suitable place inside the unit. If only the box moves and the output number stays in its place, or vice versa, the building of the compound object was not successful.
7. Define the unit as a compound again in the same way as explained above. By moving the object around afterwards, it can be tested whether all parts of it are included in the compound. Take special care of the parameter block, which is hidden, but can be detected by the symbols: <<>>.
8. **Copy** and **Paste** the modified unit back into the system graphic. Now the new output unit can be connected. If there are many error messages now during the conversion process, probably the modified unit wasn't defined as compound properly. If there's only one error message but the connecting line does meet the new output box and some input box, you may have to repeat the complete process. Probably something went wrong with defining the additional output box as compound object and `cnv.exe` does not recognize it.

4.4 Runtime errors

Missing line: If the conversion of the simulation script with **fig2dek** or `cnv.exe` was successful, but after starting the simulation run a message like the following appears:
`init_set_mp: no connection found, when searching a following unit to: [19][1] ([unit][output])`,
you forgot a line connecting hydraulic unit 19 with its subsequent hydraulic unit. This error was not detected during the conversion process since here it is only checked for flawed connections, not for missing ones.

Incomplete conversion: If there were error messages during the conversion process which you ignored, it also results in a runtime error like this.

Missing weather data: Another common runtime error is caused by missing weather data in `./weather`, then something like `datafile 'weather/try07/910101.dat' not found` appears on the screen. Since the weather data is quite extensive, not all TRY data bases (see section 5.5.5) are present in the standard ColSim versions. Look in the folder `./weather` and find out which data bases are available in your version, then adapt the parameters in the `weather` unit and eventually also in the `sim control` unit (the simulation time span must be covered by the weather data).

Wrong time interval: If the message is `time_intervall of 'load.dat' is different from parameter1 of load_reader`, the time span covered in load.dat (a link to the load profile data file in the projects folder) differs from that given in PAR1 of the load profile unit.

Error message by unit: These error messages begin with the number of the unit concerned and the name of the type. If you didn't edit the source code, the error will mostly stem from the parameter settings. Use **Sort .xfig** to make sure that you got the actual parameter comments and check whether the values are reasonable. If this doesn't work, take a look at the source code of the type in `./src` and find out under which condition the message is printed out. You can activate the analytic mode by setting `a=1` at the beginning of the code, or add new print lines to check the variables concerned.
Note that the problem can also be caused by the previous unit which e.g. sends absurd values for the heat flow. Some thermodynamical functions used here (see section 5.6.4) use fit formulas which are only reliable for temperatures above 0 C and which cause funny results at e.g. -50 C. Then, a negative relative humidity may be calculated from this which causes an error in the unit sending the message.

Error message by main: Such a message may begin like: `System-Energy-Check: abs. Error: 12345 [Ws], script file: ' more error_script '` In ColSim, a complete energy balance is performed at each time step, which is a huge advantage because many errors are detected by this. If the energy balance is incorrect,

i.e. if the energy stored momentarily in the total system minus that at simulation start differs from the total of all gains minus the losses by more than a given maximal error (PAR5 of sim control), the simulation is stopped. An energy balance of the individual units is performed to determine whether they had exchanged energy in the current time step. These units, called active units, are listed with their numbers and the "culprit" is most likely among them.
If you didn't modify the source code, you probably forgot or deleted a necessary connection or forgot to set necessary parameters.[3] First, check for warnings you may have overlooked: Restart the simulation by typing `sim > dummyfile` (some new filename) in the ColSim directory. Then all runtime messages can be checked unhurriedly by looking at `dummyfile` with an editor or the commands `less` or `more`. Also take a look at the file `error_script` in the ColSim directory.

[3] Of course it's the job of the ColSim programmers to prevent flawed parameter and line settings which cause such a crash. Please contact one of them, e.g. via the ColSim homepage www.colsim.de

Chapter 5

The ColSim types

5.1 General remarks

In the following all available types, i.e. system components like pipes, pumps, storages, controller, collectors, walls, heaters, etc. are discussed in detail. People who are not interested in the physics and the modeling of a type may only read the first part of each section concerning the operation of the type, i.e. how it is integrated into the system and how its parameters should be set. However, only those parameters are discussed that are not sufficiently explained in the source code comments (readable with the help of the info-edit panel).

Also for the in-and outputs, this source of information should be used primarily. If the meaning of an in-or output cannot be resolved from the comments, one should take a look at the source code describing the type under consideration (all source codes or links to the source codes are in `./src`) and search for the part where calculated quantities are assigned to the in-or output.

You'll find that not all in-and outputs which appear in the info-edit panel are visible in the xfig picture of the unit. There are two possible explanations why an existing in-or output is not shown there:

- Itt's not interesting. E.g. if INP n is a heat flow inlet we know that INP n+1 must be the corresponding mass flow inlet, which will be automatically supplemented during the conversion of the xfig system graphics to the dek file.
- There are too many of them. To reduce the complexity of the graohics, not all existing outputs informing about the current state of some system variable are represented by the corresponding numerated output boxes in the xfig-picture of the unit. See section 4.3.6 for instructions about how to add the extra in-or output boxes you need to the xfig picture.

Note that OUT 6,7 and 8 are always reserved for the unit's gains, its losses and its internal energy, respectively.

In the simulation, time is discretized into small steps of size `h`. Therefore, the fluid moving through the hydraulic units is also discretized into so-called plugs: The mass

of one plug is $mp \cdot h$, where mp is the mass flow of the fluid, here in `kg/s`, and `h` is the simulation time step in seconds. So one plug is the fluid amount which is passed from one hydraulic unit to the next during one timestep h. However, at the inlet ports of the unit, not the mass of the plug arriving is given but the current mass flow in kg/s, which is internally multiplied with `h`.

On the contrary, the heat flow arriving at the inlet ports of a unit is already multiplied with `h` and given in `J=Ws`, i.e. it's really an amount of energy ΔQ. Nevertheless, we address both quantities as "mass flow" and "heat flow", respectively.

5.2 Fluid types

5.2.1 Pipe

Operation The fluid pipes are used to simulate the transport of fluid between two other hydraulic units in a more realistic way than it would be if the units' mass flow in-and outlets were simply connected by lines. The heat loss to the ambient can be modeled and, with `pipe3.c`, also the "deadtime": If there is more than one node, the fluid entering the pipe in one time step is not the same which leaves the pipe in the same time step. This effect reduces the efficiency e.g. of solar collectors, since every time the solar pump stops, hot fluid is left in the pipes and looses its energy to the ambient instead of transfering it into the storage. The controlling mechanism should take this effect into account in order to maximize the efficiency.

Modeling There are two models for the pipe, the corresponding source code files are called `pipe1.c` and `pipe3.c`. `Pipe1.c` is a simple node model without an exact representation of the deadtime: In each timestep, the incoming fluid plug is mixed with the fluid of the first node. From this mixture, a new plug is taken and mixed with the fluid of the second node, etc. From the last node, the outgoing plug is taken. The losses to the ambient are calculated for each node while the plug is present. The temperatures of all nodes are stored in the `deriv`-arrays.

The alternative model, `pipe3.c`, is based on the opposite view: The incoming plug presses the fluid in the pipe forward without mixing with it. From the end of the pipe, the outgoing plug is taken. Then, the losses to the ambient are calculated for each node. Since after each time step, only the node temperatures are stored, there is an effective mixing within the volume of one node. Therefore, the form of the deadtime curve (temperature of outgoing plug as a function of time, after a jump in the temperature of the incoming fluid) can be varied from a step function form for infinite number of nodes to a smooth increase for only few nodes. Mostly, 5-10 nodes will be used. The deadtime itself is independent of the number of nodes and corresponds to the travel time of the plug through the pipe.

Both pipe models employ an Euler stabilization for the losses to the ambient: If due to the linearization of the actually exponential function (heat loss as a function of time) cooling below the ambient temperature level occurs, e.g. for large time steps h

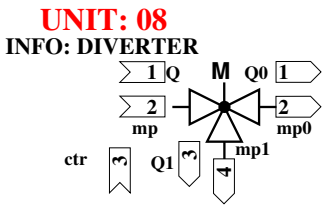
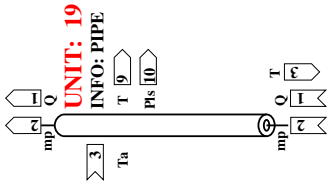


Figure 5.1: Diverter

or large values of kA, the losses to the ambient are limited in such a way that exactly the ambient temperature is reached by the node under consideration. Therefore, the calculation is stable for all timesteps and heat transfer coefficients kA/l.



5.2.2 Diverter

The `diverter`[1] models the branching of a pipe (a T-piece) by splitting the mass (and heat) flow into branch 0 and branch 1 (see fig.5.2.2). At the current time, this is the only mode of operation, so PAR1 must be set to 1. How the fluid is distributed to the two branches is defined by the control signal `ctr = 0..1` from INP3: The flow in branch 0 and branch 1 equals the incoming flow, multiplied with `ctr` and `(1 - ctr)`, respectively.

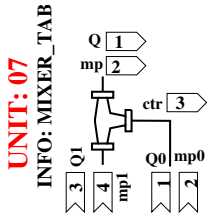
Since at the current time, there is no hydrodynamical calculation done for the fluid flow in the tubes, the flowdiverter is a very simple component part. Note that every diverter must be matched by a mixer 5.2.3 reunifying the mass flow.

5.2.3 Mixer

The mixer models a T-piece unifying and mixing the mass flow of two branches (branch 0 and branch 1), which were separated by a preceding diverter. At the current time, this is the only mode of operation, so PAR1 must be set to 1. OUT3 informs about the

[1] In older ColSim versions, the source code was called `flow_div.c`

contribution from the indiviudual branches: Its signal is calculated from `1 - mp0/(mp0 + mp1)`, where `mp0` and `mp1` are the incoming mass flows from branch 0 and 1.



5.2.4 Pump

Operation

In every hydrodynamic cycle, there must be a pump to create mass flow. Since up to now, there's no hydrodynamical calculation of the fluid flow in ColSim, the pump is modeled in a very simple way: PAR1 contains `mp_max`, the maximal mass flow the pump can maintain. PAR 5 sets the mode of operation; If the pump is run in mode 0, it receives a control signal `ctr=0..1` from INP3. The output mass flow is then given by `mp_max`, multiplied by `ctr`. Alternatively, the pump can run in mode 1, where the target mass flow is read directly off INP4. Mode 2 is a combination of these cases, where INP4 is used if it is connected, and INP3 otherwise.

With PAR2, the heat capacity of the fluid passing the pump is set. Note that it causes errors hard to detect when this heat capacity is not the same in all hydraulic units in one cycle. E.g. in the SchichtSpeicherSystem, all units belonging to the solar cycle, i.e. collector, pipes, solar pump and the hot side of the external heat exchanger, have their heat capacity parameter set to 3.8 kJ/kgK, corresponding to a mix of water and an anti–freeze agent. The hydraulic cycle containing the cold side of the external heat exchanger, i.e. the storage and the pump inbetween them, as well as all other hydraulic cycles in the system have their heat capacity parameter set to 4.19 kJ/kgK, that of fresh water.

The pump's electrical power consumption in primary energy units is calculated from the given primary power consumption at maximal mass flow (PAR3) and a power law whose exponent `a` is given in PAR4: $P_{el} = P_{max} \cdot (mp/mp_{max})^a$. In the ideal case and for laminar flow, i.e. low mass flow of 25-100 kg/h, the exponent a is 2.

The pump's thermal mass can be tuned with PAR7, see next section for details.

Modeling

In section 6.4.1, it is explained why in ColSim the simulation of each hydrodynamic cycle starts and ends at a pump. Consequently, the pump units are called twice in every time step: During the first call, the new mass flow for this timestep is calculated like explained above. The new heat flow leaving the pump is calculated from the pump's heat capacity multiplied with its temperature.

Actually, the pump's heat capacity is not really a physical property of the pump and its fluid content alone. Rather, it is meant to summarize system capacities not considered yet (e.g. those of pipe walls, mixers, diverters, tabs, sockets, etc.). The pump's thermal mass is set by PAR7, which is then internally multiplied with the fluid's heat capacity (PAR2) to give the pump's capacity. If PAR5=0, the pump's thermal mass is set to the maximum mass of one plug, i.e. $mp_{max} \cdot h$.

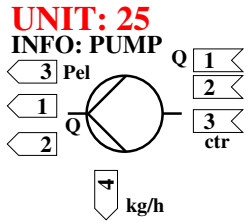
The pump's temperature is taken from the array element `deriv_o[unit][1]`, which was written during the second call of the pump in the previous timestep (see section 6.4.2 for an explanation on global fields like `deriv_o, deriv_n, qp_sum` etc.).

When mass and heat flow output values are calculated, the outgoing energy is added to the array element `qp_sum[unit][1][1]`, which summarizes the pump's output of energy.

In the second call of the pump in the same time step, it is checked whether the incoming mass flow equals the outgoing mass flow like calculated during the first call. If this is not the case, the simulation is stopped. Furthermore it is checked whether vanishing outgoing mass flow is combined with non-vanishing incoming heat flow or vice versa.

The incoming heat flow is added to the array element `qp_sum[unit][1][2]`, which summarizes the pump's energy input. Then, the new pump temperature is calculated from the pump's initial energy minus the summarized output plus the summarized input, everything divided by the pump's heat capacity.

The source code for the pump described above is `pump.dynamic.c`. There is a non-standard alternative, `pump.static.c` in `./src/more_types`, which works without a heat capacity for the pump. In principle, this pump model leads to the same results for a longterm simulation as `pump.dynamic.c` with PAR7=0 (minimal thermal mass). However, the dynamical pump is preferable to the static one since the latter tends to accumulate small unwanted and unphysical energy gains which result from differences in energy in-and outflow due to the finite timestep h.



5.2.5 Solar collector

Operation: The solar collector model absorbs solar radiation and transfers the energy to a fluid cycle. Apart from the collector's geometrical properties, it is specified by:

- the collector efficiency factor F', which is the ratio of two heat transfer coefficients, namely that from the fluid to the ambient air and that from the absorber plate to the ambient air.
- the average transmittance-absorptance product $\eta_0 = (\tau\alpha)_{av}$, which is defined as the ratio of the absorbed solar radiation to the incident solar radiation. The name of this quantity results from the fact, that is is some functional of the transmittance of the absorber's glass cover τ and the absorptance of the absorber plate or tube α. In first order it is indeed the product of α and τ , but due to the effects of multiple reflections, wave lenght dependence and the fact that the back reflection of the absorber is diffus, there are corrections.
- the collector overall heat loss coefficient $k[W/m^2K]$, which summarizes the heat losses to the ambient.
- the total heat capacity of the collector per squaremeter (including fluid), given in PAR12.

In the collector's parameter list, the product of $F'$and η_0 for beam radiation of perpendicular incidence must be given in PAR2. For flat plate collectors, this product is multiplied internally with a function $f(\vartheta)$, describing the angular depencence on the incident angle (see section "Modeling" for details). In PAR14, the parameter b0 for this function must be given, e.g. -0.12 for a collector with two glass covers.

For a CPC collector, PAR14 must be set to -1, and a table called `iam_table.dat` must be present in `./iam`, giving the angular dependence as a function of two incident angles, since the CPC has no rotational symmetry. See section 5.6.3 for details.

In PAR3, the product of F' and $\eta_{0,diff}$ for diffuse incident radiation should be given. It is smaller than that for the beam radiation, since it is already multiplied with the angular dependance and integrated over all incident angles.

With PAR4=k_0 and PAR5=k_1, the product of k and F' is specified, where the Ansatz $k \cdot F' = k_0 + k_1 \cdot \Delta T$ is used, i.e. the product $k \cdot F'$ is assumed to depend on the temperature difference between fluid and ambient.

In PAR7, the specific heat capacity of the collector fluid should be given. Note that the same value must be used for all hydraulic units in the collector cycle.

If PAR15 is set to 1, the file `./horizon/horizon_table.dat` is read which contains information about shading objects which reduce the horizon (see section 5.6.2).

The collector must be connected to a weather unit (see section 5.5.5): From INP4 the global irradiance on a horizontal plane (Igh) is read in, from INP5, the diffuse irradiance on a horizontal plane (Idh). From INP3, the ambient temperature is read.

Modeling:

The standard ColSim collector model is an $1 \times n$-type model, which means that n nodes are used in the flow direction and 1 node in the vertical direction. So, the absorber plate and the fluid are treated as one capacity, and the internal heat transfer between them is not modeled. There is also a $2 \times n$-collector model, where absorber and fluid have different capacities, but stability problems can occur in certain circumstances.

The collector consists of an absorber and a fluid pipe in thermal contact with the absorber. Along the pipe, the collector is divided into n nodes. Just like in the pipe model described in section 5.2.1, in each time step a fluid plug enters the collector pipe. In a loop over all nodes, the energy balance for each node is performed and the new node temperature is calculated. At last, the outgoing plug is calculated. However, in difference to the normal pipe model, the collector pipe receives also energy from the absorber plate. The energy difference between solar gains and losses to the ambient per node for one time step h are described by the following formula (see [1],p.271):

$$\Delta Q = A/n \cdot (F' \cdot \eta_0 \cdot f(\vartheta, \varphi) \cdot I_{beam} + F' \cdot \eta_{0,diff} \cdot I_{diff} - k \cdot F' \cdot (T_i - T_{ambient})) \cdot h,$$

where A is the absorber area, n the number of nodes and T_i the current temperature of node i. The beam and diffuse solar irradiance I_{beam} and I_{diff} onto the collector which its slope and orientation as given in PAR8 and PAR9, are calculated by the radiation processor `rad_processor.c` (see section 5.6.1 for details). The function `horizon.c` is called afterwards to check whether the direct sun is shaded by something, e.g. the roof of a neighboring house (see section 5.6.2 for details.) Then, the function $f(\vartheta, \varphi)$ for the angular dependent transmission is calculated with the help of `inc_angl_mod.c`, see section 5.6.3for details).

There are two standard collector models, `collector1xn.c` and `collector.pipe3.c` which are based upon the pipe models `pipe1.c` and `pipe3.c`, resepctively. Apart from this, they are equal.

5.2.6 Heat exchanger

Operation:

The heat exchanger is a device inside of which heat is exchanged between two streaming fluids. Consequently. it possesses two mass flow inlets and two outlets. Up to now, the heat exchanger type can only model a counterflow heat exchanger, therefore PAR1 must be set to 1. With PAR2 and PAR3, the specific heat capacities of the fluids on the hot and the cold side can be given, respectively. E.g. if water with an anti-freeze agent is used on the hot side, the value could be something like 3.8 kJ/kgK, whereas on the cold side there will mostly be simple water with a value of 4.19 kJ/kgK. In PAR4, the overall heat transfer coefficient-area product in W/K must be given.

Modeling:

Our standard type is the static heat exchanger `heatexchanger.static.c`. Here, only the capacities of the passing fluids are regarded, and the heat exchanger itself neither has a heat capacity nor an internal volume. Consequently, no losses to the ambient are calculable. There is also a dynamical model, `heatexchanger.dyn.c`,but is hasn't been in use recently.

The modeling is based upon the schematic adiabatic counterflow exchanger described in [1], p.178f. The kA-value given in PAR4 equals the theoretical heat exchange

per temperature difference for two fluids with infinite heat capacity. Since in reality, the fluids heat capacities are finite and therefore their temperatures change during the heat exchanging process, the exchanged heat is smaller than this. However, from the kA-value, the effectiveness of the heat exchanger can be calculated, which is defined as the ratio of the actual heat exchange that occurs to the maximum possible one.

5.2.7 Storage

Operation:

General remarks: The storage is a central module of a fluid system, since it is connected to all hydraulic cycles. There is only one standard storage type and all varieties like storages with or without internal heat exchangers or with stratifiying tubes can be modeled with the corresponding parameter settings. However, for better survey different xfig pictures can be used which remind of the storage specifications: The left side of fig.5.2.7 e.g. shows a picture of a storage with internal solar heat exchanger, the right side of fig.5.2.7 shows one without internal heat exchangers, but with a stratifying tube.[2]

The storaget's heat loss coefficient to the ambient is given in PAR1, its volume in PAR2 and the specific heat capacity of the fluid stored in it in PAR3.

In-and outlets: At the bottom of each storage in fig.5.2.7, the in-and outlet for the solar cycle are drawn (INP1 and OUT1). At the storaget's left side, the in-and outlet for the heater cycle can be found (INP5 and OUT9). The cold water inlet INP3 is at the right side at the bottom, and the hot water leaves the storage through OUT3, drawn at their top. It's also possible to connect a circulation or heating cycle (INP7, OUT17), though the in-and outlet boxes are not drawn in this picture. They can simply be added to the xfig-picture (see section 4.3.6). Of course the position of the individual in-and outlet boxes in the xfig-pictures does not matter, they can be arranged completely differently for another storage model.

The height where in-and outlets are attached to the storage can be varied by editing the storage's parameters, e.g. PAR5 and PAR6 give the positions of the ports for the heater cycle, PAR11 and PAR12 for the solar cycle . Since the storage is modeled as a chain of n nodes (where n is given in PAR4), the positions are a number between 1 and n, where node 1 refers to the top of the storage, node n to the bottom of the storage.

Also the heat capacities of the fluid entering through the inlets must be adjusted, using PAR7 and PAR13.

[2]In the stratifying mode, the incoming warm water is fed into a node of the storage where the water is just somewhat cooler. So convection and entropy creation are inhibited. In reality, this stratification is achieved by a tube with shutters which open only when the water in the tube has almost the same temperature as the surrounding water.When the warm water enters the tube at the bottom of the storage, it rises inside it (due to its increased temperature with respect to the surrounding water in the storage, its density it lower, which leads to thermal buoyancy) and leaves it just in the right layer of the storage.

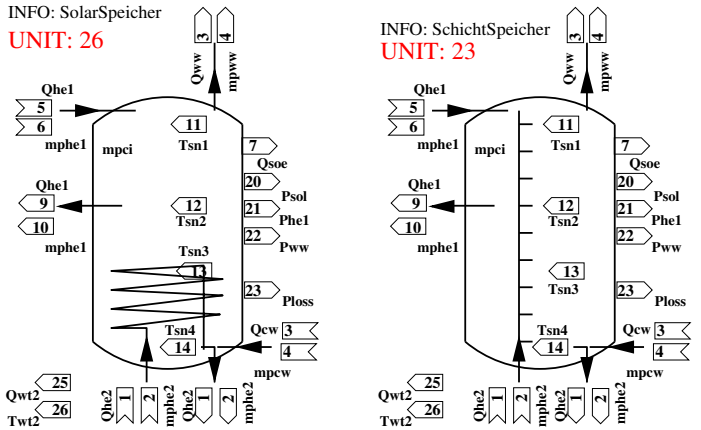


Figure 5.2: Left side: Simple solar storage. Right side: Stratifying solar storage.

Mode of heat conversion: The different pictures in fig.5.2.7 are only used for better orientation. Whether the storage works with or without internal heat exchangers is solely decided by PAR22:

- If PAR22 is set to 0, the storage has two internal heat exchangers, called HE1 and HE2, which are used for the auxiliary heater cycle and the solar cycle. In this mode, PAR8,9,10 as well as PAR14,15,16 must be set since they specify the physical properties of both internal heat exchangers.
- If PAR22 =2, there is only the heat exchanger for the solar cycle (like in the left picture of fig. 5.2.7). PATR14,15,16 must be set.
- If PAR22=1, there is no internal heat exchanger and the water arriving at the inlets of HE1 and HE2 enters the storage directly. However, for the reasons mentioned above, the in-and outlet positions of HE1 and HE2, as well as the heat capacities of HE1 and HE2 have to be set nevertheless.

Note that if the mode of operation is changed, sometimes the heat capacities of HE1 and HE2 have to be adapted. E.g. if the storage is run in mode 2 and an anti–freeze agent is used inside the collector, the heat capacity for HE2 (PAR13) must be something like 3.8 W/kgK. When the mode is changed from 2 to 1, i.e. the internal heat exchanger for the solar cycle is turned off, an external heat exchanger must be used in order to avoid the mixing of the fresh water in the storage with the anti–freeze agent from the collector. Then, the storage is not connected directly to the solar cycle anymore, but via the external heat exchanger-cycle, where fresh water is circulating. Consequently, the heat capacity of the water entering the storage must be set to 4.19 kJ/kgK, the value for fresh water without glycol.

Mode of operation The generic storage device can be tuned to simulate two different kinds of storages, namely a normal storage and a stratifiying storage, which differ in the way the incoming warm water from the solar cycle is filled in.

- For PAR28=0, the storage is run in the normal mode.
- For PAR28=1, the storage is run in the stratifying mode, symbolized in the right picture of fig.5.2.7. The presence of a stratifying tube is assumed, which guarantees that the hot water entering the storage is let out in a segment with similar temperature. By this, convection is suppressed and the water in the storage is stratified with temperatures rising from the bottom to the top. The solar system works more efficiently since the solar (Vorlauf) temperature is always kept low. Thermal losses of the storage to the ambient are smaller, since only the water in the top region must be kept at high temperatures for immediate use. Note that in this mode, PAR11 and PAR12 don't have to be set, since the inlet node for the solar cycle is adjusted automatically and the outlet node is always set to n.

Modeling:

As mentioned above, the storage is modeled as a chain of n nodes, where n =10 normally. The nodes are characterized by their temperatures stored in the arrays deriv_o[unit][i] and deriv_n[unit][i] (see section 6.4.2 for a general explanantion of the global arrays), where unit is the unit number of the storage and i counts the nodes. Since the storage is usually connected with several hydraulic cycles, it will be called more than once within one timestep (see section 6.4.1 for information about the order in which units are called). Only at the last call of each timestep, effects like convection, heat conduction and thermal losses to the ambient are calculated:

Convection: Convection sets in if the temperature at node i+1 is higher than that at node i. The physical picture is that bubbles of hot fluid start to rise inside the storage due to their lower density with respect to the surrounding colder fluid. They don't stop until they reach a level where the surrounding fluid has the same temperature. This convective process is modeled in the following way: If a non-vanishing convective heat transfer coefficient kA_down_up (PAR25) is given, the energy transfered in one timestep from the hotter node i+1 to the colder node i is given by the transfer coefficient multiplied by the temperature difference and the timestep h. If PAR25 is set to 0, maximal mixing between the two nodes is assumed, i.e. the temperatures are equalized in each time step.

Heat conduction: Heat conduction transfers energy from hotter nodes to colder nodes, regardless of the direction. If a transfer coefficent for heat conduction is given (PAR26, in W/K), the energy transfered between node i and node i+1 is calculated from this coefficient multiplied by the temperature difference and the timestep. If PAR26=0, a literature value of 0.598 W/Km for the heat conductability of water at 20

C is used. This value is multiplied with the storage cross section area (calculated from the storage volume, PAR1, and its height, PAR27) and divided by the node thickness, i.e. the storage height divided by the number of n. Note that the kA-value for the heat conduction calculated in this way underestimates the real effect if the storage contains internal heat exchangers!

Losses to the ambient: The storages's energy losses to the ambient are summarized in the heat transfer coefficient given in PAR1. If PAR27, the storage height, is set to 0, the energy losses per node in one timestep are calculated from PAR1 divided by the number of nodes, multiplied with the node's temperature difference to the ambient and the timestep h. For a non-vanishing value of the storage height, the losses of node 1 and node n at the storage's top and bottom are increased by 10 %, whereas the losses of the other nodes are reduced.

Energy transfer to storage with internal heat exchanger: Every time the storage is called as a part of some hydraulic cycle, it is calculated how much energy and/or water is fed into the storage by this cycle and how much is taken from it.

If there's a heat exchanger connected to the hydrodynamic cycle under consideration, the heat transfer from the fluid in the heat exchanger to the fluid in the storage is modeled with an internal pipe model based upon `pipe1.c` like described in section 5.2.1. The heat exchanger is regarded as a winded pipe through which the hot fluid travels from the inlet to the outlet. The pipe has as many nodes as there are in the storage between inlet and outlet. So when for each node the pipe's heat losses to the ambient are calculated, they represent the heat transfered to this specific node of the storage by the heat exchanger.

If no or a vanishing heat transfer coefficient for the heat exchanger of the solar cycle (HE2) is given, this coefficient is calculated from a fit function in case the heat exchanger is needed. This fit function uses the values of the incoming fluid's temperature, the node temperature at the inlet of HE2 and the mass flow through HE2. Note that the fit functions coefficents are determined from a fit to a 1500l tank.

Without heat exchanger: If there's no heat exchanger connected, the fluid enters the storage at the node where the hydraulic cycle under consideration enters. In every timestep one plug enters (one plug equals the amount $mp{\cdot}h$, where mp is the actual mass flow in kg/s and h is the timestep in s. See section 6.4 for details) and mixes with the nodet's fluid content. Then, another plug with the node's old temperature is sent from this node to the next neighboring node in the direction towards the outlet of the cycle, and so on.

Energy balance At every call of the storage unit, an energy balance for each node is performed. In order to reduce the accumulation of errors due to the limited machine precision, the different energy gains and losses treated above are stored separately in the field qp_sum[unit][i][j], where unit is the unit number of the storage, i counts the

nodes and j counts the different energies. The actual node temperature follows from the node's energy divided by the node's heat capacity.Note that the energy balance for the whole system is checked by the main program after each time step.

Alternative storage models The storage model described above corresponds to the source code `storage4.c` in `./src/more_types`. There is an older version, `storage3.c`, where the energy transfer into the storage with heat exchangers is modeled with an external call of `pipe3.c`.

5.2.8 Heater

Operation:

The heater or auxiliary heater is used to heat up fluids according to one of the following modes which are chosen with PAR4:

Mode0: Here, a set temperature can be given in PAR2 for the fluid leaving the heater. If there is mass flow and the control signal from INP 3 equals 1, the fluid is heated up exactly to this set temperature, given that the heating power needed does not exeed the maximum heating rate given in PAR1. This mode can also be used for cooling.

Mode1: In this mode, the set temperature can be variable or fixed: Ist is read off INP3. The heating power is limited to the maximum heating rate from PAR1, but there is also a lower limit given with PAR6. If the heating rate necessary to reach the set temperature is smaller than the minimal rate, it is replaced by this minimal rate. However, if the temperature of the outgoing fluid would exceed the set temperature by a hysteresis term given by PAR2, the heater is turned off. Cooling is not possible in this mode.

Mode2: Here, the set temperature is also read off INP3. If there is mass flow, the fluid is heated up exactly to the set temperature. There is no limitation by minimal or maximal heating rates in this mode. Cooling is possible.

Mode3: Here, the fluid is heated up with constant maximal power like given in PAR1, if there is mass flow and the control signal from INP3 is equal to 1.

Mode4: In this mode, the heating power which is applied to the fluid, if there is mass flow, is read off INP3. PAR1 is used as limitation of the heating power.

With PAR3, the specific heat capacity of the fluid can be chosen. With PAR5, it can be chosen whether the transformation of primary energy into heating energy is ideal, i.e. one-to-one (PAR5=1) or realistic (PAR5=2). In the latter case, the effectiveness of the heater is calculated from a fit function valid for a condensing boiler, i.e. the effectiveness can be greater than one, depending on the reflux temperature. Note that the value of PAR5 only effects OUT9 and OUT10 which give the auxiliary power and energy, used for heating up the fluid. In OUT4, always the heating power for ideal heating is given.

Modeling:

This type is modeled very simple, since it has no internal capacity except for the fluidt's capacity and consequently cannot represent heat losses to the ambient. A more realistic model should include an internal capacity, since with a very heavy heater, the system effectiveness can be lowered considerably if it starts up very often. Then, a great portion of the heating energy is only used to heat up the heater which afterwards looses its heat to the ambient in its inactive periods.

What the current heater model does is only to either add the set heating power to the fluids energy and calculate its new temperature or to set the fluid's temperature according to PAR2 or INP3 and calculate the heating energy needed for this.

5.2.9 Heating

The heating type is currently subject to larger modifications.

5.3 Ventilation types

5.3.1 Pipe_air

Operation

`Pipe_air` is used to simulate the thermodynamics inside a ventilated room whose volume is given in PAR1. The room is filled with humid air and there are in-and outlets for dry air, water steam, carbon dioxide, water and heat, where only the heat in-and outlets are shown in the xfig pictures. All the others are connected automatically. Pipe_air can be used in two ways:

1. Stand-alone within a ventilation cycle, e.g. together with a `heater_air` and a `pump_air`. Again, there are two possible modes of operations:

   - If PAR8 is set to 1, in PAR10 the total kA-value for heat losses of the air inside the room to the ambient must be given. In this way, a simulation of a simplified ventilated building can be performed.
   - If PAR8=0, the room to be simulated must have the form of an air tube whose properties are specified with PAR2,3 and 4. Then, the total kA-value for the heat transfer from the streaming air inside the pipe to the ambient is calculated. Note that in this mode also mass elements (see section 5.4.3) can be connected to `pipe_air` at the temperature inputs INP6-INP9. In this mode, `pipe_air` was used as an earth heat exchanger earlier.

2. The `pipe_air` unit is connected to an `AirRad_node` unit, i.e. INP14 is connected. Then, PAR8 and PAR2,3,4 and PAR10 are irrelevant, since the heat losses of the air contained in `pipe_air` are calculated by `AirRad_node`. The `AirRad_node` unit itself represents an air and radiation node of a building with 6-8 walls and must be connected to `wall` elements (see section 5.4.2 and 5.4.1). It balances the energy exchange of the walls with the ambient and wall heatings and sends the energy gained in each time step to the connected `pipe_air` unit. In turn, the `pipe_air` unit calculates the new equilibrium state and sends the current room temperature OUT16 back to `AirRad_node`. So when simulating a building, `pipe_air` is used to calculate the thermodynamics of the air inside and the air exchange via the ventilation, and `AirRad_node` balances the energies from the walls and windows.

If PAR6 is set to 1, the adsorption of humidity by imaginary walls can be simulated (this feature is independent of "real" walls connected to `pipe_air` via `AirRad_node`). PAR11 sets the maximal amount of steam which can be adsorbed and PAR12 gives the adsorption rate.

With PAR13 and PAR14, the inital humidity and CO_2 content of the air in the room can be adjusted. The initial room temperature must be given in DERIV1, the ambient temperature in INP6.

Of course, several pairs of `pipe_air` and `AirRad_node` units can be used to simulate a building with more than one room, see section 5.4.2for details.

Pipe_air has several inlets: INP6 can be connected to a `weather` unit (see section 5.5.5) to get the ambient temperature, otherwise the inital value of INP6 will be used. INP10,11,12 and 13 can be used to simulate the direct and indirect influence of people in the building who add water steam and carbon dioxide to the air by their breathing, bring in heat and turn on heat and water sources. If the inputs are not connected, constant values can be given as input initial values, but mostly they will be connected to a unit `load_profile` (see section 5.5.7), so a detailed profile of the occupantst' behaviour can be given.

Modeling

This module originates from an earth heat exchanger developped by Klaus Rittenhofer. At this time, it was the equivalent to the fluid type `pipe`, except for the fact that not only water and heat flow are balanced but also dry air, water steam and carbon dioxide. Since water steam can change into water when the temperature drops, more sumptuous calculations are necessary to determine the thermodynamical equilibrium than in a pure fluid system. The thermodynamical routines used are collected in thermodynamics.c, see section 5.6.4.

In contrast to the original earth heat exchanger, there is currently only one node in `pipe_air`, i.e there are no gradients in steam-and CO_2-concentration and there is only one temperature in the room. In a ventilation cycle, the mass stream of the dry air is kept constant throughout the cycle, though the mass stream of water steam and carbon dioxide may change. When a plug of dry air, steam and CO_2 arrives at the inlets of pipe_air[3], the routine `GetMixedOutPlug` is called which mixes the incoming amount of humid air with the resident air and calculates the new thermodynamical equilibrium

[3] The water in-and outlets are not in use yet in the ventilation types. If condensation occurs within one unit, the water is balanced and stored within this unit.

state. If condensation occurs, an iteration is necessary to determine the new amounts of steam and water and the new temperature (see also section 5.6.4). The outgoing plug is taken from the room volume.

5.3.2 Pump_air

Operation

Pump_air is essentially a ventilator, which creates a mass stream of dry air. In PAR1, the maximal mass stream `mp_max` is set in [kg/h]. PAR2 and PAR3 serve to calculate the electrical power consumed by the ventilator. With PAR4, the mode of controlling the pump can be selected: In mode 0, the control signal `ctr=0..1` is read off INP6 and the current mass stream of dry air is given by `ctr` · `mp_max`. In mode 1, the mass stream of dry air is read directly off INP7. With PAR7, the thermal mass of the ventilator can be modeled, see next paragraph for details.

Modeling

Just like the pump in fluid systems, the `pump_air` unit of a ventilation cycle is called twice in each time_step: Once as first unit of the cycle to set the mass stream for all other units, and a second time after all other units to balance masses and energies sent out and received back and calculate the new `pump_air` temperature.

In the first call, the outgoing mass stream of dry air is determined by the control signals (INP6 or INP7), whereas the mass streams of steam and CO_2 follow from the absolute humidity and the CO_2 -concentration inside the pump volume. This volume is calculated from the capacity for dry air given in PAR7[4]. For the inital time step, a steam content of the pump corresponding to a relative humidity of 50 % and a CO_2 -concentration of 500 ppm is assumed. Afterwards, the steam and CO_2 contents during the first call are read from the old values of OUT24 and OUT22, where they were stored during the second call of `pump_air` at the last time step. There is no water content since the possibility of condensation within the pumpt's volume is neglected.

The temperature of the outgoing plug is that of the air in the pump's volume. It is given by the old value of DERIV24, where it was stored during the second call of the last time step.

In the second call of `pump_air`, its new temperature is calculated from the total enthalpy contained in `pump_air's` dry air and steam content and the outgoing and incoming energy flow. The new temperature is stored in DERIV24. Also, the new steam and CO_2 contents are calculated and stored in OUT24 and OUT22, respectively.

[4]Note that this capacity for dry air can be chosen considerably higher than the pumpt's volume would allow, in order to make up for other system capacites which are not taken into account. On the other hand, its minimal value is given by one plug of dry air. If PAR7 is smaller than this minimal value, it is corrected automatically.

5.3.3 Heater_air

Operation

The `heater_air` is used primarily to heat up air, but can also be used to cool it down or to change its humidity or carbon dioxide content. With PAR2, five modes of operation can be chosen:

0. In this mode, there is no set temperature, but a certain heating power (INP8) can be applied to the air passing through the `heater_air`. With PAR4, the air temperature can be limited, mostly to 55 C, if the system is used to simulate a building with air heating. The steam content of the air is left unchanged, but its relative humidity may change of course.

1. Here, the air is warmed up or cooled down to a set temperature (INP6). Again, the absolute humidity of the air is not changed.

2. In this mode, only the relative humidity of the air is changed according to value $\varphi_{set} = 0..1$ given in INP7. The temperature is kept constant.

3. Here, the temperature and the relative humidity can be chosen (INP6,7).

4. In this mode, the temperature as well as the relative humidity and the carbon dioxide volume concentration [ppm] can be chosen via INP6,7,9.

Note that in contrast to the heater for fluid systems, the heating power is not limited here. However, a limitation can be set in the controller unit (see section 5.5.2). With PAR6 it can be chosen whether the set humidity from INP7 is interpreted to be in the range 0..100 (for PAR6=1) or 0..1 (other). The reason for this feature is that the heater is often used to simulate the ambient: In ColSim, only closed hydraulic or ventilation cycles are allowed, therefore the warm air leaving the building through the ventilation is sent through a heater which impresses ambient temperature, ambient relative humidity and ambient CO_2 content on it. For this prupose, INP6,7,9 of `heater_air` are directly connected with the `weather` unit (see section 5.5.5), which retrieves its information from a weather data base file. And in some of these data bases, relative humidity is given from 0..1, in others in %.

Modeling

The heater_air is essentially as simple as the heater in fluid systems, i.e. is has no internal capacities and cannot simulate thermal inertia and energy losses to the ambient. However, since there are four mass flows (dry air, steam, water and CO_2), some thermodynamical calculations are necessary, using the functions contained in `thermodynamics.c`.

If condensation occurs, the steam content of the outgoing air is limited corresponding to 100 % relative humidity, and the spare water is stored in the heater. If water must be evaporated to rise the humidity of the outgoing air, stored water is used if

possible. Otherwise, water is added to the heater at ambient temperature if INP6 is connected, assuming that the heater simulates the ambient. If INP6 is not connected, a temperature of the added water of 10 C is assumed.

5.3.4 Heatexchanger_air

Operation

The type `heatexchanger_air` is a very simple model of a mechanism to transfer heat from warm air leaving a building to fresh air entering it, i.e. to simulate a heat recovery device. At INP1, the heat flow of the incoming fresh air must be connected. OUT1 is the heat flow outlet of the warmed up fresh air, i.e. INP1 and OUT1 belong to the "cold" side of the heat exchanger. However, the "hot" side is not modeled in detail, but only the information about the temperature of the warm air from the building (INP6) and relative humidity (INP7) is needed.

In PAR1, the efficiency of the heat exchange must be given as a number between 0 and 1, e.g. 0.7 - 0.85. PAR2 must be set to 1, since at the current time only a counterflow heat exchanger can be modeled.

With PAR3, a bypass temperature can be set: If the temperature of the warm room air (INP6) used to heat up the fresh air is larger than PAR3, no heat transfer is made. This feature is necessary if the buildingis getting too warm e.g. at sunny days.

With PAR4, the limiting ambient temperature can be given where the heating is turned off. So if the temperature of the incoming fresh air exceeds PAR4, the heat exchanger is bypassed also.

Modeling

The modeling is as simple as that of `heatexchanger_static.c` for the fluid systems and based upon the same formula (see section 5.2.6). However, here the effectiveness is given directly and has not to be calculated from the heat transfer coefficient-area product like in `heatexchanger.c`. The device has no internal capacity and no heat losses to the ambient. Effects of condensation are not taken into account yet.

5.3.5 Diverter_air

`Diverter_air` is used to divide a stream of air into two, according to a control signal ctr received from INP6. The outgoing mass stream at branch 1 is ctr ·mp_in, with the incoming mass flow mp_in. Consequently, the outgoing mass stream at branch 2 is $(1-ctr)$·mp_in. This is valid for dry air, steam, water as well as and CO_2, and also for the heat flow.

The parameters are only needed for the hydraulic calculation, which is not active yet.

5.3.6 Mixer_air

The `mixer_air` is the counterpart of `diverter_air`: For every diverter, there has to be a mixer reunifying the mass and energy flows. However, in mixer_air there are more calculations to be done, because of the possibility of condensation inside the mixer. If the condensation point is approached, an internal iteration is performed in order to determine the new thermodynamical equilibrium of the mixed state, i.e. the new temperature and the masses of steam and water (compare section 5.3.1). The mixer stores the condensed water and tries to evaporate it again in the next time step.

No parameters have to be set. Note that OUT9 gives ctr_{out}, the relative contributions of the two incoming branches to the outgoing mass flow: $ctr_{out} = 1 - mp_2/(mp_1 + mp_2)$, where mp_1 and mp_2denote the incoming mass flows.

5.4 Building types

5.4.1 Wall_pcm

This type is subject to larger modifications at the current time.

5.4.2 AirRad_Node

This type is subject to larger modifications at the current time.

5.4.3 Mass element

5.5 General types

5.5.1 Sim control

A `sim control` unit must be present in every system to be simultated, since it sets start and end of simulation; In PAR1, the start date must be given in the format `yymmdd`, where yy is the year, mm the month and dd the day. Likewise, PAR2 denotes the last day to be simulated. Note that these dates refer to the weather data base, i.e. in the chosen data base (see section 5.5.5), the data files for this time span must be availlable.

Note that if the time span to be simulated crosses the turn of the year, the simulation carries on with january, the first, etc., but the year is not increased by 1. This makes sense, since the weather data bases mostly contain only data for one year. To simulate e.g. a heating period, the combination of e.g. PAR1=911001 and PAR2=910430 is possible.

With PAR3 and PAR4, the start time and end time can be set, respectively. The format is `hhmmss` (hours, minutes and seconds).

With PAR5, the maximal error in the system energy balance can be set.

PAR7 gives the simulation time step h in seconds. Note that if it is chosen too large, some units may report warnings refering to stability problems: Since in ColSim,

the Euler method is applied in order to solve the differential equation set describing the status of the system, the iteration is only stable if the physical time constants involved are much larger than the simulation time step h. If this is not the case, warnings or error messages are printed and the program execution may even be halted. Then, PAR7 should be reduced considerably in the next run. There's no use in trying to keep it as close to the instability limit as possible since then the simulation results will most certainly depend upon the simulation time step which is unphysical.

If PAR8>0, the fast iteration mode is activated and PAR8 is interpreted as the number of time steps which are skipped if possible. See section 6.4.2for details.

Note that the `sim control` type is not a real type in the sense that it is represented by an own source code file, but it is part of the main program `main.c`: The main program, knowing the type number of `sim control`, reads in its parameters and uses them to control the simulation.

5.5.2 Controller

Controller units are used to trigger pumps, ventilators, diverters, heaters and heatings in order to achieve e.g. a given set temperature in the storage or a certain air quality and temperature in a building. Since ColSim was mainly developped to serve as a simulation environment for developping and testing controlling algorithms, there are many variants of the controller type:

The default controller for fluid systems is `./src/more_types/controller.c` and it can control the pump of the solar collector cycle or alternatively the mixer of a heating cycle.

With PAR1, the mode is selected:

1. On&Off: At OUT1, a control signal for the pump in the solar cycle is sent. The signal is either 0 or 1 (i.e. the pump is turned off or on) and is calculated in the following way: If the pump was off, and the collector temperature (INP1) is larger than the storage temperature (INP2) plus a hysteresis term (PAR2), the pump is switched on. If the pump was on and the collector temperature (INP1) is smaller than the storage temperature (INP2) plus another hysteresis term (PAR3), the pump is switched off. Also, if the temperature at the storage's top (INP4) exceeds a given maximum (PAR7) or if the collector temperature exceeds 110 C, the pump is turned off.

2. Unisol: This mode describes a special storage and is not for general use.

3. MatchedFlow: Like in mode1, a control signal for the pump in the solar cycle is sent out at OUT1, but now it can be variable between 0 and 1.There are three MatchedFlow modes:

   (a) If PAR5 is larger than 20, it is interpreted as a set temperature for the collector. Then, the controller tries to keep this collector temperature constant if possible:

   If the pump was on and the collector temperature (INP1) is larger than the storage temperature (INP2) plus a hysteresis term (PAR3), the new control signal is calculated from a kI-controlling algorithm: $ctr = kP \cdot dT + kI \cdot integr_{dT}$, where the constants kP and kI are given by PAR4 and PAR6, respectively. The term dT is the difference of the collector temperature and the set temperature (PAR5). The term $integr_{dT}$ is an integral of dT over all past time steps, with a limitation to 2000. Here, the pump is turned off if the collector temperature (INP1) is lower than the storage temperature (INP2) plus a hysteresis term (PAR3).The control signal is limited to a minimum value given by PAR8.
   If the pump was off and the collector temperatur is higher than the storage temperature (INP2) plus a hysteresis term (PAR2), the pump is turned on at minimal flow (PAR8).
   Else, the control signal is 0. The pump is also turned off if the temperature at the storage's top (INP4) exceeds PAR7, additionally if the collector exceeds that PAR7 by 30 C.

   (b) If PAR5 is smaller than 20, it is interpreted as a set temperature difference between the collector and the storage temperature. Then, the controller tries to keep this temperature difference constant. So the term dT (see case (a)) is calculated as collector temperature (INP1) minus storage temperature (INP2) minus PAR5. Otherwise, the control signal is calculated just like in case a).

   (c) If PAR5=-1, everything's just like in case a) or b), except that not the kI-controlling algorithm is used to calculate the control signal, but the formula $ctr = (15 + 0.04 \cdot I_{glob,tilted} - 0.188 \cdot T_{in})/50$. This fit formula is a result of a static optimization of the mass flow as a function of the global irradiance onto the collector plane and the temperature of the fluid entering the collector, which is identified with the storage's temperature (INP2) here.

4. All other modes are for heating control, i.e. control values for the heating pump and the mixer in the heating circuit are calculated. However, these modes haven't been in use for quite a long time and need revision, therefore we skip them here.

There is another controller for ventilation and heating control in the folder `./projects/zone1`. It can control the ventilation by using the CO_2concentration as indicator for the quality of the room air and also is able to control the temperatures of several rooms by calculating a heating power, depending on the actual room temperatur in comparison to the set temperature. Again, with PAR1, the mode is selected:

1. Ventilation and heating control: In this mode, a control signal 0..1 for the unit `pump_air` (OUT1) and up to six heating powers (OUT9...OUT14) are calculated, depending on how many subsequent temperature signals (INP1...INP6) are connected and accompanied by non-vanishing set temperatures (PAR12 and PAR15-PAR19). A kP-control (which is a kI-control with vanishing integral term, see

explanation of kI-control in mode3a of default controller) is used for the ventilation, where the value for kP is given in PAR11, the set CO_2 value in PAR13 and the measured CO_2 is read from INP7. If the CO_2 is lower than a lower limit given in PAR14, the control signal for the ventilation is set to 0. If PAR9=1, the control signal stays constant, independent of the CO_2 value. This constant value can be set in PAR10. For the heating, a kI-control is used, where the parameters kP and kI must be given in PAR4 and PAR6, respectively. PAR5 contains the maximal total heating rate.

2. Fuzzy controller for ventilation and heating control: Here, a fuzzy algorithm is used for the same tasks as in mode1. Note that to use this mode, the right fuzzy settings (include files called `settings_heat.h` and `settings_air.h`) must be present in `./projects/zone1`. The same in-and outputs are used as in mode 1 and also PAR5, PAR12 and PAR15-18 have the same meaning. With PAR3, a value between 0 and 1 can be selected for λ, the fuzzy ventilation control parameter: λ=0 causes the maximal economic ventilation program with the lowest mean ventilation rate and λ=1 causes the maximal comfortable ventilation program with the highest mean ventilation rate. With PAR4, the cooling mode can be turned on, i.e. the ventilation is enhanced if the temperature specified by PAR6 exceeds PAR10. With PAR7, the inertia of the heating control signal can be varied (mostly set to something like 0.7 %). With PAR8 and PAR9, the amount by which the fuzzy sets for CO_2 and relative humidity are shifted for λ<1 is specified. The shifting is proportional to (1-λ). PAR14 specifies the CO_2 concentration in the ambient since only the difference of the measured CO_2 concentration in the room to that in the ambient is used to calculated the ventilation signal. With PAR2, the air heating mode can be turned on, which means that the ventilation is enhanced if the room temperature does not reach the set value: Since with the air heating, the air temperature is always limited to 50-55 C, the ventilation must be sufficient to transport enough heating power into the room. If PAR2=1, INP1 is interpreted as the measured temperature and PAR12 as the set temperature for the air heating.

5. This mode is the same as mode1 but without the ventilation control.

7. This mode equals mode1 again, but this time without the heating control.

5.5.3 Gnuplotter

With the help of the gnuplotter unit, the state of the system can be observed during a simulation run. Any quantity which corresponds to an output box in the xfig-picture of the simulation dek can be connected with one of the gnuplotter's input boxes and will be drawn then as a function of time during the simulation run. There are two exceptions; the mass and heat flow outlets cannot be connected with the gnuplotter (or printer), since this would be interpreted as mass and heat flow into the plotting

devices and definitely cause severe damage... Sometimes, there are special outlet ports where information about the current mass or heat flow can be obtained.

On the other hand, not only those quantities can be plotted which already have visible output boxes in the xfig-picture, but all those which are listed as outputs in the info-edit window of a unit (i.e. which appear in the comment part of the source codes of the corresponding type). In section 4.3.6 it is explained how the output boxes needed can be added to the xfig-picture.

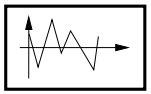
The gnuplotter unit, using the Public Domain program "Gnuplot", draws two diagrams: Plot 0 for INPUT 1..5 and plot 1 for INPUT 6..10. The inital values of INP1..10 define the scaling factors used for the plot. These scaling factor are shown in the legends of the plot, together with the unit number and outbox number which is plotted. In Plot 0, also the names of the plotted output quantities are printed in the legends. This information is taken from the comments to the output variables of the specific unit. These comments can be edited with the help of the **info_edit** window.

Note that INP1..5 and INP6..10 must be connected subsequently, since for the sake of speeding up the simulation, the program stops the output of data to the plot file when it encounters an unconnected input.

With PAR2, the "refresh time interval" can be chosen: The plot is drawn completely new after every refresh time interval. Since the simulation continues meanwhile, the lines drawn become longer and longer after every interval up to the chosen maximal time span (PAR1), where a new plot is started. In this way, it's easier to follow the progress of the simulation, compared to the alternative that the lines are drawn at once for the whole maximal time span (PAR1=PAR2). However, the latter parameter choice makes the simulation run considerably faster. The fasted run can be achieved by turning off the gnuplot output completely by setting PAR7=0.

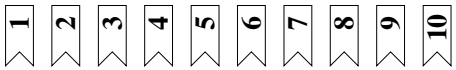
If PAR8=1, the plot ranges are chosen automatically and adapt themselves continuously to the data. If PAR8=0, the ranges have to be set with PAR3..6. If PAR9=1, the line "`continue ?? no:echo n>no, yes:echo y>no, cancel:echo c>no`" is plotted for every day during the simulation run. Its meaning is that the simulation can be terminated in a controlled way without the menu by typing `echo n>no` in a text window. By this, the letter `n` is written into the file `no`, which is checked by the main program after each simulated day.

The plot file is called „plot.dat". It contains data from the current day, which can also be plotted with an external call of gnuplot: After typing "gnuplot" in some text window, type e.g. `plot "plot.dat" using 1:2`. This command plots the second column of the data file as a function of the first column. The command `plot "plot.dat" using 1:3 with lines` will plot the third column as a function of the first column, with the data points interconnected by lines. For more information on gnuplot try the online help by typing `help`.



INFO: GNUPLOTTER

UNIT: 14



5.5.4 Equation

The `equation` type can be used to apply an arbitrary formula to several input values and receive the results as output values. However, for defining this formula, the source code file `equation.c` has to be edited. The best approach is to copy the file from `./src/more_types` into the folder of the current project. Then, in the `.cfg`-file of the project, a link has to be set to this file, otherwise the original source code in `./src/more_types` is read. See section 7.1 for details..

Now, the local file `equation.c` in the projects folder can be edited: It contains a `switch/case` structure, where PAR1 is the switch used to decide which case is active. Now either use an existing case and modifiy the commands there or insert a new case between the last current case and the `default` case. If `n` is the number of the current last case, the new case has to look like this

`case n+1:{variable definitions; commands; setting the OUT variables; break;}`.

The input variables are in `input[i]`, the parameters can be addressed by `PAR (i)` and the output variables by OUT (i), where the latter two expressions are macros defined in `./src/global.h`.

PAR (1) must be set to the case number of the formula to be applied. The other parameters can be used for defining the formula. If more INP or OUT boxes are needed than there are in the xfig picture of the `equation` unit, simply add those you need by following the instructions in section 4.3.6.

Note that after editing `equation.c`, you have to use the **make** button in the ColSim menu to activate the changes.

5.5.5 Weather

The `weather` unit reads data from the weather data in `./weather` and sends it to its OUT variables, where it can be used by other units which need information about the solar irradiation, the ambient temperature and humidity, etc. However, note that in principle all kind of data can be read by the `weather` unit, i.e. it may also be used as a substitute for the `load_profile` unit e.g. if a different load profile should be defined for every day of the year.

Currently, the weather unit expects to find subfolders in `./weather` whose names are composed of `try` and a two digit number, e.g. `try05` or `try`14.[5] The number is

[5] Try is an abbreviation for "test reference year". See ? for information about this weather data base.

specified in PAR5 of the `weather` unit. Inside these subfolders[6], a weather data file for each day of the year (or at least for all days which will be simulated) should be present, and its name must look like 970220.dat, i.e. it contains the number of the year, the month and the day.

Note that the original `try` data does not look like that, rather it is a single file for all days of the year, containing a header with information about the location where the data was recorded. Moreover, there are many more data columns than in the ColSim `try` data files. The original files are useful for the information about the locations (see explanations to PAR3 and PAR4). But if you only have these original files and no data in the ColSim format like explained above, you can use the convertion programm `try_convert.awk` (see comment part at the top of the file for how to use it). Normally, it is contained in a `try` subfolder, otherwise contact your suppliers of ColSim.

In PAR1, the timezone (with respect to Greenwich, east positive) where the weather data was measured should be given. E.g. Germany is east of Greenwich, therefore PAR1=+1 hour. PAR2 contains the time difference between two lines of the data file. In PAR3, the geographics length of the measurement place with respect to Greenwich must be given, but here east is counted negative. The geographical latitude must be given in PAR4.

With PAR6, it is determined whether the data is interpolated or not. If PAR7 is set to 1, a warning is written if the time difference between one data line and the next does not equal PAR2.

The OUT variables of the weather unit correspond to the data columns of the files in the `try` subfolders, i.e. if the ambient temperature is the first column after the time column, it can be read off OUT1 of the `weather` unit.

5.5.6 Printer

The `printer` unit writes the data it receives through its inputs to the file `sim_outn.dat`in the ColSim directory. The number `n` is given by PAR1 of the `printer`. PAR2 is only read if PAR3=0, then it gives the print interval in seconds, i.e. the time difference between two subsequent lines in the output data file. Normally PAR3 is used to define the print interval and there are 8 possibilities to choose from.

With PAR4, the format of the time variable in the data files can be chosen: If PAR4=1, the time is given with a twelve digit number, where year, month, day, hour, minute and second are each represented with two digits. For PAR4=1,2,3,4 the time is given in seconds, hours, days or minutes, respectively, where the counting always starts at the beginning of the year.

If PAR5=0, the data is not modified, but if it is set to 1, it is divided by 1000 and 3600, which can be used to transform an energy output in J or Ws into kWh. However, this mode is only active if the data is integrated or summed up. This can be achieved by setting the input inital values:

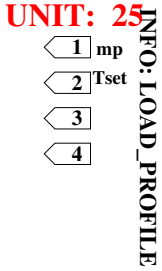
[6] Note that the try-subfolders can also be links. A link can be defined with the help of the `.cfg`-file of the project by making an entry e.g. for `try05` instead of a type name. Note that the last entry in the new line of the `.cfg`- file must be KERNEL then, not TYPE.

If the input inital value of INP1 is 0, the variable connected to INP1 is averaged over one print interval (and printed into the first column after the time variable). For an inital value of 1, the momentary value at the time where the dataline is written is used. For a value of 2, the variable is integrated over one print interval, for 3, it is summed up over one print interval.

5.5.7 Load profile

Originally, the unit `load profile` was designed to model human behaviour concerning the consumption of warm water. The unit reads a data file, `load.SystemName.dat` in the folder `./projects/SystemName`, which is linked to `load.dat` in the ColSim directory. In this file, there is a entry for every change in the mass flow and the temperature of the demanded warm water. like table 5.1 shows. The data read from the file is then sent to the outlets of the `load profile` unit and hereby made available for other units, e.g. the mass flow can be sent to a pump and the set temperature to a control unit which controls a mixer mixing hot and cold water.

However, `load profile` can be used in a much wider context now, namely to imprint almost any input quantity onto the system. E.g. the reaction of hydraulic devices like storage or pipes to sudden changes in the ambient temperature can be tested by connecting the ambient temperature input not to the weather unit but to the `load profile`, where a designed temperature course is read from a data file. Also, inhabitants of a building can be simulated with the help of the `load profile` by making entries for the water steam, CO_2 and heat emissions by the people, as well as the indirect loads like steam production by plants and wet clothes and heat emissions by electrical devices.



In table 5.1, the structure for a simple version of the data file is shown. The maximum number of columns after the time column is OUTMAX0, a number globally defined in the file `./src/global.h`. The maximum number of lines is LOAD_SIZE, also defined in `./src/global.h`.[7] In the first line, the names of the quantities listed below should be given (there must be as many words in the first line as data columns). The second line stays empty. Beginning from the third line, the data entries follow.

[7]Note that to change these values, the file `./src/global.orig.h` must be edited since `global.h` is created anew during every configuration process. See section 6.2.

Table 5.1: Simple data file for load profile.

time	mp	Tset
000000	0	10
063000	120.0	40
064000	0	10
130000	85.0	50
130500	0	10
180000	150.0	40
182000	0	10

The format of the numbers is not important except for the first entry in each line: the time. It must be given either with 6 digits or with 7 digits, where the first two or three digits, respectively, count the hours, the following two the minutes and the last two the seconds. So e.g. 103000 means half past 10 on the first day.

The total time span defined with the data file is read off PAR1. It is considered as cyclic, i.e. when one time span has elapsed, the data file is read from the beginning to give the input for the next time span.

Note that there are only entries in `load.dat` when something changes, here temperature or mass flow. The values given after the time variable are used beginning from that time until the time of the next entry.

The load profile unit can have inlet ports (not shown in the xfig picture), though they cannot be connected to other units. Rather, the input variables always stay at their input inital values, which can be used to divide between working days and weekend: If the input initial value of INPn is set to 1, the corresponding output value OUTn is set to 0 on weekends. Whether or not it's a weekend is decided by the main program which knows beginning and end date of the simulation.

Note that each outlet of `load profile` corresponds to a data column of the data file, where the correspondance is one-to-one with the exception that OUT6,7,8 are left out since they are reserved for the energy balance. So OUT5 presents the fifth column of `load.dat`, and OUT9 the sixth column.

5.6 Type related kernel routines

5.6.1 Radiation processor

The function `rad_processor` (contained in the source code file `rad_processor.c`) is called from within types which balance solar energy, i.e. the solar collector and the wall type. It is used to calculate Idt and Ibt, the diffuse and beam radiation incident onto a tilted plane from Igh and Idh, the global and diffuse irradiation onto the horizontal plane. The collector as well as the wall type call the radiation processor and provide it with information about Igh and Idh, the albedo of the ambient, the current time,

the location and the slope and orientation of the absorbing surface.

The first thing the radiation processor does is to calculate the solartime, the hour angle and the declination. From this, the incidence angle ϑ of the beam radiation onto the absorbing plane is calculated (the angle between the beam and the normal to the plane). With ϑ_z as the incidence angle of the beam radiation onto the horizontal plane, Ibt is given by $Ibh \cdot \cos(\vartheta)/\cos(\vartheta_z)$. If the sun is low, a correction is applied since the term $\cos(\vartheta_z)$ diverges for $\vartheta_z = 90°$. Though Ibh is very small then, it does not vanish completely for $\vartheta_z = 90°$but still is visible due to the diffraction in the atmosphere.

Then, Idt is calculated from $Idh \cdot 0.5 \cdot (1 + \cos(\beta))$, where β is the slope of the surface (0 for horizontal surface). Another contribution to Igt, the global irradiation onto the horizontal plane, is Irt, the irratiation reflected back from the ground. It is given by $Igh \cdot 0.5 \cdot (1 - \cos(\beta) \cdot \rho_g$, where ρ_g is the albedo of the ambient. All formulas origin from [1].

If a special horizon is defined, i.e. if PAR15 of collector or wall is set to 1, Idt and Irt are recalculated by the function `horizon` (see section 5.6.2). Also, a shading value is calculated which is 1 or 0 and either extinguishes the beam radiation totally or leaves it unchanged.

In the new version of the `radiation processor`, also the radiation which is transmitted through a glazing is calculated. From the collector or the walls, the value b_0 is delivered which characterizes the angular depencence of the transmittance. It is used during the call of the function `Incidence angle modifier` (see section 5.6.3) which returns multiplicative factors for the reduction of beam and diffuse radiation by the glazing. Note, however, that these values still have to be multiplied with the transmission factor at vanishing incidence angle.

The `radiation processor` returns all the calculated information about the irradiation onto the tilted surface and in addition also the angles describing the position of the sun, absolute and relative to the surface.

5.6.2 Horizon

5.6.3 Incidence angle modifier

5.6.4 Thermodynamics

The source code file `thermodynamics.c` contains several functions describing thermodynamical relations. There is e.g. the function `Get_Temp_from_enthalpy` which is used to calculate the temperature in a volume from the enthalpy and the masses of dry air, water steam and water contained in it. Some functions also contain fit formulas, e.g. `Get_p_steam_s` which calculates the partial pressure of water steam in saturated air as a function of temperature. This formula is valid between 0 and 100 C (see e.g. [2] p.103f).

The maybe most important function of `thermodynamics.c` is called `GetMixedOutPlug`: It calculates the mixing of an entering plug of humid air into the air volume corresponding to one node, and determines temperature and composition (dry air/steam) of the

outgoing plug. Here we give a short outline of the calculation:

First, the enthalpy of the plug is added to that of the node to give H_{new}, likewise its masses of dry air and steam are added to those of the node, giving $m_{dry,mix}$ and $m_{steam,mix}$. The resident water of the node was temporarily added to $m_{steam,mix}$. Then, the absolute humidity of the mixture, x_{mix}, is calculated from $x_{mix} = m_{steam,mix}/m_{dry,mix}$. Now, there are two possibilities:

1. $x_{mix} < x_s$, where x_s is the absolute humidity of the saturated state, calculated from the function `Get_xs_from_enthalpy2`(which depends upon the enthalpy of the humid air, divided by the mass of dry air). This is the easy case since there is no condensation, therefore no water (in liquid form) is present. The new temperature simply follows from the function `Get_Temp_from_enthalpy`, since all masses are known (the mass of dry air is always preserved, the mass of steam is x_{mix}times the mass of dry air and the mass of water vanishes).

2. $x_{mix} > x_s$. Now there is a problem: We know that condensation occurs during the mixing and there is an unknown amount of water in the system. We know that x_{mix} equals x_s then, but the problem is that the x_s we calculated is most probably wrong: We used the function `Get_xs_from_enthalpy2` with the argument $H_{new}/m_{dry,mix}$, but we know now that H_{new} also contains the enthalpy of water and `Get_xs_from_enthalpy2` expects only the enthalpy of humid air (i.e. dry air and steam) as an argument.
   But since the new temperature is unknown also, we can't use the functions `Get_x(p_steam)` and `Get_p_steam_s(T)` in combination either to calculate the new x_s. On the other hand, the new temperature can only be calculated if the masses of steam and water are known, and for this we need the right x_s of the equilibrium state.
   Therefore, an iteration is necessary to determine the new mass relations and temperature simultaneously: We start with the x_s like calculated above and use it to calculate temporary values for $m_{steam,node} = x_s \cdot m_{dry,mix}$ and $m_{water,node} = m_{steam,mix} - m_{steam,node}$. From these masses, a temporary temperature T_{mix} can be calculated with `Get_Temp_from_enthalpy`,since the total enthalpy H_{new} is constant and known. From the new temperature, the enthalpy of the humid air can be calculated from $H_{new} - m_{water,node} \cdot T_{mix} \cdot cp_{water}$, with cp_{water}as the specific heat capacity of water. Now we have a better input value for `Get_xs_from_enthalpy2` and can calculate a closer guess for x_s, etc. The iteration is stopped as soon as the temperature changes by less than 0.0001 C.

Now the new equilibrium state is determined and the properties of the outgoing plug can be calculated.

In thermodynamics.c there is also the function `Get_alpha_air` which calculates the heat transfer of the air streaming in a pipe to the walls of the pipe.

5.6.5 Fuzzy routines

Chapter 6

Structure of ColSim

In this chapter, it is explained how ColSim works from the configuration to the individual steps in the simulation run. It's not absolutely necessary to read it if one uses the menu based configuration and simulation and only works with the standard systems (SchichtSpeicherSystem, StandardKollektorAnlage, etc.) with modified parameters and slight rearrangements of types.

However, for people who have to use the command line execution of system configuration and simulation or who intend to modify parts of the source code or design completely new systems, this chapter is a "must".

6.1 Installation

The installation command `INSTALL`, which is executed directly after unpacking the ColSim programs, is a simple shell script. Its tasks are:

1. Setting a link from the file `.xfig` in the home directory to `.xfig` in the ColSim directory. The file `.xfig` is created by the graphic program Xfig and is used for temporarily storing graphical objects.
2. Compiling the source code file `config.c` into the executable program `config` which is important in the system configuration.
3. Creating the gnuplot data folders which are used by the fuzzy controller.
4. Compiling the source codes in the folder `./cnv` and creating the executable program `cnv.exe` (called **fig2dek** in the menu) which is used to convert the graphical simulation input into the more compact form used by the simulation program itself.

6.2 System configuration

6.2.1 Tasks of the shell script `ColSim`

In the following, the processes started by the call `ColSim Systemname` are explained in detail, again with the help of a test system for demonstration.

1. The first command in the shell script `ColSim` concerns the definition of the path to the ColSim directory, which depends upon the location where ColSim was installed.
2. Then, the ColSim customized xfig program is searched and, if found, established as the standard xfig which is invoked via the **xfig** button in the ColSim menu.
3. Now it is checked whether `Systemname` corresponds to an existing project. If not, the `project_organizer` is invoked and a project can be selected from it with a double click.
4. The chosen project is configured with the help of the program `config`. We'll come back to this in the next section.
5. All source code files selected during the configuratino process are compiled with the help of the makefile in `./src` and the executable program `sim` is created in the ColSim directory.
6. A link is set from `load.dat` in the ColSim directory (needed by the type `load profile`, see section 5.5.7) to the file `load.Systemname.dat` in the folder `./projects/Systemname`.
7. Links are set from `sim.fig` in `./cnv` to `Systemname.fig` in `./projects/Systemname`, and from `sim.dek` in the ColSim directory to `Systemname.dek`, also in `./projects/Systemname`.
8. The ColSim menu is invoked.

6.2.2 The purpose of the routine `config`

The advantage of the special configuration method described in the following, is that it is highly flexible: The source code file which is used to describe a type in the system can be exchanged by an alternative one very easily. In this way it is possible to modify the source code files and test the results by working with local copies in the projects folder, leaving the original files unchanged. Then, the modifications will only affect the current project and not all other ColSim projects, too.

So the main task of the configuration routine `config` is to put together the right source code files needed for the simulation of the current system and to prepare the right auxiliary files. By the xfig graphics file, the types appearing in the system are determined, but there can be different source code files describing the same type: The pipe e.g. can be described with `pipe1.c`, `pipe2.c` or `pipe3.c` in the folder `./src/more_types`, which differ in the way the physical processes in the pipe are modeled. In the file `./cfg/default.cfg`, the default settings are given. Take a look at it with `less ./cfg/default.cfg` to study its structure:

- The first column contains the name of a source code file or link corresponding to the type of the same name, e.g. `storage.c`. Note that actually the types are not identified by their names but by a type number contained as well in the xfig picture of the types as in the comment part of the source code.

- The second column contains the name of the file linked with the name in the first column. E.g. the storage is linked with `storage4.c` in the folder `./src/more_types` which is the default setting. An alternative would be `storage3.c` in the same folder.

- The third column of the `config` file characterizes the entry as `TYPE` or `KERNEL`. Like explained earlier, a type corresponds to a unit in the xfig graphics of a system and to the source code file which is called for this unit at every time step of the simulation run. On the contrary, files characterized by "KERNEL" are either called by the main program or from within type related programs. An example for the latter case is the `radiation processor` (see section 5.6.1).

When the configuration process is started for a system, the file `./cfg/default.cfg` is read first. Then, the config file of the system - `Systemname.cfg` - is read, which must be present in the folder `./projects/SystemName`. Here, all non-standard configurations are listed. This may be replacements of standard versions of the source codes with non-standard versions or completely new developed programs describing user-defined types. In addition, the use of weather data sets other than the standard one in `./weather` can be defined (see section 5.5.5).
The determined configuration is written to the file `running_config.cfg` in `./cfg`, if something has changed with respect to the old `running_config.cfg` file. Then, the links listed in the `.cfg-files` are set in the folder `./src`, e.g. `./src/storage.c` becomes a link to `./src/more_types/storage4.c` if nothing else was specified in `Systemname.cfg`.

Moreover, several files are created, mostly in the folder `./src`:

- A makefile which contains compilation instructions for all source codes needed by the system.

- The header file `unit_order_init.h` which lists a `PRIORITY` and a `HYDRAULIC` value for each type. The values are read off the source code comments of the individual types. When trying to implement newly developed types, both values must be set correctly (see section 6.4.1 for details). The PRIORITY value will be used by the program `unit_order_init.c` which, at the beginning of each simulation run, determines the order in which the units will be called in every time step. The `HYDRAULIC` variable describes the hydraulic function of the type.

- The header file `global.h` which contains definitions like the maximal number of units, parameters, out-and inlets, etc. For these values, the file `global.orig.h` is used as input. Moreover, the ColSim functions and their arguments are defined in global.h. The structure of the functions is read off their source codes.

- A program named `type_call.c`, which is called by the main program during the simulation run. In every time step, the main program runs through all units of the system according to a previously determined unit order (see section 6.4.1). For every unit, it calls the function `type_call`, which determines the type of this unit and calls the right function describing the type.

6.3 Conversion of the simulation script

In section 3.1.2, the conversion process was roughly explained already. The executable program which is started with the button **fig2dek** in the ColSim menu is called `cnv.exe` and can be found in the directory `./cnv`. The source codes are in `./cnv/src`. This program opens the text file `./cnv/sim.fig` (which is a link to `./projects/Systemname/Systemname.fig`). Take a look at `sim.fig` with the help of an editor or with `less sim.fig` in the folder `./cnv`.

Although the parameters are not visible in the xfig graphics of the system, they appear in this file. How the parameters are edited is explained in chapter 4. What `cnv.exe` does is to read the information about the system from `sim.fig` and to write it to `sim_new.dek` in more compact form:

- It reads in the type number and the name of each unit, which are given at the top of the parameter list and after the letters `INFO`, respectively.

- It reads in the unit's parameters found behind the letters `PAR` and the input inital values following the letters `INP`.

- It identifies the lines connecting each unit's in-and outlets to those of other units.

Moreover, it assigns unique, continuous unit numbers from 1 to n to the units. When the conversion is finished, two new files are present in `./cnv`, namely `sim_new.dek` and `sim_new.fig`. The latter is an exact copy of `sim.fig`, except for the updated unit numbers.

Then, take a look at `sim_new.dek`. For every unit of the system, all parameters are listed here, and also the interconnections with other units in the following way: The block which is titled "INPUTS" lists for every input port of the unit under consideration first the number of the unit, then the number of the outlet it is connected to. If an input is not connected, both entries are zeros. In this case, the input inital value, given in the block below, is assigned as a fixed value to this input.

There is one extra thing that **fig2dek** does apart from starting `cnv.exe`: It copies the file `./cnv/sim_new.dek` to `./projects/Example/Example.dek` and sets a link from `sim.dek` in the ColSim directory to the dek file in the projects folder. This step is not necessary when doing a command line execution of a simulation. It is sufficient to link `sim.dek` in the ColSim directory to `sim_new.dek` in the folder `./cnv` (see section 3.2.2).

6.4 The simulation run

6.4.1 Initialization procedure

When the simulation program is started by using the **sim** button in the menu or by typing `sim` in the ColSim directory, the main program is started (source code `main.c`, in the folder `./src` like all functions discussed in the following). After initializing variables and arrays, it calls as the first function `dek_reader.c`. The latter opens the simulation dek `./sim.dek` and reads in the information about how many units and which types are used in the system, their parameters and their interconnections. The parameters of the unit `sim control`, which must be present in every system, are used immediately for calculating beginning and end of the simulation time in seconds.

Then, the function `unit_order_init` (source code file `unit_order_init.c`) is called. It determines the order, in which the individual units will be called in every time step by creating an onedimensional array called `unit_order` which contains the unit numbers to be called subsequently. For this purpose, it uses the header file `unit_order_init.h` which was created during the configuration process. In this file, a `PRIORITY` number is assigned to each type, except for `sim control`. Types with low priority numbers are called first within one timestep in the following way:

1. The type `sim control` needs no `PRIORITY` number, since it is only called once at the beginning of the simulation, not in every time step. It sets beginning and end of the simulation and the lenght of one time step.

2. Both of the types `weather` and `load_profile` have `PRIORITY` 1. When `weather` is called it reads the weather and radiation data from the adequate file in the folder `./weather` and makes it accessible to other units. Similarly, the type `load_profile` reads the consumer load profile from the file `load.dat`.

3. The `equation` types has priority number 2.

4. The `controller` types have `PRIORITY` 3. On the basis of measured temperature values from the last time step, they set controller signals for pumps and diverters for the actual time step.

5. The first hydraulic type to be called in every time step is a pump. Pumps have `PRIORITY` 5, whether it's a fluid pump or a `pump_air`, i.e. a ventilator. The pump in a hydraulic cycle is always called before the other units, since it sets the mass flow for the whole cycle. As mentioned before, ColSim is based on plug flow modeling, therefore at the beginning of a time step the pump puts out a plug of fluid with mass $mp \cdot h$. Here, mp is the fluid mass per time being pumped through the cycle and h is the simulation time step. Every hydraulic unit succeeding the pump receives the plug and sends it on to the next unit within the cycle.

6. All hydraulic types (by which we also mean the ventilation types) except for pumps have `PRIORITY` 6. The hydraulic types of a specific cycle are called after the pump of this cycle according to the order in which they are passed by the fluid. Since at branching points, this criterion is not sufficient to determine a unique unit order, the function `init_set_mp` (source code file `init_set_mp.c`) is called from within `unit_order_init`. How this function works is discussed in detail in the next paragraph.

7. The pump is called a second time after the other units of its hydraulic cycle in order to compare the mass of the incoming plug to that of the plug sent out before (but in the same time step). If they are not equal, the program is stopped. Also, the thermal energy is balanced (also see section 5.2.4for details). After one hydraulic cycle is completed, the next pump and its cycle are calculated.

8. The `AirRad_node` has `PRIORITY` 10, so it is called after the hydraulic types, but before the wall types. Therefore, it receives the current room temperature from `pipe_air`, but the gains from the walls stem from the last time step.

9. The wall types have `PRIORITY 12`, i.e. they are called after the `AirRad_node`.

10. `Gnuplotter` and `printer`, with `PRIORITY` number 15, are called last since they don't influence the physical behaviour of the system.

In the header file `unit_order_init.h`, not only a `PRIORITY` number but also a `HYDRAULIC` number is assigned to each type. This number contains information about the hydraulic qualities of the type and is used by the function `init_set_mp` to determine the unit order within hydraulic cycles. In `init_set_mp`, one hydraulic cycle after the other is run through and the units contained in it are ordered in the following way by assigning a unit order number to each of them:

1. Fluid types with `HYDRAULIC` number 1 possess only one heat flow inlet and one heat flow outlet. Examples are the collector, the pipes, the heater, etc. All units with this `HYDRAULIC` number are simply ordered in the way they are passed by the fluid leaving the pump. The ventilation type `pipe_air` has `HYDRAULIC` number 32, the `heater_air` has number 50.

2. The pump carries `HYDRAULIC` number 3 (`pump_air`: 49), though it also has only one mass flow inlet and one outlet. However, the pump is a special hydraulic type, since the calculcation of a hydraulic cycle starts and ends at the cycle's pump, i.e. the pumps are called twice in every time step. Therefore, each pump appears twice in the list of unit numbers to be called subsequently at every time step.

3. The storage carries `HYDRAULIC` number 4. It possesses many heat flow in-and outlets, and the calculation of all cycles must be finished before the storage's system quantities can be updated. Therefore, the storage is called several times in one time step and its unit number will consequently appear several times in the array `unit_order`. When `init_set_mp` reaches a storage while running through

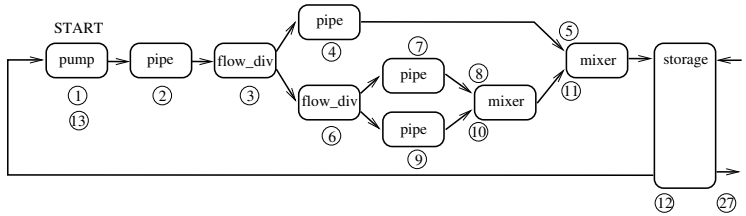


Figure 6.1: Unit ordering at branching points.

a hydraulic cycle, it continues with the unit after the storage which belongs to the same cycle[1]. When during the simulation run a hydraulic cycle connected to the storage is calculated, old temperature values from the last time step are used instead of the not yet availabe updated values. In each time step, only the last hydraulic cycle with connection to the storage receives updated values from the storage's outlets. However, due to its large heat capacity and large inherent time scale, this poses no problems.

4. The heat exchanger, which carries `HYDRAULIC` number 10 and has two heat flow inlets and two outlets, is treated in a different way: When a heat exchanger is encountered by `init_set_mp` for the first time, the program leaves the corresponding cycle and continues with the other cycles until the cycle is finished which is connected to the same heat exchanger. Then, it returns to the heat exchanger and finishes the unfinished cycle. Therefore, the heat exchanger is called three times in each time step. Note that `heatexchanger_air` (`HYDRAULIC` number 50) is a different case since it has only one heat flow inlet and one outlet.

5. The `diverter` has `HYDRAULIC` number 11 (`diverter_air`:47). Fig.6.1 illustrates what happens at such a branching point: The program `init_set_mp` first continues with an arbitrary unit succeeding the diverter. Several units later, a mixer must appear, since in ColSim only closed hydraulic cycles are allowed. If it is found that the mixer's output cannot be calculated since no current input values are at hand, the program returns to the diverter and continues with the second branch. Also branchings inside branchings are possible.

6. The `mixer` has `HYDRAULIC` number 13 (`mixer_air`: 48). Just like the `diverter`, it is always called two times within one time step.

7. All other types (i.e. with no heat flow in-and outlets) carry `HYDRAULIC` number 99.

[1] As an example: When the unit before the storage was connected to the storage via the auxiliary heater inlet, as the following unit the one is chosen which is connected to the storage via the auxiliary heater outlet.

The determined unit order can be viewed by starting the simulation with `sim -d | more`. Then, the unit numbers and the type names of all units are shown during the program execution. The array `select_unit` counts the calls of the storage within one time step. See also the source code of the main program, `main.c`, and check the use of `unit_order`.

6.4.2 Main loop

When the process of determining the right unit order is completed and the array `unit_order` is filled with all unit numbers to be called subsequently during one time step, the simulation starts. The main program calculates start and endtime of the simulation and prints out the actual day. In the main loop over all simulation time steps, a second loop is contained which runs through all entries in the array `unit_order`, i.e. it runs through all units in the system according to their previously determined order. Within this loop, the function `type_call.c` is called, which was generated during the configuration process. In `type_call`, first the type is identified which corresponds to the current unit number, then the function describing this type is called. To most of these functions, the following important variables and arrays are handed over which `type_call` itself receives from the main program:

1. The variable `unit` informs the function about the current unit number. Since the same type, e.g. a `pipe`, can appear several times in a system, the unit number is needed in order to identify the right parameters and to store the calculated state quantities of the unit.

2. The twodimensional array `par[n][i]`, where n is a unit number and i numbers the parameters of each unit. This array is filled at the beginning of the program run with the parameters read from `sim.dek`. Note that there is a macro defined in `./src/global.h: PAR(i)` means the same as `par[n][i]`.

3. The array `in_u[n][i]`, where i numbers the inlets of each unit. The array element `in_u[unit][k]` contains the number of that unit which is connected to the current unit via its inlet port number k.

4. The array `in_nr[n][i]`, where i numbers the inlets of each unit again, but now the array element `in_nr[unit][k]` contains the outlet number of that unit which is connected to the current unit via its inlet port number k.

5. The arrays `out_o[n][i]` and `out_n[n][i]`, where i numbers the outlets or output variables of each unit. For `out_n[n][i]`, the macro `OUT (i)` was defined. The array element `out_o[unit][k]` contains the value of the `k`th output variable of the current unit which was calculated at the last time step. Likewise, in `out_n` the newly calculated values are stored. Note that these arrays are used on the one hand to store at short notice system quantities for internal calculations, e.g. sometimes the old values are needed as a basis for the calculation of the new values. Moreover, in `out_n[n][i],i=6,7,8` the nth unit's energy input, its output

and its internal energy are stored, respectively, and used for the system energy balance at every time step.
On the other hand, the output variables are at the user's disposal for information about the system state: For each existing output number of a unit (different types have a different number of output variables, see chapter 5), an output box can be drawn in the xfig picture of the system. This box can be connected to a gnuplotter or a printer, by which information about the specific output variable can be visualized or stored (see chapter 4for instructions on such manipulations).

6. The temperature arrays `deriv_o[n][i]` and `deriv_n[n][i]`, where i numbers the nodes of each unit. Some types are divided into several nodes like e.g. the pipe, the storage and the collector, others consists of only one "node" or are not assigned a temperature at all (in this case, these fields are not handed over). The array element `deriv_o[unit][k]` contains the old temperature at node `k` of the current unit. Likewise, `in deriv_n` the newly calculated temperatures of node k will be stored.

7. The variable h contains the lenght of one simulation time step in seconds.

8. The array `init_energy[n][i]`, where i numbers the nodes of each unit. The array element init_energy[unit][k] contains the initial energy of the current unit's node number k. Since the fluid in the system is considered as incompressible, "energy" equals "thermal energy". The reference point for the thermal energies in ColSim is a temperature of 0 C, i.e. the energy of a node with 0 C vanishes. The energy of a node is calculated from its heat capacity multiplied by its temperature difference with respect to 0 C.

9. The threedimensional array `qp_sum[n][i][j]`, where `i` numbers the nodes of each unit and `j` denotes the sums of different energies gained or lost. This array is not handed over to all type functions, only to those types which are assigned a temperature. E.g. in the pipe with the unit number `unit`, the array element `qp_sum[unit][k][1]` denotes the summed up energy losses of node `k` to the ambient. Contrary to this, the element `qp_sum[unit][k][2]` contains the sum of the external gains, where "sum" means a sum over all time steps from the start of the simulation to the current simulation time. Likewise, the incoming and outgoing energies are stored in order to calculate the new temperature of node k from the total sum over all energies, divided by the node's heat capacity.

10. The variable `tsim` counts the actual simulation time of the current day in seconds.

11. The variable `init_flag` is set to 1 at the beginning of the simulation run. In every function describing a type, there is a special part only for the case `init_flag=1` where parameters are checked for plausibility, inital energies and temperatures are set and some variables are set to their default values. After this initialization loop, `init_flag` is set to 0.

The function describing a type first reads in the information about parameters and old node temperatures. Then, the incoming signals, e.g. the mass and the heat flow arriving at the unit's inlet ports are read in from the array `out_n` with the help of the constant arrays `in_u` and `in_nr`: The array element `out_n[in_u[unit][k]][in_nr[unit][k]]` refers to the unitt's inlet number k. Literally, this array element contains the current output of the unit's predecessor unit, to be specific that output which arrives at the current unit's inlet port number `k` (see the explanation of `in_u` and `in_nr` in the paragraph above). Since the above expression is somewhat lenghty, the macro IN(k) was defined for `[in_u[unit][k]][in_nr[unit][k]]`, so inlet `k` of a unit can be addressed by `out_n IN(k)`.

Since the calculation of the units in ColSim proceeds according to the mass flow, the predecessor unit's newly calculated output for the actual time step can be used as input for the current unit in the same time step. If a unit has more than one predecessor unit, e.g. a mixer or a heat exchanger, it is called several times until all input values are updated (see section 6.4.1).

When the input is complete, the function starts calculating the behaviour of the current unit, then actualizes its column of the output array `OUT(i)`, if necessary sums up the lossed and gained internal energies in `qp_sum` to calculates the new temperatures and eventually returns control to the main program. More about the physics and modeling of the types in chapter 5.

Within the loop over all entries in the array `unit_order`, all units in the system are calculated according to this previously determined order. Afterwards, at the end of each time step, the main program performs an energy balance with the help of the function `energy_outcome`.c which balances the entries in `OUT(i),i=6,7,8,` summed over all units with heat flow. For more complicated types like the `storage` and the `collector`, more detailed energy balances are performed. If a certain tolerance in the balance is exceeded, the program execution is stopped.

The last commands in the time loop of `main` are concerned with switching the `out_o` with the `out_n` array and the `deriv_o` with the `deriv_n` array. Then, the next time step is calculated, until the end of the given simulation time interval is reached.

Chapter 7

Programmer's section

7.1 How to integrate new types into the system

Here, we deal with the case that for a special system called `Systemname` a standard type is exchanged with a non-standard type or a modified standard type, but the name of the type stays the same. E.g. in the simulation of `Systemname` instead of the standard fluid pipe (`pipe2.c` or `pipe3.c`. depending on the ColSim version) `pipe1.c` should be used.

Then, the only thing necessary is to edit the configuration file of the system, in this case `./projects/Systemname/Systemname.cfg`.

If there is no entry yet for the pipe, i.e. the standard pipe model mentioned in `./cfg/default.cfg` is used, the following line must be added:

```
pipe2.c        src/more_types/pipe1.c          TYPE.
```

Note that the reason why the generic pipe type is called `pipe2.c` here and not simply `pipe.c` is that "pipe" is a reserved name in Linux operating systems. Now save the file and exit the editor.

The configuration process can be started by closing the ColSim menu for the system and typing `ColSim Systemname` again. Alternatively, it can also be done without closing the menu by changing into the folder `./cfg` and typing: `config ../projects/-Systemname/Systemname.cfg`. This program sets the links listed in `./cfg/default.cfg` and `./projects/Example/Example.cfg`. Afterwards, a new compilation must be performed, either with the **make** button of the ColSim menu or by typing `m` in the folder `./src`.

7.2 How to write new types

By this we mean a type which didn't exist before, i.e. which receives a type number not used before. The easiest way to create such a new type is to copy the source code of an existing type as similar as possible and adapt the comment part at the beginning of the function. This comment part is important for the configuration process. Take a look at `./src/init_type_def_string.c` and choose a type number not used yet. Make

an entry with the name of the new type. If you append the type after the last existing entry, correct the number of types in the program lines below.

Then, a PRIORITY and HYDRAULIC number must be chosen with the help of the information in section 6.4.1. Next, the comments to parameters, in-and outlets should be adapted, since they appear in the `info_edit` window later.

Now the program code, written in ANSI-C follows. Note that there is an `init_flag`, which can be used for commands only to be executed once at program start. Please read chapter 6 for information about the global arrays. Note that the energy balance will also be performed for the new type, therefore, OUT 6,7,8 must be set correctly. If it's a hydraulic or ventilation type with mass flow, it consists of one or several nodes and the array `DERIV` (i) must receive the actual node temperatures. See `./src/global.h` for preprocessor definitions and macros.

At the end of the function characterizing the new type, the array `OUT(i)` (i.e. `out_n[unit][i]`) must be filled with the calculated values according to the explanations given in the comment part. These values can be made accessible to other units in the following way:

A xfig object must be created corresponding to the new type. Again, the best way is to copy an existing unit from a xfig graphics of a ColSim system and modify it. Follow the instructions in section 4.3.6 for opening the compound, then the individual parts of the unit can be deleted, modified or replaced with new graphical symbols. Also, the right in-and outlet boxes (corresponding to the in-and outlets used in the source code) can be created by changing the numbers in the old ones, if necessary. If finished, create a compound object again and copy it into the xfig graphics of the system which should contain the new type.

Now all units connected to the outlet boxes of the new type should be able to receive the values written to the `out_n` array in the program code. First, try to set the parameters of the new type by deleting it from the system graphics with the right mouse button, then choose **sort .xfig** in the ColSim menu. If you open the info_edit window afterwards, the right number of parameters together with the right comments (like written in the source code file) should appear. Edit the parameters and try to convert your system graphics with **fig2dek** afterwards,

Chapter 8

Appendix: Reference systems for checks

There are three reference systems at this time:

1. `SchichtSpeicherSystem` for the fluid systems. To check whether the fluids types function correctly, use `SchichtSpeicherSystem.fig` without any changes and perform a simulation. Then, the essential quantity is the auxiliary heating energy for one year which can be read off the entry for INP 4 in `sim_out0_total.dat`. The value should be 1081 kWh (ColSim0.57). Note that in this system the simulation time step shouldn't be chosen much larger than 10 secs, otherwise the simulation results start to depend upon the time step.

2. `Simple_zone` for the thermodynamics. Copy `simple_zone.ref.fig` onto `simple_zone.fig` and `load.ref.dat` onto `load.simple_zone.dat`. Then perform a simulation. The reference results for the room temperature, the relative and absolute humidity are stored in the picture `ref.mgr`. Type `xmgr` or `xmgrace` to start the graphics program, open `ref.mgr` in it and load the simulation results (`sim_out1.dat`, plot columns 3,4,5 versus column 1) into the picture in order to compare them with the reference results.

3. Zone1 for the building types: Copy `zone1.trnsys.fig` onto zone1.fig, then type `check_ref.ssc`. This shell script starts the simulation and evaluates the results automatically. In addition, a picture is shown with the calculated room temperature compared to that of a Trnsys simulation.

Bibliography

[1] J.A.Duffie and W.A.Beckmann, "Solar engineering of thermal processes", John Wiley&Sons Inc. 1991.

[2] H.Recknagel,E.Sprenger und W.Hönmann, "Taschenbuch für Heizung und Klimatechnik", Oldenbourg Verlag 1992.