DRAFT

HIGH PERFORMANCE COMMERCIAL BUILDING SYSTEMS

LIBRARY OF COMPONENT REFERENCE MODELS FOR FAULT DETECTION (AHU AND CHILLER)

June 20, 2001

Peng Xu and Philip Haves
Lawrence Berkeley National Laboratory

Subtask 2.3.3 Develop semi-automated, component-level diagnostic procedures
Element 5 – Integrated Commissioning & Diagnostics

BERKELEY LAB
ERNEST ORLANDO LAWRENCE
BERKELEY NATIONAL LABORATORY

TABLE OF CONTENTS

INTRODUCTION

The increasing complexity of building HVAC control and management systems heightens the need for the devolvement of tools to assist in monitoring the performance of these systems. The application of these tools is expected to lead to improved comfort, energy performance and reduced maintenance costs. The function of these tools may be limited to collecting raw data from sensors and control system outputs and displaying it for manual analysis by operators or engineers. Alternatively, the tools may analyze the data in order to determine whether the operation is correct or faulty (automated fault *detection*) and may also identify the location or nature of the physical cause of a problem (automated fault *diagnosis*).

In automated commissioning and fault diagnosis, a baseline model of correct operation is normally first configured and calibrated against design information and manufacturers' data. Next, the model is fined-tuned to match the actual performance after any faults have been fixed and the model is then used as part of a performance monitoring diagnostic tool for operations. The reference model is used to predict performance that would be expected in the absence of faults. A comparator is used to determine the significance of any differences between the predicted and measured performance and hence the level of confidence that a fault has been detected. Model-based fault detection is illustrated in Figure 1.

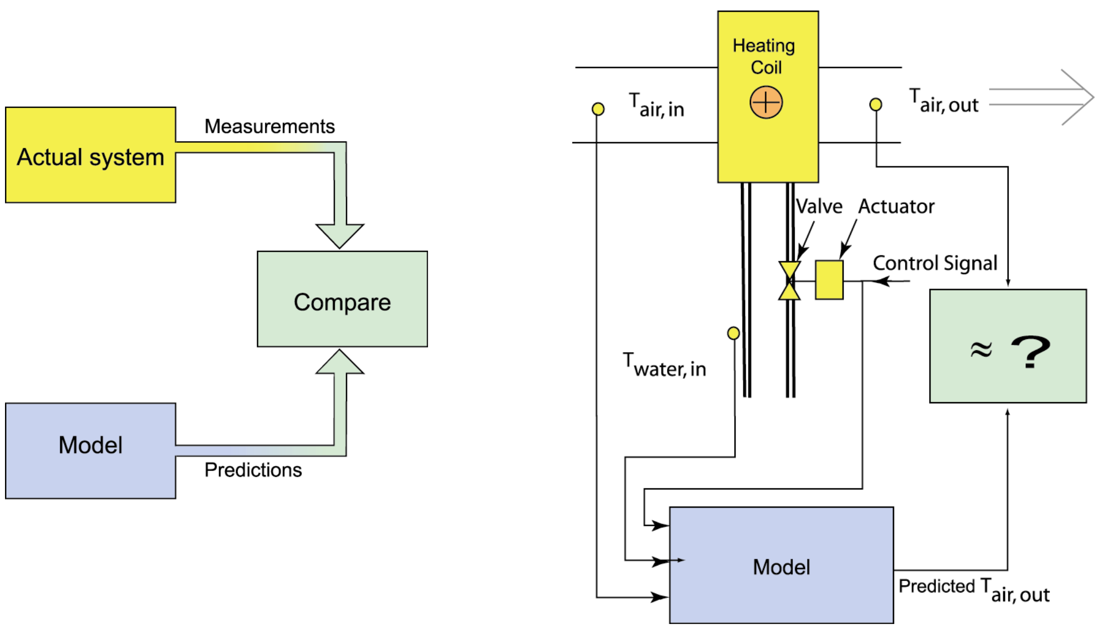

Figure 1: The concept of model-based fault detection and its application to a heating coil

Automated fault detection and diagnosis (AFDD) tools may either be implemented in a separate computer networked to the energy management and control systems (EMCS) or may be embedded in the EMCS itself. Separate implementation can be achieved with less engineering development work than embedded implementation and provides a stepping stone towards the tighter integration of embedded implementation, as well as providing a viable deplyment path in its own right.

The work reported here is part of Task 3.3 - *Develop Semi-Automated, Component-Level Diagnostic Procedures*, within Element 5 - *Integrated Commissioning and Diagnostics*. The aim of Task 3.3 is to create software procedures that will provide component-level fault detection and functional testing methods. The specific subtasks include:

- Develop a library of equipment models for component-level functional testing and performance monitoring
- Develop a toolbox of software procedures to support component-level, functional testing and performance monitoring.
- Conduct field trials to assess the performance of the methods and software tools.

This report describes and documents the library of equipment models as of July, 2003. A complete library of reference models would include models of all types of HVAC equipment, and possibly the associated controls. The work reported here focuses on developing diagnosis models for secondary HVAC system (air handling units and distribution systems) and chillers (simple chiller power model). It is expected that additional models will be added as a result of the work of IEA Annex 40 *Commissioning of Buildings and HVAC Systems for Improved Energy Performance* (see http://www.commissioning-hvac.org/). The library will be made available for download from http://buildings.lbl.gov/hpcbs/Element_5/02_E5_P2_3_3.html. The source code will be included in the download and will be subject to the provisions of LBNL's Open Source agreement~~(?)~~.

SPARK

The simulation program SPARK (SPARK 2003) has been used to develop and implement the reference model library. SPARK is an object-based software system that can be used to simulate physical systems that can be modeled using sets of differential and algebraic equations. 'Object-based' means that components and subsystems are modeled as objects that can be interconnected to form a model of the entire system. Often the same component and subsystem models can be used in many different system models, reducing the cost of development.

The process of describing a problem in order to produce a SPARK model begins by breaking it down in an object-oriented way. This involves thinking about the system in terms of its components, so that each component can be represented by a SPARK object. Then, a model is developed for each component not already available in a SPARK library. Since there may be several components of the same type, SPARK object models, i.e., equations or groups of equations, are defined in a generic manner, called classes. Classes serve as templates for creating any number of like objects that may be needed in a problem. The problem model is then completed by linking objects together, thus defining how they interact, specifying data values that specialize the model to represent the actual problem to be solved and providing boundary values.

SPARK models have a hierarchical structure. The smallest programming element is a class consisting of an individual equation, called an atomic class. Atomic classes are saved as files with extension **.cc**. A macro class consists of several atomic classes (and possibly other macro classes) combined together into a higher level unit. Macro classes are saved as **.cm** files. The ports of the different atomic classes with equal value are linked using equal objects defined in equal_link.cm.

Problem models are similarly described, using the atomic and macro classes, and placed in a problem specification file. When the problem is processed by SPARK, the problem specification file is converted to a C++ program, which gets compiled, linked and executed to solve the problem for a particular set of boundary conditions specified at run-time.

Just before the end of the High Performance Commercial Building Systems program, and after the completion of the work described here, a new Version of SPARK (VisualSPARK Version 2.0) was released. Version 2.0 has two new features that offer important advantages for the implementation of model-based fault detection:

- A SPARK problem can be compiled to produce a Dynamic Link Library (DLL), which allows the simulation to be called as a function from a fault detection program rather than using an 'exec' call with its associated overhead
- SPARK objects may be 'multi-valued', i.e. calculate more than one output variable, simplifying the process of creating and connecting models

New variants of the models in the library that are compatible with Version 2.0 will be added to the library as resources permit. In the meantime, it is suggested that intending users consider porting models of interest to Version 2.0 and contact the authors if they have questions or problems.

Contents of the Model Library

This document describes the component models developed using SPARK. For secondary systems, the major goal is to develop a full component model library to treat air-handling units (AHU). The components that have been modeled are coils (cooling and heating), fans, valves, and mixing boxes. Combining all these models together creates a model of an AHU. For primary systems, a simple chiller model based on the Gordon-Ng algorithm has been developed.

In this report, the documentation of each reference model in the library consists of a general description, a model description, and the source code. The general description describes the nature and function of the component and the possible common faults. The model description explains the model structure and the governing equations. The definitions of all the variables and the source code for all the classes for each component are also presented.

General description

'Coils' are fin-tube, air to water heat exchangers that are typically used for either cooling or heating the air supplied to conditioned spaces. Heating coils typically have one or two rows of tubes and are essentially cross-flow devices. Cooling coils typically have four or more rows and are essentially counterflow devices. They may provide dehumidification as well as sensible cooling and the surface in contact with the air may then be partially or completely wet. Most heating coils, at least in climates where there is no risk of freezing, and all cooling coils, are controlled by varying the flow rate of water through the coil. Coils in VAV systems also experience variable air-flow rate. The challenges in coil modeling are to treat the variation in surface resistance with flow rate and to treat partially wet operation.

The most common fault to be detected in either heating or cooling coils is fouling of the heat exchange surface, either on the air or the water-side. In order to detect fouling when it occurs, it is only necessary to model full load operation. However, in order to be able to predict loss of capacity at peak load before it occurs, it is necessary to model part load operation as well.

A significant number of coil models have been developed over the last few decades; none of the models that treat partly wet operation is entirely suitable for fault detection. In particular, there are two cooling coil models in the ASHRAE Secondary Toolkit (Brandemuehl et al., 19930. The simple model approximates partially wet operation as all wet or all dry, which leads to errors of up to 5%. The detailed model treats the dry and wet regions separately and iterates to find the position of the boundary. Testing of this model performed as part of the work described here showed that the iterative scheme employed in the model sometimes fails to converge under conditions of high humidity. For this reason, it was decided to develop a new model of partially wet coil operation.

Model description

In the new model, the coil is divided into discrete sections along the direction of fluid flow. In each section, heat and mass balance equations are established for each fluid, together with rate equations describing the heat and mass transfer. If the dew point temperature of the air is lower than the metal surface temperature, that section of the coil is treated as dry. If not, the water condensation rate is assumed to be proportional to the difference between the humidity ratio of the bulk air stream and the humidity ratio of saturated air at the temperature of the coil metal surface. The coefficient of proportionality is determined by assuming the value of the Lewis Number is unity. The sections that make up the coil are linked together by associating the fluid inlet conditions of one section with the outlet conditions for the adjacent upstream section.

The resulting set of coupled equations is then solved by SPARK. Although the computational burden of the new coil model is significantly greater than that of the ASHRAE Toolkit models, the model is robust, and it has the additional advantage of being a suitable starting point for a dynamic cooling coil model.

SPARK can not simulate a model with a dynamic number of objects. In the code, the number of sections~~layers~~ of the coil is hard-wired to 20 instead of being variable. Dividing the cooling coil into 20 layers provides enough accuracy, because under the common operation range of the cool coil, the driving temperature difference in each layer is one order of magnitude lower than the temperature change along the flow direction.

Two macro classes were developed for the cooling coil models. The class to model the heat and mass transfers within one section~~layer~~ of the coil is ***coil_section~~layer~~.cm***. The class that models the performance of a counter flow coil is ***coi_~~l~~counter_flow.cm***. This latter class invokes *coi_section~~l.layer~~.cm* to solve the heat and mass transfer equations for each section~~layer~~ of the coil. The counter flow coil class can be used for cooling and heating. ***RaCoil.cc*** is the class to calculate airside heat resistance and ***RLCoil.cc*** is the class to calculate waterside heat resistance

A simple heating coil model for crossflow heat exchange is used to simulate heating coil performance. With a known overall heat transfer coefficient, the capacity rates of the two fluid streams, the inlet fluid states, and the flow configuration, and effectiveness-NTU method can be used to determine outlet states. As in the cooling coil, the U value of the coil on the air-side and the water side is modeled as a function of the fluid flow rate. The macro class for this crossflow coil is ***coil_heating_cross_flow.cm***.

Governing equations

UA value

UA value of heat exchanger external surface and internal surface:

$$\frac{1}{UA_{total}} = R_{air,coil} + R_{liq,coil} + R_{metal}$$

$$R_{air,coil} = R_{air,coil,n}\left(\frac{m_{air,n}}{m_{air}}\right)^{0.6}$$

$$R_{liq,coil} = R_{liq,coil,n}\left(\frac{m_{liq,n}}{m_{liq}}\right)^{0.8}$$

$$UA_{int} = \frac{1}{R_{liq,coil}+R_{metal}}$$

$$UA_{ext} = \frac{1}{R_{air,coil}}$$

Cooling coil

For each layer of the cooling coil **:**
(coillayer.cm)

Heat transfer between air and cooling coil surface:

$$q_{sen,layer} = (\frac{T_{air,ent} + T_{air,lvg}}{2} - T_{sur}) \cdot UA_{ext}$$

$$q_{lat,layer} = \max\left\{0, (\frac{w_{air,ent} + w_{air,lvg}}{2} - w_{sur}) h_{fg} \cdot h_{mass}\right\}$$

Heat transfer between cooling coil surface and water flow:

$$q_{tot,layer} = (T_{sur} - \frac{T_{liq,ent} + T_{liq,lvg}}{2}) \cdot UA_{\text{int}}$$

~~**Coil REFERENCE MODELS**~~

Heat balance:

$$q_{tot,layer} = q_{sen,layer} + q_{lat,layer} = m_{liq} c_{liq} (T_{liq,lvg} - T_{liq,ent}) = m_{air,dry} (h_{air,ent} - h_{air,lvg})$$

$$q_{sen,layer} = m_{air,dry} c_p (T_{air,ent} - T_{air,lvg})$$

Others functions:
(In SPARK HVAC/Toolkit library, based on the ASHRAE Handbook Fundamentals)

$$h_{air,lvg} = enthalpy(T_{air,lvg}, w_{air,lvg})$$

$$h_{air,ent} = enthalpy(T_{air,ent}, w_{air,ent})$$

$$w_{sur} = humratio(T_{sur})$$

$$c_p = cpair(w_{air})$$

Counter-flow -cooling coil:
(coil_counter_drywet.cm)

$$T_{air,ent,i} = T_{air,lvg,i-1}$$

$$T_{air,lvg,i} = T_{air,ent,i+1}$$

$$T_{liq,ent,i} = T_{liq,lvg,i+1}$$

$$T_{liq,lvg,i} = T_{liq,lvg,i-1}$$

$$q_{sen} = \sum q_{sen,layer}$$

$$q_{tot} = \sum q_{tot,layer}$$

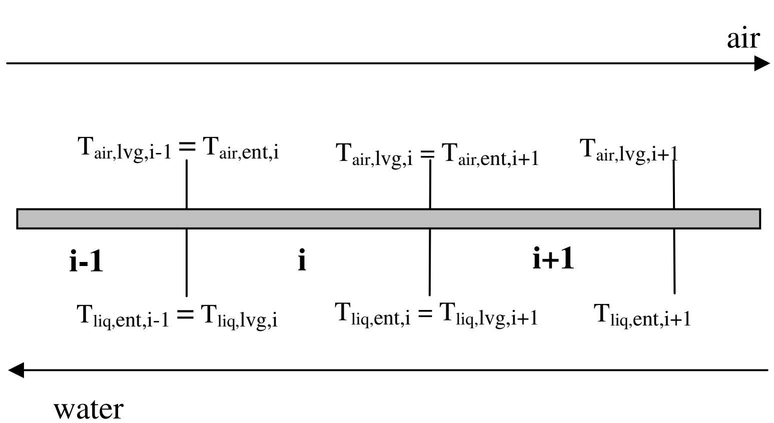

Figure 1:- Discrete sections of counter flow coil in the cooling coil model

Heating coil

(Cross flow, one side unmixed)

Determining the number of transfer units (NTU),

$$C_{AIR} = (c_{p,air} \cdot m_{air})$$
$$C_{LIQ} = (c_{liq} \cdot m_{liq})$$
$$C_{\min} = MIN(C_{AIR,}, C_{LIQ})$$
$$C_{\max} = MAX(C_{AIR,}, C_{LIQ})$$

$$NTU = \frac{UA_{total}}{C_{\min}}$$

The ratio of the two fluid capacity rates is,

$$R = \frac{C_{\min}}{C_{\max}}$$

Effectiveness for cross flow, minimum capacity rate stream unmixed,

$$\varepsilon = \frac{1 - e^{-R \cdot (1 - e^{-NTU})}}{R}$$

Outlet fluid condition are calculated from the definition of the effectiveness,

$$\frac{T_{liq,lvg} - T_{liq,ent}}{T_{air,ent} - T_{liq,ent}} = \varepsilon \cdot \frac{C_{\min}}{c_{liq} \cdot m_{liq}}$$

$$\frac{T_{air,ent} - T_{air,lvg}}{T_{air,ent} - T_{liq,ent}} = \varepsilon \cdot \frac{C_{\min}}{c_{air} \cdot m_{air}}$$

Nomenclature

Variables		Description	Unit
A_{ext}	AExt	External heat exchange area	m^2
A_{int}	AInt	Internal heat exchange area	m^2
c_{air}	CpAir	Air specific heat at constant pressure	kJ/kg.K
c_{water}	CLiq	Water specific heat	kJ/kg.K
C_{max}	CMax	Maximum of the two capacity rates	kW/K
C_{min}	CMin	Minimum of the two capacity rates	kW/K
C_{ext}	CExt	Constant of heat exchange of external surface	Dimensionless
C_{int}	CInt	Constant of heat exchange of internal surface	Dimensionless
ε	E	Effectiveness of the heat exchanger.	Dimensionless
$h_{air,ent}$	hairEnt	Coil entering air enthalpy	kJ/kg
$h_{air,lvg}$	hAirLvg	Coil leaving air enthalpy	kJ/kg
h_{mass}	hMass	Mass transfer coefficient	kg/s
h_{fg}	hfg	Eenthalpy of vaporization	kJ/kg
$m_{air,dry}$	mAir	Dry air flow rate	kg_dryair/s
m_{liq}	mLiq	Wwater flow rate	kg/s

Variables		Description	Unit
NTU	NTU	Number of transfer units of heat exchanger	Dimensionless
q_{sen}	qSen	Sensible heat transfer rate. Positive for air cooling	W
q_{lat}	qLat	Latent heat transfer rate. Positive for air cooling	W
q_{tot}	qTot	Heat transfer rate. Positive for air cooling	W
$T_{air,ent}$	TAirEnt	Coil entering air temperature	^oC
$T_{air,lvg}$	TAirLvg	Coil leaving air temperature	^oC
$T_{liq,lvg}$	TLiqLvg	Coil leaving water temperature	^oC
$T_{liq,ent}$	TLiqEnt	Coil entering water temperature	^oC
T_{sur}	TSur	Coil surface temperature	^oC
UA_{ext}	UAExt	Coil air side -external- heat transfer conductance	W/ K
UA_{int}	UAInt	Wet coil liquid side -internal- heat transfer conductance	W/ K
v_{air}	vAir	air velocity	m/s
v_{liq}	vLiq	water flow velocity	m/s
w_{sur}	wSur	Saturate air humidity ratio at coil surface temperature	kg/kg
$w_{air,lvg}$	wAirLvg	Coil leaving air humidity ratio	kg/kg
$w_{air,ent}$	wAirEnt	Coil entering air humidity ratio	kg/kg
$T_{air,ent,i}$	TAirEnt	Coil entering air temperature at layer i	^oC
$T_{air,lvg,i}$	TAirLvg	Coil leaving air temperature at layer i	^oC
$T_{liq,lvg,i}$	TLiqLvg	Coil leaving water temperature at layer i	^oC
$T_{liq,ent,i}$	TLiqEnt	Coil entering water temperature at layer i	^oC
$T_{air,ent,i-1}$	TAirEnt	Coil entering air temperature at layer i-1	^oC
$T_{air,lvg,i-1}$	TAirLvg	Coil leaving air temperature at layer i-1	^oC
$T_{liq,lvg,i-1}$	TLiqLvg	Coil leaving water temperature at layer i-1	^oC
$T_{liq,ent,i-1}$	TLiqEnt	Coil entering water temperature at layer i-1	^oC
$T_{air,ent,i+1}$	TAirEnt	Coil entering air temperature at layer i+1	^oC
$T_{air,lvg,i+1}$	TAirLvg	Coil leaving air temperature at layer i+1	^oC
$T_{liq,lvg,i+1}$	TLiqLvg	Coil leaving water temperature at layer i+1	^oC
$T_{liq,ent,i+1}$	TLiqEnt	Coil entering water temperature at layer i+1	^oC
R	R	Ratio of the C_{min} and C_{max}.	Dimensionless

```
/* CLASSMACRO  coil_layer

ABSTRACT
        modeling one portion or layer of the dry/wet coil.
ABSTRACT_END

Equations:
        qSen = ( (TAirEnt + TAirLvg)/2 - TSur ) * UAExt;
        qLat = ( (wAirEnt + wAirLvg)/2 - wSur ) * A*hMass
            =0 if [(wAirEnt + wAirLvg)/2 - wSur <0];
        qTot = UAInt * ( TSur - (TLiqLvg + TLiqEnt)/2 )
        qTot = mLiq * C_Water*(TLiqLvg - TLiqEnt)
        qSen = CAir * (TAirEnt -TAirLvg)
        qTot = mAir * ( hairEnt - hAirLvg)
        hAMass = UAExt * hfg /Cpm
        hAirLvg = f (TAirLvg, wAirLvg)
        hAirEnt = f (TAirEnt, wAirEnt)
        wSur = f ( TSur )
        qTot = qSen + qLat
        UAExt = CExt*AExt*mAir^0.8
        UAInt =Cint *AInt* mLiq^0.8

TEST_INPUT
        TAirEnt   = 31
        TAirLvg   = unknown
        wAirEnt   = 0.02
        wAirLvg   = unknown
        TLiqEnt   =8
        TLiqLvg   = unknown
        UAExt     =200
        UAInt     =400
        mAir      =1
        mLiq      =0.5
        PAtm      =101325
        qSen      = unknown
        qLat      = unknown
        qTot      = unknown
*/
// ==== PORTS ====
PORT    TAirEnt     "Coil entering air dry bulb temperature"                  [deg_C] ;
PORT    TAirLvg     "Coil leaving air dry bulb temperature"                   [deg_C] ;
PORT    wAirEnt     "Coil entering air humidity  ratio"                       [kg_water/kg_dryAir] ;
PORT    wAirLvg     "Coil leaving air humidity  ratio"                        [kg_water/kg_dryAir];
PORT    TLiqEnt     "Coil entering water temperature"                         [deg_C] ;
PORT    TLiqLvg     "Coil leaving water temperature"                          [deg_C] ;
PORT    UAExt       "Coil air side -external- heat transfer coefficient"      [W/deg_C] ;
PORT    UAInt       "Wet coil liquid side -internal- heat transfer coefficient" [W/deg_C] ;
PORT    mAir        "Air flow rate"                                           [kg_dryAir/s] ;
PORT    mLiq        "Liquid flow rate"                                        [kg/s] ;
PORT    PAtm        "Atmospheric pressure"                                    [Pa];
PORT    qSen        "Sensible heat transfer rate.  Positive for air cooling." [W];
PORT    qLat        "Latent heat transfer rate.  Positive for air cooling."   [W];
PORT    qTot        "Heat transfer rate.  Positive for air cooling."          [W];

declare cond qSen qLat qLiq qAirSen qLiqInt qAirTot;
declare average TAir wAir TLiq;
declare max2 max1;
declare safquot quot;
declare lat_rate hMass;
declare cpair cp;
declare capratel cpLiq;
declare cap_rate CAir;

declare equal_link qSen1 qSen2;
declare equal_link qLat1 ;
declare equal_link qTot1 qTot2 qTot3;
```

```
declare equal_link TSur1 TSur2;
declare equal_link TAirEnt1 TAirEnt2;
```



```
declare equal_link TAirLvg1 TAirLvg2;
declare equal_link wAirEnt1;
declare equal_link wAirLvg1 wAirLvg2;
declare equal_link UAExt1;
declare equal_link hMass1;
declare equal_link wSur1 wSur2;
declare equal_link TLiqEnt1;
declare equal_link TLiqLvg1;
declare equal_link hAirLvg1;
declare equal_link hAirEnt1;
declare equal_link mAir1;


// qSen = ( (TAirEnt + TAirLvg)/2 - TSur ) * UAExt
LINK            qSen.q                  qSen1.a  ;
LINK    .TAirEnt TAir.a                 TAirEnt1.a;
LINK    .TAirLvg TAir.b                 TAirLvg1.a;
LINK            TAir.c  qSen.T1         ;
LINK            qSen.T2                 TSur1.a;
LINK    .UAExt  qSen.U12                UAExt1.a ;

//qLat = ( (wAirEnt + wAirLvg)/2 - wSur ) * hMass or =0 if [(wAirEnt + wAirLvg) *0.5 - wSur <0]
LINK            qLat.q                  qLat1.a;
LINK    .wAirEnt wAir.a                 wAirEnt1.a;
LINK    .wAirLvg wAir.b                 wAirLvg1.a;
LINK            wAir.c  max1.a;
LINK            max1.b  qLat.T2         wSur1.a;
LINK            max1.c  qLat.T1;
LINK            qLat.U12                hMass1.b;

//hMass = UAExt * hfg /Cpm
LINK            quot.a                  UAExt1.b;
LINK            cp.w                    wSur1.b wSur2.a;
LINK            cp.CpAir quot.b;
LINK            quot.c  hMass.mAir;
LINK            hMass.cap               hMass1.a;

//qLiq = UAInt * ( TSur - (TLiqLvg + TLiqEnt)/2 );
LINK            qLiq.q                  qTot1.a;
LINK    .TLiqEnt TLiq.a                 TLiqEnt1.a;
LINK    .       TLiqLvg TLiq.b          TLiqLvg1.a;
LINK    TLiq.c qLiq.T2;
LINK            qLiq.T1                 TSur1.b TSur2.a;
LINK    .UAInt  qLiq.U12;

//qLiq = mLiq * C_Water*(TLiqLvg - TLiqEnt)
LINK            qLiqInt.q               qTot1.b qTot2.a;
LINK            qLiqInt.T2              TLiqEnt1.b;
LINK            qLiqInt.T1              TLiqLvg1.b;
LINK    .mLiq   cpLiq.mWater;
LINK            cpLiq.cap qLiqInt.U12;

//qSen = CAir * (TAirEnt -TAirLvg)
LINK            qAirSen.q qSen1.b qSen2.a;
LINK            qAirSen.T1              TAirEnt1.b TAirEnt2.a;
LINK            qAirSen.T2              TAirLvg1.b TAirLvg2.a;
LINK    .mAir CAir.mAir                 mAir1.a;
LINK            CAir.w                  wAirLvg1.b wAirLvg2.a;
LINK            CAir.cap qAirSen.U12;

//qTot = mAir * ( hairEnt - hAirLvg)
LINK            qAirTot.q               qTot2.b qTot3.a;
LINK            qAirTot.T2              hAirLvg1.a;
LINK            qAirTot.T1              hAirEnt1.a;
LINK            qAirTot.U12             mAir1.b;
```

```
// hAirLvg = f (TAirLvg, wAirLvg)
declare enthalpy enAirLvg;
```



```
LINK            enAirLvg.h          hAirLvg1.b;
LINK            enAirLvg.TDb        TAirLvg2.b;
LINK            enAirLvg.w          wAirLvg2.b;

// hAirEnt = f (TAirEnt, wAirEnt)
declare enthalpy enAirEnt;
LINK            enAirEnt.h          hAirEnt1.b;
LINK            enAirEnt.TDb        TAirEnt2.b;
LINK            enAirEnt.w          wAirEnt1.b;

//wSur = f ( TSur )
declare enthsat Surf;
LINK    .PAtm   Surf.PAtm;
LINK            Surf.TDb            TSur2.b;
LINK            Surf.w              wSur2.b;
LINK    hSur    Surf.hSat;

//qTot = qSen + qLat
declare sum q;
LINK    .qTot   q.c                 qTot3.b;
LINK    .qSen   q.a                 qSen2.b;
LINK    .qLat   q.b                 qLat1.b;
```

```
/* CLASSMACRO  coil_counter_drywet
        "model of counter flow coil, including total dry, total wet, partial dry and partial wet conditions"
         the model can be used both for cooling and heating purpose"
ABSTRACT
        The counter flow coil is divided into 20 layers in the direction of air flow.  The leaving
        condition of one layer is the entering condition of next layers. In each layer, model of class
        coil_layer is used.  This model demonstrates advantage over the model in HVAC toolkit and SPARK HVAC
toolkit
        in terms of mathematical stability and handling partial dry and wet conditions.

        ...
ABSTRACT_END
TEST_INPUT
        TAirEnt    = 31
        TAirLvg    = unknown
        wAirEnt    = 0.02
        wAirLvg    = unknown
        TLiqEnt    =8
        TLiqLvg    = unknown
        RAirN = 1/200
        RLiqN =1/4000
     RMet  =1/8000
        mAirN = 1
        mLiqN =1
        mAir       =1
        mLiq       =0.5
        PAtm       =101325
        qSen       = unknown
        qLat       = unknown
        qTot       = unknown
*/
PORT    TAirEnt         "Coil entering air dry bulb temperature"[deg_C] ;
PORT    TAirLvg         "Coil leaving air dry bulb temperature"     [deg_C] ;
PORT    wAirEnt         "Coil entering air humidity  ratio"         [kg_water/kg_dryAir] ;
PORT    wAirLvg         "Coil leaving air humidity  ratio"          [kg_water/kg_dryAir];
PORT    TLiqEnt         "Coil entering water temperature"           [deg_C] ;
PORT    TLiqLvg         "Coil leaving water temperature"            [deg_C] ;
PORT    RAirN           "Norminal coil air side -external- heat transfer resistance" [deg_C/W] ;
PORT    RLiqN           "Norminal coil liquid side -internal- heat transfer resistance" [deg_C/W] ;
PORT   RMet        "Coil metal heat transfer resistance" [deg_C/W];
PORT    mAirN           "Norminal coil air mass flow rate " [kg/s] ;
PORT    mLiqN           "Norminal coil liquid mass flow rate" [kg/s] ;
PORT    mAir            "Air flow"                                  [kg_dryAir/s] ;
PORT    mLiq            "Liquid flow"                                         [kg/s] ;
PORT    PAtm            "Atmospheric pressure"                                [Pa];
PORT    qSen            "Sensible heat transfer rate.  Positive for air cooling."          [W];
PORT    qLat            "Latent heat transfer rate.  Positive for air cooling." [W];
PORT    qTot            "Heat transfer rate.  Positive for air cooling." [W];

declare coil_layer l1 l2 l3 l4 l5 l6 l7 l8 l9 l10 l11 l12 l13 l14 l15 l16 l17 l18 l19 l20 ;

declare Racoil RA;
declare RLcoil RL;

link .RAirN RA.RairCoilN ;
link .mAirN RA.mAirCoilN ;
link .RLiqN RL.RLCoilN ;
link .mLiqN RL.mLCoilN ;

declare safrecip ext int;
declare sum intmet;
link RAir2 ext.a RA.RairCoil  ;
link RLiq2 intmet.a RL.RLCoil  ;
link .RMet intmet.b ;
link RLiqMet intmet.c int.a;

declare div20 div1 div2;
```

```
LINK UAExt ext.c div1.a;
LINK UAExtLayer div1.c I1.UAExt I2.UAExt I3.UAExt I4.UAExt I5.UAExt I6.UAExt I7.UAExt I8.UAExt I9.UAExt I10.UAExt
I11.UAExt I12.UAExt I13.UAExt I14.UAExt I15.UAExt I16.UAExt I17.UAExt I18.UAExt I19.UAExt I20.UAExt  ;
LINK UAInt int.c div2.a;
LINK UAIntLayer div2.c I1.UAInt I2.UAInt I3.UAInt I4.UAInt I5.UAInt I6.UAInt I7.UAInt I8.UAInt I9.UAInt I10.UAInt I11.UAInt
I12.UAInt I13.UAInt I14.UAInt I15.UAInt I16.UAInt I17.UAInt I18.UAInt I19.UAInt I20.UAInt ;

LINK .PAtm I1.PAtm I2.PAtm I3.PAtm I4.PAtm I5.PAtm I6.PAtm I7.PAtm I8.PAtm I9.PAtm I10.PAtm I11.PAtm I12.PAtm
I13.PAtm I14.PAtm I15.PAtm I16.PAtm I17.PAtm I18.PAtm I19.PAtm I20.PAtm ;
LINK .mAir    I1.mAir I2.mAir I3.mAir I4.mAir I5.mAir I6.mAir I7.mAir I8.mAir I9.mAir I10.mAir I11.mAir I12.mAir I13.mAir
I14.mAir I15.mAir I16.mAir I17.mAir I18.mAir I19.mAir I20.mAir RA.mAirCoil;
LINK .mLiq   I1.mLiq I2.mLiq I3.mLiq I4.mLiq I5.mLiq I6.mLiq I7.mLiq I8.mLiq I9.mLiq I10.mLiq I11.mLiq I12.mLiq I13.mLiq
I14.mLiq I15.mLiq I16.mLiq I17.mLiq I18.mLiq I19.mLiq I20.mLiq RL.mLCoil;

declare sum19 TL;
LINK  .TLiqEnt I1.TLiqEnt match_level=0, break_level=0;
LINK  T1 I1.TLiqLvg I2.TLiqEnt TL.a1 match_level=0, break_level=0;
LINK  T2 I2.TLiqLvg I3.TLiqEnt TL.a2 match_level=0, break_level=0;;
LINK  T3 I3.TLiqLvg I4.TLiqEnt TL.a3 match_level=0, break_level=0;;
LINK  T4 I4.TLiqLvg I5.TLiqEnt TL.a4 match_level=0, break_level=0;;
LINK  T5 I5.TLiqLvg I6.TLiqEnt TL.a5 match_level=0, break_level=0;;
LINK  T6 I6.TLiqLvg I7.TLiqEnt TL.a6 match_level=0, break_level=0;;
LINK  T7 I7.TLiqLvg I8.TLiqEnt TL.a7 match_level=0, break_level=0;;
LINK  T8 I8.TLiqLvg I9.TLiqEnt TL.a8 match_level=0, break_level=0;;
LINK  T9 I9.TLiqLvg I10.TLiqEnt TL.a9 match_level=0, break_level=0;;
LINK  T10 I10.TLiqLvg I11.TLiqEnt TL.a10 match_level=0, break_level=0;;
LINK  T11 I11.TLiqLvg I12.TLiqEnt TL.a11 match_level=0, break_level=0;;
LINK  T12 I12.TLiqLvg I13.TLiqEnt TL.a12 match_level=0, break_level=0;;
LINK  T13 I13.TLiqLvg I14.TLiqEnt TL.a13 match_level=0, break_level=0;;
LINK  T14 I14.TLiqLvg I15.TLiqEnt TL.a14 match_level=0, break_level=0;;
LINK  T15 I15.TLiqLvg I16.TLiqEnt TL.a15 match_level=0, break_level=0;;
LINK  T16 I16.TLiqLvg I17.TLiqEnt TL.a16 match_level=0, break_level=0;;
LINK  T17 I17.TLiqLvg I18.TLiqEnt TL.a17 match_level=0, break_level=0;;
LINK  T18 I18.TLiqLvg I19.TLiqEnt TL.a18 match_level=0, break_level=0;;
LINK  T19 I19.TLiqLvg I20.TLiqEnt TL.a19 match_level=0, break_level=0;;
declare sum nouse3;
link zero2 nouse3.a TL.a;
LINK TLiqLvg1 nouse3.b I20.TLiqLvg;
LINK  .TLiqLvg nouse3.c match_level=10, break_level=10;

declare sum19 TA;
LINK  .TAirEnt I20.TAirEnt;
LINK  Tw20 I20.TAirLvg match_level = 9 I19.TAirEnt TA.a1 match_level=10, break_level=10;
LINK  Tw19 I19.TAirLvg match_level = 9 I18.TAirEnt TA.a2 match_level=10, break_level=10;
LINK  Tw18 I18.TAirLvg match_level = 9 I17.TAirEnt TA.a3 match_level=10, break_level=10;
LINK  Tw17 I17.TAirLvg match_level = 9 I16.TAirEnt TA.a4 match_level=10, break_level=10;
LINK  Tw16 I16.TAirLvg match_level = 9 I15.TAirEnt TA.a5 match_level=10, break_level=10;
LINK  Tw15 I15.TAirLvg match_level = 9 I14.TAirEnt TA.a6 match_level=10, break_level=10;
LINK  Tw14 I14.TAirLvg match_level = 9 I13.TAirEnt TA.a7 match_level=10, break_level=10;
LINK  Tw13 I13.TAirLvg match_level = 9 I12.TAirEnt TA.a8 match_level=10, break_level=10;
LINK  Tw12 I12.TAirLvg match_level = 9 I11.TAirEnt TA.a9 match_level=10, break_level=10;
LINK  Tw11 I11.TAirLvg match_level = 9 I10.TAirEnt TA.a10 match_level=10, break_level=10;
LINK  Tw10 I10.TAirLvg match_level = 9 I9.TAirEnt  TA.a11 match_level=10, break_level=10;
LINK  Tw9 I9.TAirLvg match_level = 9 I8.TAirEnt  TA.a12 match_level=10, break_level=10;
LINK  Tw8 I8.TAirLvg match_level = 9 I7.TAirEnt  TA.a13 match_level=10, break_level=10;
LINK  Tw7 I7.TAirLvg match_level = 9 I6.TAirEnt TA.a14 match_level=10, break_level=10;
LINK  Tw6 I6.TAirLvg match_level = 9 I5.TAirEnt  TA.a15 match_level=10, break_level=10;
LINK  Tw5 I5.TAirLvg match_level = 9 I4.TAirEnt  TA.a16 match_level=10, break_level=10;
LINK  Tw4 I4.TAirLvg match_level = 9 I3.TAirEnt TA.a17 match_level=10, break_level=10;
LINK  Tw3 I3.TAirLvg match_level = 9 I2.TAirEnt  TA.a18 match_level=10, break_level=10;
LINK  Tw2 I2.TAirLvg match_level = 9 I1.TAirEnt TA.a19 match_level=10, break_level=10;
declare sum nouse2;
link zero1 nouse2.a TA.a;
LINK TAirLvg nouse2.b I1.TAirLvg;
LINK  .TAirLvg nouse2.c;

declare sum19 w;
```

```
LINK  .wAirEnt l20.wAirEnt;
LINK  w20 l20.wAirLvg match_level = 2 l19.wAirEnt w.a1;
LINK  w19 l19.wAirLvg match_level = 2 l18.wAirEnt w.a2 ;
LINK  w18 l18.wAirLvg match_level = 2 l17.wAirEnt w.a3;
LINK  w17 l17.wAirLvg match_level = 2 l16.wAirEnt w.a4;
LINK  w16 l16.wAirLvg match_level = 2 l15.wAirEnt w.a5;
LINK  w15 l15.wAirLvg match_level = 2 l14.wAirEnt w.a6;
LINK  w14 l14.wAirLvg match_level = 2 l13.wAirEnt w.a7;
LINK  w13 l13.wAirLvg match_level = 2 l12.wAirEnt w.a8;
LINK  w12 l12.wAirLvg match_level = 2 l11.wAirEnt w.a9;
LINK  w11 l11.wAirLvg match_level = 2 l10.wAirEnt w.a10;
LINK  w10 l10.wAirLvg match_level = 2 l9.wAirEnt  w.a11;
LINK  w9 l9.wAirLvg match_level = 2 l8.wAirEnt  w.a12;
LINK  w8 l8.wAirLvg match_level = 2 l7.wAirEnt  w.a13;
LINK  w7 l7.wAirLvg match_level = 2 l6.wAirEnt   w.a14;
LINK  w6 l6.wAirLvg match_level = 2 l5.wAirEnt  w.a15;
LINK  w5 l5.wAirLvg match_level = 2 l4.wAirEnt  w.a16;
LINK  w4 l4.wAirLvg match_level = 2 l3.wAirEnt  w.a17;
LINK  w3 l3.wAirLvg match_level = 2 l2.wAirEnt   w.a18;
LINK  w2 l2.wAirLvg match_level = 2 l1.wAirEnt  w.a19;
declare sum nouse1;
link zero nouse1.a w.a;
LINK  wAirLvg nouse1.b l1.wAirLvg;
LINK  .wAirLvg nouse1.c;

declare sum20 qSen;
LINK  qs1 l1.qSen qSen.a1;
LINK  qs2 l2.qSen qSen.a2;
LINK  qs3 l3.qSen qSen.a3;
LINK  qs4 l4.qSen qSen.a4;
LINK  qs5 l5.qSen qSen.a5;
LINK  qs6 l6.qSen qSen.a6;
LINK  qs7 l7.qSen qSen.a7;
LINK  qs8 l8.qSen qSen.a8;
LINK  qs9 l9.qSen qSen.a9;
LINK  qs10 l10.qSen qSen.a10;
LINK  qs11 l11.qSen qSen.a11;
LINK  qs12 l12.qSen qSen.a12;
LINK  qs13 l13.qSen qSen.a13;
LINK  qs14 l14.qSen qSen.a14;
LINK  qs15 l15.qSen qSen.a15;
LINK  qs16 l16.qSen qSen.a16;
LINK  qs17 l17.qSen qSen.a17;
LINK  qs18 l18.qSen qSen.a18;
LINK  qs19 l19.qSen qSen.a19;
LINK  qs20 l20.qSen qSen.a20;
LINK  .qSen qSen.sum;

declare sum20 qLat;
LINK  qL1 l1.qLat qLat.a1;
LINK  qL2 l2.qLat qLat.a2;
LINK  qL3 l3.qLat qLat.a3;
LINK  qL4 l4.qLat qLat.a4;
LINK  qL5 l5.qLat qLat.a5;
LINK  qL6 l6.qLat qLat.a6;
LINK  qL7 l7.qLat qLat.a7;
LINK  qL8 l8.qLat qLat.a8;
LINK  qL9 l9.qLat qLat.a9;
LINK  qL10 l10.qLat qLat.a10;
LINK  qL11 l11.qLat qLat.a11;
LINK  qL12 l12.qLat qLat.a12;
LINK  qL13 l13.qLat qLat.a13;
LINK  qL14 l14.qLat qLat.a14;
LINK  qL15 l15.qLat qLat.a15;
LINK  qL16 l16.qLat qLat.a16;
LINK  qL17 l17.qLat qLat.a17;
LINK  qL18 l18.qLat qLat.a18;
LINK  qL19 l19.qLat qLat.a19;
LINK  qL20 l20.qLat qLat.a20;
```

```
LINK  .qLat qLat.sum;

declare sum20 qTot;
LINK  qt1 l1.qTot qTot.a1 match_level=10, break_level=10;
LINK  qt2 l2.qTot qTot.a2 match_level=10, break_level=10;
LINK  qt3 l3.qTot qTot.a3 match_level=10, break_level=10;
LINK  qt4 l4.qTot qTot.a4 match_level=10, break_level=10;
LINK  qt5 l5.qTot qTot.a5 match_level=10, break_level=10;
LINK  qt6 l6.qTot qTot.a6 match_level=10, break_level=10;
LINK  qt7 l7.qTot qTot.a7 match_level=10, break_level=10;
LINK  qt8 l8.qTot qTot.a8 match_level=10, break_level=10;
LINK  qt9 l9.qTot qTot.a9 match_level=10, break_level=10;
LINK  qt10 l10.qTot qTot.a10 match_level=10, break_level=10;
LINK  qt11 l11.qTot qTot.a11 match_level=10, break_level=10;
LINK  qt12 l12.qTot qTot.a12 match_level=10, break_level=10;
LINK  qt13 l13.qTot qTot.a13 match_level=10, break_level=10;
LINK  qt14 l14.qTot qTot.a14 match_level=10, break_level=10;
LINK  qt15 l15.qTot qTot.a15 match_level=10, break_level=10;
LINK  qt16 l16.qTot qTot.a16 match_level=10, break_level=10;
LINK  qt17 l17.qTot qTot.a17 match_level=10, break_level=10;
LINK  qt18 l18.qTot qTot.a18 match_level=10, break_level=10;
LINK  qt19 l19.qTot qTot.a19 match_level=10, break_level=10;
LINK  qt20 l20.qTot qTot.a20 match_level=10, break_level=10;
LINK  .qTot qTot.sum;

/*

PROBE wSur2 l2'wSur2 INIT= 0.01 match_level=10, break_level=10;
PROBE wSur3 l3'wSur2 INIT= 0.01 match_level=10, break_level=10;
PROBE wSur4 l4'wSur2 INIT= 0.01 match_level=10, break_level=10;
PROBE wSur5 l5'wSur2 INIT= 0.01 match_level=10, break_level=10;
PROBE wSur6 l6'wSur2 INIT= 0.01 match_level=10, break_level=10;
PROBE wSur7 l7'wSur2 INIT= 0.01 match_level=10, break_level=10;
PROBE wSur8 l8'wSur2 INIT= 0.01 match_level=10, break_level=10;
PROBE wSur9 l9'wSur2 INIT= 0.01 match_level=10, break_level=10;
PROBE wSur10 l10'wSur2 INIT= 0.01 match_level=10, break_level=10;
PROBE wSur11 l11'wSur2 INIT= 0.01 match_level=10, break_level=10;
PROBE wSur12 l12'wSur2 INIT= 0.01 match_level=10, break_level=10;
PROBE wSur13 l13'wSur2 INIT= 0.01 match_level=10, break_level=10;
PROBE wSur14 l14'wSur2 INIT= 0.01 match_level=10, break_level=10;
PROBE wSur15 l15'wSur2 INIT= 0.01 match_level=10, break_level=10;
PROBE wSur16 l16'wSur2 INIT= 0.01 match_level=10, break_level=10;
PROBE wSur17 l17'wSur2 INIT= 0.01 match_level=10, break_level=10;
PROBE wSur18 l18'wSur2 INIT= 0.01 match_level=10, break_level=10;
PROBE wSur19 l19'wSur2 INIT= 0.01 match_level=10, break_level=10;


 */
```

```
/* CLASSMACRO  coil_section

ABSTRACT
            modeling one portion or layer of the dry/wet coil.
            ...
ABSTRACT_END

Equations:
            qSen = ( (TAirEnt + TAirLvg)/2 - TSur ) * UAExt;
            qLat = ( (wAirEnt + wAirLvg)/2 - wSur ) * hMass
                =0 if [(wAirEnt + wAirLvg)/2 - wSur <0];
            qTot = UAInt * ( TSur - (TLiqLvg + TLiqEnt)/2 )
            qTot = mLiq * C_Water*(TLiqLvg - TLiqEnt)
            qSen = CAir * (TAirEnt -TAirLvg)
            qTot = mAir * ( hairEnt - hAirLvg)
            hAMass = UAExt * hfg /Cpm
            hAirLvg = fh (TAirLvg, wAirLvg)
            hAirEnt = fh (TAirEnt, wAirEnt)
            wSur = fwSur ( TSur )
            qTot = qSen + qLat

TEST_INPUT
            TAirEnt   = 31
            TAirLvg   = unknown
            wAirEnt   = 0.02
            wAirLvg   = unknown
            TLiqEnt   =8
            TLiqLvg   = unknown
            UAExt     =200
            UAInt     =400
            mAir      =1
            mLiq      =0.5
            PAtm      =101325
            qSen      = unknown
            qLat      = unknown
            qTot      = unknown
*/
// ==== PORTS ====

PORT    TAirEnt        "Coil entering air dry bulb temperature"[deg_C] ;
PORT    TAirLvg        "Coil leaving air dry bulb temperature"    [deg_C] ;
PORT    wAirEnt        "Coil entering air humidity  ratio"        [kg_water/kg_dryAir] ;
PORT    wAirLvg        "Coil leaving air humidity  ratio"         [kg_water/kg_dryAir];
PORT    TLiqEnt        "Coil entering water temperature"          [deg_C] ;
PORT    TLiqLvg        "Coil leaving water temperature"           [deg_C] ;
PORT    UAExt          "Coil air side -external- heat transfer coefficient" [W/deg_C] ;
PORT    UAInt          "Wet coil liquid side -internal- heat transfer coefficient" [W/deg_C] ;
PORT    mAir           "Air flow"                                 [kg_dryAir/s] ;
PORT    mLiq           "Liquid flow"                                         [kg/s] ;
PORT    PAtm           "Atmospheric pressure"                                [Pa];
PORT    qSen           "Sensible heat transfer rate.  Positive for air cooling."         [W];
PORT    qLat           "Latent heat transfer rate.  Positive for air cooling." [W];
PORT    qTot           "Heat transfer rate.  Positive for air cooling." [W];

declare cond qSen qLat qLiq qAirSen qLiqInt qAirTot;
declare average TAir wAir TLiq;
declare max2 max1;
declare safquot quot;
declare lat_rate hMass;
declare cpair cp;
declare capratel cpLiq;
declare cap_rate CAir;

declare equal_link eq1 eq2 eq3 eq4 eq5 eq6 eq7 eq8 eq9 eq10 eq11 eq12 eq13 eq14 eq15 eq16 eq17 eq18 eq19 eq20
eq21 eq22 eq23 eq24;


// qSen = ( (TAirEnt + TAirLvg)/2 - TSur ) * UAExt
```

```
LINK            qSen    qSen.q                  eq1.a   ;
LINK    .TAirEnt TAir.a                 eq2.a;
LINK    .TAirLvg TAir.b                 eq3.a;
LINK            TAirEnt  TAir.c    qSen.T1              ;
LINK            TAirSur  qSen.T2                eq4.a;
LINK    .UAExt          qSen.U12        eq5.a   ;

//qLat = ( (wAirEnt + wAirLvg)/2 - wSur ) * hMass or =0 if [(wAirEnt + wAirLvg) *0.5 - wSur <0]
LINK            qLat    qLat.q                  eq6.a;
LINK    .wAirEnt wAir.a                         eq7.a;
LINK    .wAirLvg wAir.b                         eq8.a;
LINK            wAirEnt  wAir.c    max1.a;
LINK            wAirSur max1.b     qLat.T2      eq9.a;
LINK            wAirSu1  max1.c    qLat.T1;
LINK            hMass    qLat.U12       eq10.b;

//hMass = UAExt * hfg /Cpm
LINK            UAExt1   quot.a                 eq5.b;
LINK            wAir     cp.w                   eq9.b eq22.a;
LINK            cp       cp.CpAir quot.b        ;
LINK    mAir    quot.c   hMass.mAir;
LINK            hMass1   hMass.cap              eq10.a;

//qLiq = UAInt * ( TSur - (TLiqLvg + TLiqEnt)/2 );
LINK            qLiq     qLiq.q                 eq11.a;
LINK    .TLiqEnt TLiq.a                 eq12.a;
LINK    .TLiqLvg TLiq.b                 eq13.a;
LINK    TLiqEnt  TLiq.c qLiq.T2;
LINK            TSur6    qLiq.T1                eq4.b eq21.a;
LINK   .UAInt            qLiq.U12;

//qLiq = mLiq * C_Water*(TLiqLvg - TLiqEnt)
LINK            qLiq1    qLiqInt.q      eq11.b eq24.a;
LINK            TliqEn1  qLiqInt.T2     eq12.b;
LINK            TliqLv1  qLiqInt.T1     eq13.b;
LINK    .mLiq            cpLiq.mWater;
LINK            CmLiq    cpLiq.cap qLiqInt.U12;

//qSen = CAir * (TAirEnt -TAirLvg)
LINK    qSen1   qAirSen.q          eq1.b eq14.a;
LINK            TAirEn2  qAirSen.T1             eq2.b eq15.a;
LINK            TAirLv3  qAirSen.T2             eq3.b eq16.a;
LINK    .mAir CAir.mAir                         eq20.a;
LINK            wAir3    CAir.w                 eq8.b eq17.a;
LINK            CmAir    CAir.cap qAirSen.U12;

//qTot = mAir * ( hairEnt - hAirLvg)
LINK            qTot2    qAirTot.q      eq24.b eq23.a;
LINK            hAirLv2  qAirTot.T2             eq18.a;
LINK            hAirEn2  qAirTot.T1             eq19.a;
LINK            mAir2    qAirTot.U12            eq20.b;

// hAirLvg = f (TAirLvg, wAirLvg)
declare enthalpy enAirLvg;
LINK            hAirLv4  enAirLvg.h             eq18.b;
LINK            TAirlv4 enAirLvg.TDb            eq16.b;
LINK            wAirLv5 enAirLvg.w      eq17.b;

// hAirEnt = f (TAirEnt, wAirEnt)
declare enthalpy enAirEnt;
LINK            hAirEn4 enAirEnt.h      eq19.b;
LINK            TAirEn4 enAirEnt.TDb            eq15.b;
LINK            wAirEn4 enAirEnt.w      eq7.b;

//wSur = f ( TSur )
declare enthsat Surf;
LINK .PAtm               Surf.PAtm;
LINK            TDb      Surf.TDb       eq21.b;
```

```
LINK            wSur2    Surf.w                  eq22.b
                INIT = 0.010 ;
LINK hSur       Surf.hSat /* useless end*/;

//qTot = qSen + qLat
declare sum q;
LINK    .qTot   q.c     INIT =0.0 break_level =10        eq23.b;
LINK    .qSen   q.a                           eq14.b;
LINK    .qLat   q.b                           eq6.b;
```

Racoil.cc Coil / SOURCE CODE

```
/* CLASS  RaCoil    "Air side R value of heat exchanger"

ABSTRACT
          ...
          ...
ABSTRACT_END
TEST_INPUT
    RairCoilN = norminal air side R value 100 Degree_C/W
        mAirCoilN = norminal air flow rate  10 kg/s
        mAirCoil = 5 kg/s
*/
#ifdef SPARK_TEXT
// ==== PORTS ====

PORT     RairCoilN  "norminal air side R value" [Degree_C/W];
PORT     RairCoil  "air side R value" [Degree_C/W];
PORT     mAirCoilN  "norminal air flow rate" [kg/s];
PORT     mAirCoil   "air mass flow rate" [kg/s];

EQUATIONS { RairCoil = RairCoilN ( mAirCoilN/mAirCoil)^0.6
          }

// ==== FUNCTIONS ====
FUNCTIONS {
        RairCoil = RairCoil( RairCoilN, mAirCoilN, mAirCoil);

          }
#endif /* SPARK_TEXT */
#include "spark.h"

EVALUATE(RairCoil)
{
  ARGDEF(0,RairCoilN) ;
  ARGDEF(1,mAirCoilN) ;
  ARGDEF(2,mAirCoil) ;
  double RairCoil;

  RairCoil  = pow ( (mAirCoilN/mAirCoil),0.6);

  RETURN(RairCoil) ;
}
```

RLcoil.cc Coil / SOURCE CODE

```
/* CLASS  RLCoil    "liquid side R value of heat exchanger"

ABSTRACT
          ...
          ...
ABSTRACT_END
TEST_INPUT
    RLCoilN = norminal liquid side R value 100 Degree_C/W
        mLCoilN = norminal air flow rate  10 kg/s
        mLCoil = 5 kg/s
*/
#ifdef SPARK_TEXT
// ==== PORTS ====

PORT     RLCoilN  "norminal air side R value" [Degree_C/W];
PORT     RLCoil  "air side R value" [Degree_C/W];
PORT     mLCoilN  "norminal air flow rate" [kg/s];
PORT     mLCoil   "air mass flow rate" [kg/s];

EQUATIONS { RLCoil = RLCoilN ( mLCoilN/mLCoil)^0.6
          }

// ==== FUNCTIONS ====
FUNCTIONS {
        RLCoil = RLCoil( RLCoilN, mLCoilN, mLCoil);

          }
#endif /* SPARK_TEXT */
#include "spark.h"

EVALUATE(RLCoil)
{
  ARGDEF(0,RLCoilN) ;
  ARGDEF(1,mLCoilN) ;
  ARGDEF(2,mLCoil) ;
  double RLCoil;

  RLCoil  = pow ( (mLCoilN/mLCoil),0.8);

  RETURN(RLCoil) ;
}
```

```
/* CLASSMACRO  coil_counter_drywet
        "model of counter flow coil, including total dry, total wet, partial dry and partial wet conditions"
        the model can be used both for cooling and heating purpose"
ABSTRACT
        The counter flow coil is divided into 20 layers in the direction of air flow.  The leaving
        condition of one layer is the entering condition of next layers. In each layer, model of class
        coil_layer is used.  This model demonstrates advantage over the model in HVAC toolkit and SPARK HVAC
        toolkit
        in terms of mathematical stability and handling partial dry and wet conditions.

        ...
ABSTRACT_END
TEST_INPUT
        TAirEnt    = 31
        TAirLvg    = unknown
        wAirEnt    = 0.02
        wAirLvg    = unknown
        TLiqEnt    =8
        TLiqLvg    = unknown
        UAExt      =200
        UAInt      =400
        mAir       =1
        mLiq       =0.5
        PAtm       =101325
        qSen       = unknown
        qLat       = unknown
        qTot       = unknown
*/
PORT    TAirEnt     "Coil entering air dry bulb temperature"                               [deg_C] ;
PORT    TAirLvg     "Coil leaving air dry bulb temperature"                                [deg_C] ;
PORT    wAirEnt     "Coil entering air humidity  ratio"                                    [kg /kg_dryAir] ;
PORT    wAirLvg     "Coil leaving air humidity  ratio"                                     [kg /kg_dryAir];
PORT    TLiqEnt     "Coil entering water temperature"                                      [deg_C] ;
PORT    TLiqLvg     "Coil leaving water temperature"                                       [deg_C] ;
PORT    AExt        "Heat exchange area - Coil air side -external"                         [W/deg_C] ;
PORT    AInt        "Heat exchange area - Wet coil liquid side -internal"                  [W/deg_C] ;
PORT    CExt        "Constant- Coil air side -external- heat transfer coefficient"         [W/deg_C] ;
PORT    CInt        "Constant - Wet coil liquid side -internal- heat transfer coefficient" [W/deg_C] ;
PORT    mAir        "Air flow"                                                             [kg_dryAir/s] ;
PORT    mLiq        "Liquid flow"                                                          [kg/s] ;
PORT    PAtm        "Atmospheric pressure"                                                 [Pa];
PORT    qSen        "Sensible heat transfer rate.  Positive for air cooling."              [W];
PORT    qLat        "Latent heat transfer rate.  Positive for air cooling."                [W];
PORT    qTot        "Heat transfer rate.  Positive for air cooling."                        [W];

declare coil_layer l1 l2 l3 l4 l5 l6 l7 l8 l9 l10 l11 l12 l13 l14 l15 l16 l17 l18 l19 l20 ;

declare UA UAExt UAInt;
LINK .AExt UAExt.AreaHX;
LINK .CExt UAExt.C;
LINK .AInt UAInt.AreaHX;
LINK .CInt UAInt.C;

declare div20 div1 div2;
LINK UAExt.UA div1.a;
LINK div1.c l1.UAExt l2.UAExt l3.UAExt l4.UAExt l5.UAExt l6.UAExt l7.UAExt l8.UAExt l9.UAExt l10.UAExt l11.UAExt
l12.UAExt l13.UAExt l14.UAExt l15.UAExt l16.UAExt l17.UAExt l18.UAExt l19.UAExt l20.UAExt  ;
LINK UAInt.UA div2.a;
LINK div2.c l1.UAInt l2.UAInt l3.UAInt l4.UAInt l5.UAInt l6.UAInt l7.UAInt l8.UAInt l9.UAInt l10.UAInt l11.UAInt l12.UAInt
l13.UAInt l14.UAInt l15.UAInt l16.UAInt l17.UAInt l18.UAInt l19.UAInt l20.UAInt ;

LINK .PAtm l1.PAtm l2.PAtm l3.PAtm l4.PAtm l5.PAtm l6.PAtm l7.PAtm l8.PAtm l9.PAtm l10.PAtm l11.PAtm l12.PAtm
l13.PAtm l14.PAtm l15.PAtm l16.PAtm l17.PAtm l18.PAtm l19.PAtm l20.PAtm ;
LINK .mAir   UAExt.m l1.mAir l2.mAir l3.mAir l4.mAir l5.mAir l6.mAir l7.mAir l8.mAir l9.mAir l10.mAir l11.mAir l12.mAir
l13.mAir l14.mAir l15.mAir l16.mAir l17.mAir l18.mAir l19.mAir l20.mAir ;
LINK .mLiq   UAInt.m l1.mLiq l2.mLiq l3.mLiq l4.mLiq l5.mLiq l6.mLiq l7.mLiq l8.mLiq l9.mLiq l10.mLiq l11.mLiq l12.mLiq
l13.mLiq l14.mLiq l15.mLiq l16.mLiq l17.mLiq l18.mLiq l19.mLiq l20.mLiq ;
```

```
LINK .TliqEnt      l1.TLiqEnt;
LINK T1            l1.TLiqLvg         l2.TLiqEnt ;
LINK T2            l2.TLiqLvg         l3.TLiqEnt ;
LINK T3            l3.TLiqLvg         l4.TLiqEnt ;
LINK T4            l4.TLiqLvg         l5.TLiqEnt ;
LINK T5            l5.TLiqLvg         l6.TLiqEnt ;
LINK T6            l6.TLiqLvg         l7.TLiqEnt ;
LINK T7            l7.TLiqLvg         l8.TLiqEnt ;
LINK T8            l8.TLiqLvg         l9.TLiqEnt ;
LINK T9            l9.TLiqLvg         l10.TLiqEnt ;
LINK T10           l10.TLiqLvg        l11.TLiqEnt;
LINK T11           l11.TLiqLvg        l12.TLiqEnt;
LINK T12           l12.TLiqLvg        l13.TLiqEnt;
LINK T13           l13.TLiqLvg        l14.TLiqEnt;
LINK T14           l14.TLiqLvg        l15.TLiqEnt;
LINK T15           l15.TLiqLvg        l16.TLiqEnt;
LINK T16           l16.TLiqLvg        l17.TLiqEnt;
LINK T17           l17.TLiqLvg        l18.TLiqEnt;
LINK T18            l18.TLiqLvg       l19.TLiqEnt;
LINK T19           l19.TLiqLvg        l20.TLiqEnt;
LINK .TLiqLvg      l20.TLiqLvg;

LINK .TAirEnt      l20.TAirEnt;
LINK Tw20          l20.TAirLvg        l19.TAirEnt;
LINK Tw19          l19.TAirLvg        l18.TAirEnt;
LINK Tw18          l18.TAirLvg        l17.TAirEnt;
LINK Tw17          l17.TAirLvg        l16.TAirEnt;
LINK Tw16          l16.TAirLvg        l15.TAirEnt;
LINK Tw15          l15.TAirLvg        l14.TAirEnt;
LINK Tw14          l14.TAirLvg        l13.TAirEnt;
LINK Tw13          l13.TAirLvg        l12.TAirEnt;
LINK Tw12          l12.TAirLvg        l11.TAirEnt;
LINK Tw11          l11.TAirLvg        l10.TAirEnt;
LINK Tw10          l10.TAirLvg        l9.TAirEnt ;
LINK Tw9           l9.TAirLvg         l8.TAirEnt ;
LINK Tw8           l8.TAirLvg         l7.TAirEnt ;
LINK Tw7           l7.TAirLvg         l6.TAirEnt;
LINK Tw6           l6.TAirLvg         l5.TAirEnt ;
LINK Tw5           l5.TAirLvg         l4.TAirEnt ;
LINK Tw4           l4.TAirLvg         l3.TAirEnt;
LINK Tw3           l3.TAirLvg         l2.TAirEnt ;
LINK Tw2           l2.TAirLvg         l1.TAirEnt;
LINK .TAirLvg      l1.TAirLvg;

LINK .wAirEnt      l20.wAirEnt;
LINK w20           l20.wAirLvg        l19.wAirEnt;
LINK w19           l19.wAirLvg        l18.wAirEnt;
LINK w18           l18.wAirLvg        l17.wAirEnt;
LINK w17           l17.wAirLvg        l16.wAirEnt;
LINK w16           l16.wAirLvg        l15.wAirEnt;
LINK w15           l15.wAirLvg        l14.wAirEnt;
LINK w14           l14.wAirLvg        l13.wAirEnt;
LINK w13           l13.wAirLvg        l12.wAirEnt;
LINK w12           l12.wAirLvg        l11.wAirEnt;
LINK w11           l11.wAirLvg        l10.wAirEnt;
LINK w10           l10.wAirLvg        l9.wAirEnt  ;
LINK w9            l9.wAirLvg         l8.wAirEnt ;
LINK w8            l8.wAirLvg         l7.wAirEnt ;
LINK w7            l7.wAirLvg         l6.wAirEnt  ;
LINK w6            l6.wAirLvg         l5.wAirEnt ;
LINK w5            l5.wAirLvg         l4.wAirEnt ;
LINK w4            l4.wAirLvg         l3.wAirEnt ;
LINK w3            l3.wAirLvg         l2.wAirEnt  ;
LINK w2            l2.wAirLvg         l1.wAirEnt ;
LINK .wAirLvg      l1.wAirLvg;

declare sum20 qSen;
```

coil_cooling_counter_flow.cm Coil / SOURCE CODE

```
LINK	l1.qSen	qSen.a1;
LINK	l2.qSen	qSen.a2;
LINK	l3.qSen	qSen.a3;
LINK	l4.qSen	qSen.a4;
LINK	l5.qSen	qSen.a5;
LINK	l6.qSen	qSen.a6;
LINK	l7.qSen	qSen.a7;
LINK	l8.qSen	qSen.a8;
LINK	l9.qSen	qSen.a9;
LINK	l10.qSen	qSen.a10;
LINK	l11.qSen	qSen.a11;
LINK	l12.qSen	qSen.a12;
LINK	l13.qSen	qSen.a13;
LINK	l14.qSen	qSen.a14;
LINK	l15.qSen	qSen.a15;
LINK	l16.qSen	qSen.a16;
LINK	l17.qSen	qSen.a17;
LINK	l18.qSen	qSen.a18;
LINK	l19.qSen	qSen.a19;
LINK	l20.qSen	qSen.a20;
LINK	.qSen	qSen.sum;

declare	sum20	qLat;
LINK	l1.qLat	qLat.a1;
LINK	l2.qLat	qLat.a2;
LINK	l3.qLat	qLat.a3;
LINK	l4.qLat	qLat.a4;
LINK	l5.qLat	qLat.a5;
LINK	l6.qLat	qLat.a6;
LINK	l7.qLat	qLat.a7;
LINK	l8.qLat	qLat.a8;
LINK	l9.qLat	qLat.a9;
LINK	l10.qLat	qLat.a10;
LINK	l11.qLat	qLat.a11;
LINK	l12.qLat	qLat.a12;
LINK	l13.qLat	qLat.a13;
LINK	l14.qLat	qLat.a14;
LINK	l15.qLat	qLat.a16;
LINK	l17.qLat	qLat.a17;
LINK	l18.qLat	qLat.a18;
LINK	l19.qLat	qLat.a19;
LINK	l20.qLat	qLat.a20;
LINK	.qLat	qLat.sum;

declare sum20 qTot;
LINK	l1.qTot	qTot.a1;
LINK	l2.qTot	qTot.a2;
LINK	l3.qTot	qTot.a3;
LINK	l4.qTot	qTot.a4;
LINK	l5.qTot	qTot.a5;
LINK	l6.qTot	qTot.a6;
LINK	l7.qTot	qTot.a7;
LINK	l8.qTot	qTot.a8;
LINK	l9.qTot	qTot.a9;
LINK	l10.qTot	qTot.a10;
LINK	l11.qTot	qTot.a11;
LINK	l12.qTot	qTot.a12;
LINK	l13.qTot	qTot.a13;
LINK	l14.qTot	qTot.a14;
LINK	l15.qTot	qTot.a15;
LINK	l16.qTot	qTot.a16;
LINK	l17.qTot	qTot.a17;
LINK	l18.qTot	qTot.a18;
LINK	l19.qTot	qTot.a19;
LINK	l20.qTot	qTot.a20;
LINK	.qTot	qTot.sum;
```

```
/*+++
 Identification:  heating coil, cross flow, stream 1 unmixed
 Abstract:
 Notes:
     The configuration is cross flow, stream 1 unmixed
 Interface:
    mAirEnt:          Air flow (kg dry air/s)
    mLiq:             Liquid flow (kg/s)
    TAirEnt:          Entering air dry bulb temperature (deg-C)
    TLiqEnt:          Entering water temperature (deg C)
    wAirEnt:          Entering air humidity  ratio (kg-water/kg dry air)
    CHx:              Constant of heat exchanger coefficient
    AHx:              Overall heat exchanger surface area (m2)
    mAirLvg:          Leaving air flow (kg dry air/s)
    wAirLvg:          Leaving air humidity  ratio (kg-water/kg dry air)
    TAirLvg:          Leaving air Temperature (deg C)
    TLiqLvg:          Leaving water temperature (deg C)
    q:                Heat transfer rate.  Positive for air cooling. (W)
 Acceptable input set:
     CHx = 1000, AHx = 1, mAirEnt = 1, mLiq = 1, TAirEnt = 15, wAirEnt = 0.001,
     TLiqEnt = 50
 Recommended matches:
        None
 Suggested breaks:
        None
 Local variables:
    capAir:   Air capacity rate (kg/s)
    capLiq:   Water capacity rate (kg/s)
 Equations:
    capAir = mAirEnt*(CpAir+WAirEnt*CpVap)
    capLiq = MLiq*CpLiq
    heatex(capLiq,TLiqEnt,capAir,TAirEnt,UA,ConfigHX,TLiqLvg,TAirLvg)     ntup = UA/capAir
    cRatiop = capAir/capLiq
    effect(cRatiop, ntup, effp)
    qRef = capAir*(TAirEnt-TLiqEnt)
    q = capAir*(TAirEnt-TLiqLvg)
    q = capLiq*(TLiqLvg-TLiqEnt)
    q = effp * qRef
    Q = capAir*(TAirEnt-TAirLvg)
    WAirLvg = WairEnt
    effect(cRatiop, ntup, effp)
    {
           if(cRatiop < 1.0){
      cRatio = cRatiop;
      ntu = ntup;
    }
    else{
      cRatio = 1.0/cRatiop;
      ntu = ntup * cRatiop;
    }
    if (ntu < SMALL)
      eff = 0.0;
    else if (fabs(cRatio) < SMALL)
      eff = 1.0 - exp(-ntu);
    else{
      e = (1.0+cRatio);
      eff = (1.0 - exp(-ntu*e)) / e;
    }
    if(cRatiop <= 1.0)
      effp = eff;
    else
      effp = eff/ cRatiop;
    }
---*/
port mAirEnt          "Air flow"                                    [kg_dryAir/s] ;
port mLiq             "Liquid flow"                                 [kg/s] ;
port TAirEnt          "Entering air dry bulb temperature"           [deg_C] ;
port TLiqEnt          "Entering water temperature"                  [deg_C] ;
port wAirEnt          "Entering air humidity  ratio"                [kg_water/kg_dryAir] ;
```

```
port mAirLvg     "Leaving air flow"                                   [kg_dryAir/s] ;
port wAirLvg     "Leaving air humidity  ratio"                        [kg_water/kg_dryAir] ;
port TAirLvg     "Leaving air Temperature"                            [deg_C] ;
port TLiqLvg     "Leaving water temperature"                          [deg_C] ;
port qSen        "Heat transfer rate.  Negative for air heating."     [W] ;
port CHx         "Constant of heat exchanger coefficient- air side";
port AHx          "Overall heat exchanger surface area - air side"     [m2];

declare equal_link eq1 eq2;

declare drcc1u Hx /*declare a cross flow heat exchange object*/ ;
link     .mAirEnt Hx.mAirEnt;
link     .mLiq    Hx.mLiq;
link    .TAirEnt  Hx.TAirEnt;
link    .TLiqEnt  Hx.TLiqEnt;
link    .wAirEnt  Hx.wAirEnt;
link              UA Hx.UA               eq1.a;
link     .mAirLvg Hx.mAirLvg             eq2.a;
link    .wAirLvg  Hx.wAirLvg;
link    .TAirLvg  Hx.TAirLvg;
link    .TLiqLvg  Hx.TLiqLvg;
link    .qSen     Hx.q;

declare UA UA;
link    .CHx      UA.C;
link    .AHx      UA.AreaHX;
link              mAir UA.m              eq2.b;
link     UA0      UA.UA                  eq1.b;
```

UA.cc Coil / SOURCE CODE

```
/* CLASS  UA          "UA value based on the flow rate and heat exchange area"

ABSTRACT
            ...
            ...
ABSTRACT_END
TEST_INPUT
      C = 2.3, m = 1.23, AreaHx =2 ;
*/
#ifdef SPARK_TEXT
// ==== PORTS ====

PORT    C               "constant of heat exchanger "          [scalar] ;
PORT    m               "mass flow rate"                       [scalar] ;
PORT    AreaHX   "heat exchange area"                          [m2] ;
PORT    UA              "UA"                                   [W/K] ;

EQUATIONS { UA = C*(m)^0.8*AreaHx ;
             }

// ==== FUNCTIONS ====
FUNCTIONS {
            UA          = UA_UA( C, m, AreaHX, UA ) ;
             }
#endif /* SPARK_TEXT */
#include "spark.h"

   double
UA_UA ( ARGS )
{
   ARGDEF(0,C) ;
   ARGDEF(1,m) ;
   ARGDEF(2,AreaHX) ;
   double UA;

   if (m < 0)
            cout<<"  error! m in UA.CC less than 0"  <<endl;
   else
            UA = C* pow (m,0.8) * AreaHX;

   return UA ;
}
```

General description

The most commonly used fans in large air handling units are centrifugal fans. In VAV systems, fan capacity is controlled by varying either the rotation speed or the position of an inlet guide vane. Return fans may be axial fans, controlled by varying either the rotation speed or the blade angle. In VAV systems, there is a pressure sensor in the supply duct and a feedback control loop to maintain the air pressure in the duct constant by adjusting the supply fan capacity. The model described here applies to VAV systems in which the capacity of each fan is controlled by varying the rotation speed.

Model description

The model treats either the supply fan or the return fan, together with the appropriate section of the distribution system (Figure 2). Fan performance is modeled by using the fan similarity laws to normalize the flow rate, pressure rise and power in terms of rotation speed and diameter. Over the limited range of normalized flow used in normal operation, the fan head curve can be approximated using a constant term and a squared term. The constant term is the pressure rise extrapolated to zero flow, which is proportional to the square of the rotation speed, and the squared term corresponds to the internal pressure drop inside the fan. The model is written in terms of total pressure (i.e. static pressure plus velocity pressure) since the energy losses are directly related to changes in total pressure.

$$p_{fan} = k_{fan} n_{fan}^2 - C_{fan} m_{air}^2$$

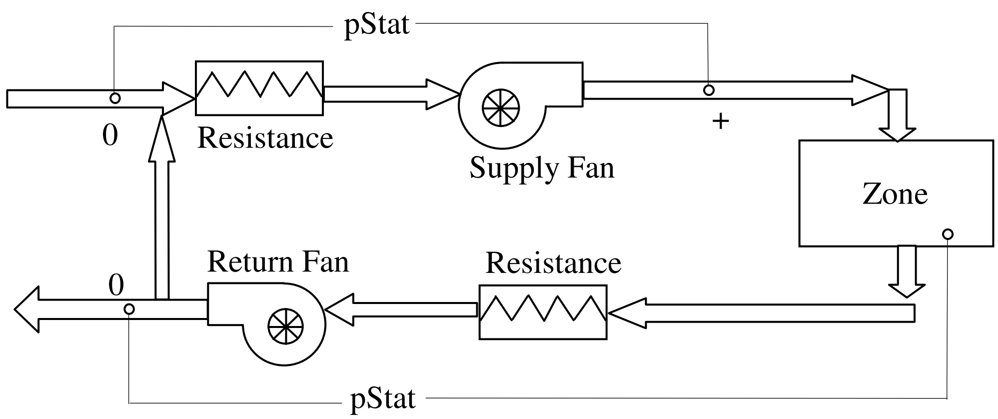

Figure 2:- System diagram of the fan-air system simulated in the model

The system curve, which represents the pressure drop through all the air handling unit (AHU) and distribution system components, also consists of a constant term and a squared term. For the supply fan subsystem, the constant term is the static pressure set-point. The squared term represents the pressure drop through the AHU and distribution system components and the velocity pressure at the static pressure sensor, both of which are proportional to the square of the air mass flow rate.

$$p_{Res} = p_{Stat} + (C_{Res} + 1/2\rho_{air}A^2)m_{air}^2$$

For the return fan subsystem, p_{stat} is the measured or assumed pressure in the occupied space and appears as a negative term, since a positive pressure in the space reduces the fan pressure rise required. The correction for the velocity pressure in the room is very small and can be ignored.

$$p_{Res} = -p_{Stat} + C_{Res}m_{air}^2$$

The fan operating point is where the pressure drop across the system equals the pressure increase across the fan, as shown in Figure 3. The air flowing through the fan increases in temperature because of the heat added to the air stream due to fan inefficiency and due to motor inefficiency, if the fan is in the air stream. Because air is a compressible fluid and can be treated as a perfect gas, it can be shown that the fluid work peformed by the fan results in the same temperature increase that would be obtained if the fluid work were completely converted to heat. The opposite is true for incompressible fluids, such as water. In the case of incompressible fluids, the fluid work only appears as heat when the fluid passes through a dissipative element.

The class of the fan-air system is ***fan_system.cm***. Atomic classes ***fan_qLoss.cc***, ***fan_effShaft.cc***, and ***fan_resistance.cc*** are the models of air heat gain, fan shaft efficiency and fan resistance respectively.

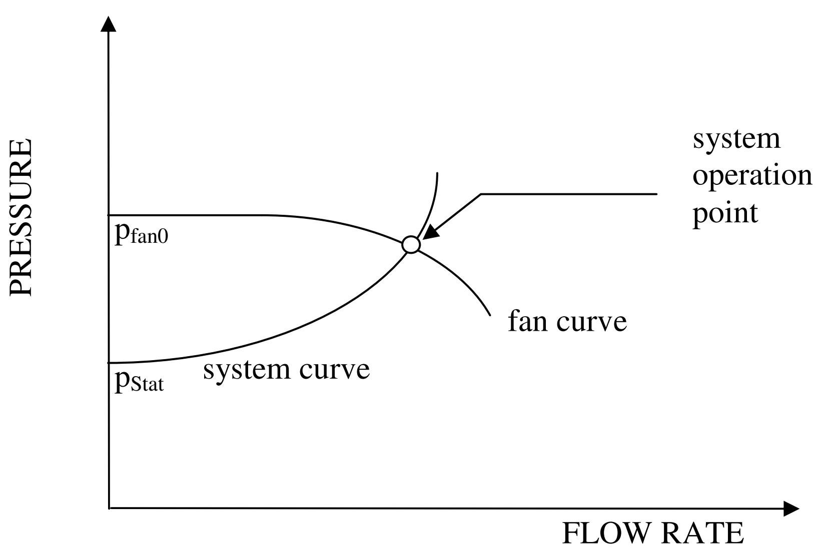

Figure 3 Schematic of the fan model

Governing equations:

Simplified fan curve:

$$p_{fan} = p_{fan0} - C_{fan} \cdot m_{air} \left| m_{air} \right|$$

System curve:

$$p_{res} = p_{stat} + (\frac{1}{2\rho_{air} A^2} + C_{res}) \cdot m_{air} \left| m_{air} \right|$$

System operation point constraint:

$$p_{fan} = p_{res}$$

Fan speed

$$p_{fan0} = k_{fan} \cdot n_{fan}^2$$

Combining all the above equations

$$k_{fan} n_{fan}^2 = p_{stat} + (\frac{1}{2\rho_{air} A^2} + C_{res} + C_{fan}) \cdot m_{air} \left| m_{air} \right|$$

Fan shaft power

$$W_s = \frac{m_{air} \cdot p_{fan}}{\eta_s \rho_{air}}$$

Total motor power

$$W_T = \frac{W_s}{\eta_{motor}}$$

Heat loss to the air stream

$$q_{loss} = W_s + (W_T - W_s) f_{motor,loss}$$

The fan efficiency as a function of air-flow rate is:

$$\eta_s = \eta_{s,\max} - C_\eta (\frac{m_{air}}{n_{fan}} - \frac{m_{air,\max}}{n_{fan}})^2$$

The internal heat gain and temperature rise cross the fan is determined from

$$(T_{air,out} - T_{air,in}) \cdot c_p \cdot m_{air} = q_{loss}$$

Nomenclature

Variables		Description	Unit
A	Area	duct cross section area	m^2
C_{fan}	CFan	fan curve constant	Dimensionless
C_{res}	CRes	resistance characteristic constant	Dimensionless
C_{η}	CEff	constant to calculate fan efficiency	Dimensionless
$T_{air,out}$	TAirOut	leaving air temperature	deg_C
$T_{air,in}$	TAirIn	entering air temperature	deg_C
$f_{motor,loss}$	MotFrac	fraction of motor heat loss entering air stream	Dimensionless
k_{fan}	kFan	pressure-fan speed constant	Dimensionless
m_{air}	mAir	air flow rate through the fan	kg/s
n_{fan}	nFan	fan speed	rpm
p_{fan}	pFan	pressure increase cross the fan	Pa
p_{fan0}	pFan0	baseline fan pressure increase	Pa
p_{res}	pRes	load pressure drop	Pa
q_{loss}	qLoss	air stream heat gain from fan	W
W_s	powerShaft	shaft power	W
W_T	powerMot	total motor power	W
η_s	effShaft	fan efficiency	Dimensionless
$\eta_{s,max}$	effShaftMax	maximum fan efficiency	Dimensionless
η_m	effMot	motor efficency	Dimensionless

```
/*+++
/*+++
 Identification: fan model

 Abstract:
     The fan curve can be treated by simplified model:
             pFan = pFan0 - CFan * m^2
             where pFan0 is directly related to fan speed, where is
             pFan = kFan * nFan^2
     The resistance is:
             pRes = pStat + (vAir/(2*area^2)+CRes )* m^2
     Given a fan-resistance system
             pFan = pRes, therefore
             pFan0 = pStat + (vAir/(2*area^2)+ CFan + CRes) * m^2

 Notes:
     None

 Interface:
             nFan:      fan speed                                   [rpm];
             pStat:     static pressure setpoint                    [Pa] ;
             pFan:      total pressure increase across fan          [Pa];
             mAir:      air flow rate through the fan               [kg_dryAir/s];
             CRes:      resistance charactristic constant           [scalar];
             CFan:      fan curve constant                          [scalar];
             kFan:      pressure-fanspeed constant                  [scalar];
             CEff:      fan effiency constant                       [scalar];
             area:      duct work crossing section area             [m2];
             TAirEnt:   Incoming air temperature                    [J/kg_dryAir] ;
             wAirEnt:   Incoming air humidity ratio                 [kg/kg_dryAir];
             TAirLvg:   Outgoing air temperature                    [J/kg_dryAir] ;
             wAirLvg:   Incoming air humidity ratio                 [kg/kg_dryAir];
             powerTot:Power consumption                             [W] ;
             effMot:    Efficiency of fan motor                     [scalar] ;
             motFrac:   Fraction of motor heat loss in air stream[fraction] ;
             effShaft:   fan efficiency                             [scalar];
             effShaftMax: fan maximum efficiency                    [scalar];
             mAirMax:  maximum air flow of the fan                  [kg_dryAir/s];
             PAtm:      Atmospheric pressure                        [Pa];

 Acceptable input set:

             nFan:              unknown                             [rpm];
             pStat:             20                                  [Pa] ;
             pFan:              unknown                             [Pa];
             mAir:              5                                   [kg_dryAir/s];
             CRes:              0.1                                 [scalar];
             CFan:              0.3                                 [scalar];
             kFan:              1.25E-3                             [scalar];
             CEff:              1e-4                                [scalar];
             area:              0.3                                 [m2];
             TAirEnt:           20                                  [J/kg_dryAir] ;
             wAirEnt:           0.08                                [kg/kg_dryAir];
             TAirLvg:           unknown                             [J/kg_dryAir] ;
             wAirLvg:           unknown                             [kg/kg_dryAir];
             powerTot:          unknown                             [W] ;
             effMot:            0.9                                 [scalar] ;
             motFrac:           1                                   [scalar] ;
             effShaft:          unknown                             [scalar];
             effShaftMax:       0.9                                 [scalar];
             mAirMax: 8                                     [kg_dryAir/s];
             PAtm:              1e5                                 [Pa];

 Recommended matches:
       None

 Suggested breaks:
       None
```

```
 Equations:
        kFan * nFan^2 = pStat + ((1/(2*density_air*area^2)) + CRes + CFan ) * mAir^2 ;
        pFan = kFan* nFan^2 - CFan  * mAir^2;
        effShaft = effShaftMax - CEff*((mAir-mAirMax)/nFan)^2;
        powerShaft = mAir * PFan / effShaft *vAir;
        powerTot = powerShaft / effMot;
        qLoss = (powerShaft)+(powerTot-powerShaft)*motFrac;
        (TAirLvg-TAirEnt)*mAir*Cp = qLoss;
---*/

PORT     nFan  "fan speed "                                   [rpm];
PORT     pStat "static pressure setpoint "                     [Pa] ;
PORT     pFan  "total pressure increase across fan"            [Pa];
PORT     mAir  "air flow rate through the fan "                [kg_dryAir/s];

PORT    CRes  "resistance charactristic constant"             [scalar];
PORT    CFan "fan curve constant"                              [scalar];
PORT     kFan "pressure-fanspeed constant"                     [scalar];
PORT    CEff "fan effiency constant"                           [scalar];
PORT    area "duct work crossing section area"                 [m2];

PORT    TAirEnt    "Incoming air temperature"                  [J/kg_dryAir] ;
PORT    wAirEnt "Incoming air humidity ratio"                  [kg/kg_dryAir];
PORT    TAirLvg    "Outgoing air temperature"                  [J/kg_dryAir] ;
PORT    wAirLvg "Incoming air humidity ratio"                  [kg/kg_dryAir];
PORT    powerTot "Power consumption."                          [W] ;
PORT    effMot      "Efficiency of fan motor"                  [scalar] ;
PORT    motFrac   "Fraction of motor heat loss in air stream"  [fraction] ;
PORT    effShaft "fan efficiency"                              [scalar];
PORT    effShaftMax "fan maximum efficiency"                   [scalar];
PORT    mAirMax  "maximum air flow of the fan"                 [kg_dryAir/s];
PORT    PAtm       "Atmospheric pressure"                      [Pa];

//LINKS
declare equal_link eq1 eq2 eq3 eq4 eq5 eq6 eq7 eq8 eq9 eq10 eq11 eq12 eq13 eq14 eq15 eq16 eq17 eq18 eq19 eq20;

//kFan * nFan^2 = pStat + ((1/(2*density_air*area^2)) + CRes + CFan ) * mAir^2
declare fan_resistance FR;
link     .nFan            FR.nFan                 eq8.a eq9.a;
link    .pStat            FR.pStat ;
link    .mAir             FR.mAir                 eq5.a eq11.a;
link    .CRes             FR.CRes  ;
link    .CFan             FR.CFan                 eq20.a;
link     .kFan            FR.kFan                 eq10.a;
link     .area            FR.area ;
link     vAir             FR.vAir                 eq1.a;

//pFan = kFan * nFan^2 -CFan * mAir^2
declare safprod pdA pdB pdC pdD;
declare sum sumA;
link    kFan1             pdA.a                   eq10.b;
link    nFan2             pdB.b pdB.a             eq9.b;
link    nFanSquare        pdA.b pdB.c;
link    pFan0             pdA.c sumA.c;
link    CFan              pdC.a                   eq20.b;
link    mAir4             pdD.b pdD.a             eq11.b;
link    mAirSquare        pdC.b pdD.c;
link    CFanmAirSquare    pdC.c sumA.a;
link     .pFan            sumA.b                  eq12.a;


//effShaft = effShaftMax - CEff*((mAir-mAirMax)/nFan)^2
declare fan_effShaft ES;
link    .effShaft         ES.effShaft             eq3.a;
link    .effShaftMax      ES.effShaftMax;
link    mAir              ES.mAir                 eq5.b eq6.a;
link    .mAirMax ES.mAirMax;
```

```
link	nFan		ES.nFan			eq8.b;
link	.CEff		ES.CEff;

//powerShaft = mAir * PFan / effShaft *vAir
declare safprod pd1 pd2;
declare safquot sq1;
link	mAir1		pd1.a			eq6.b eq7.a;
link	pFan		pd1.b			eq12.b;
link	mAirPFan	pd1.c pd2.a;
link	vAir2		pd2.b			eq1.b eq2.a;
link	mAirpFanvAir	pd2.c sq1.a;
link	effShaft	sq1.b			eq3.b eq4.a;
link	powerShaft	sq1.c			eq13.a;

//powerTot = powerShaft / effMot
declare safquot sq;
link	powerShaft1	sq.a			eq13.b eq14.a;
link	.effMot		sq.b;
link	.powerTot	sq.c			eq15.a;

//qLoss = (powerShaft)+(powerTot-powerShaft)*motFrac
declare fan_qLoss qL;
link	powerShaft2	qL.powerShaft		eq14.b;
link	effShaft1	qL.effShaft		eq4.b;
link	powerTot	qL.powerTot		eq15.b;
link	.motFrac	qL.motFrac;
link	 qLoss1		qL.qLoss		eq19.a;

//(TAirLvg-TAirEnt)*mAir*Cp = qLoss
declare enthalpy en1 en2;
link	.TAirEnt	en1.TDb			eq17.a;
link	.wAirEnt	en1.w			eq16.a;
link	.TAirLvg	en2.TDb ;
link	.wAirLvg	en2.w			eq16.b eq18.a;
declare sum sum1;
link	hAirLvg		sum1.c en2.h ;
link	hAirEnt		sum1.a en1.h ;
declare safprod pd4;
link	hAirIncrease	sum1.b pd4.a;
link	mAir2		pd4.b			eq7.b;
link	qLoss		pd4.c			eq19.b;

//specific volume of the air
declare specvol sv;
link	.PAtm		sv.PAtm;
link	TAirLvg1	sv.TDb			eq17.b;
link	wAirLvg1	sv.w			eq18.b;
link	vAir1		sv.v			eq2.b;
```

```
/*+++
 Identification: fan model using the simplified method.

 Abstract:
     The fan curve can be treated by simplified model:
             pFan = pFan0 - CFan * m^2
             where pFan0 is directly related to fan speed, where is
             pFan = kFan * nFan^2
     The resistance is:
             pRes = pStat + (vAir/(2*area^2)+CRes )* m^2
     Given a fan-resistance system
             pFan = pRes, therefore
             pFan0 = pStat + (vAir/(2*area^2)+ CFan + CRes) * m^2

 Notes:
     None

 Interface:
    nFan:        fan speed                              [rpm]
    pFan:        pressure increase cross the fan        [Pa]
    pStat:       static pressure setpoint               [Pa]
    mAir:        air flow rate through the fan          [kg/s]
    CRes:        resistance charactristic constant      [scalar]
    CFan:        fan curve constant                     [scalar]
    kFan:        pressure-fanspeed constant             [scalar]
    vAir:        Air specific volume                    [m^3/kg_dryAir]

 Acceptable input set:
    area:        0.3
    pStat:       20
    mAir:        2
    CRes:        0.1
    CFan:        0.3
    kFan:        1.25e-3
    vAir:        1.0

 Recommended matches:
       None

 Suggested breaks:
       None

 Local variables:
             pRes:  pressure resistance  [Pa]

 Equations:

             kFan * nFan^2 = pStat + ((1/(2*density_air*area^2)) + CRes + CFan ) * mAir^2 ;

---*/

#ifdef SPARK_TEXT

PORT     nFan  "fan speed "                                    [rpm];
PORT     pStat "static pressure setpoint "                     [Pa] ;
PORT     mAir  "air flow rate through the fan "                [kg/s]
                  INIT = 2.0 ;
PORT     CRes  "resistance charactristic constant"             [scalar];
PORT     CFan "fan curve constant"                             [scalar];
PORT     kFan "pressure-fanspeed constant"                     [scalar];
PORT     area "duct work crossing section area"                [m2];
PORT     vAir       "Specific volume"                          [m^3/kg_dryAir] ;

EQUATIONS {
             kFan * nFan^2 = pStat + ((1*vAir/2*area^2) + CRes + CFan ) * mAir^2 ;
              }

// ==== FUNCTIONS ====
```

```
FUNCTIONS {
            nFan       = fan_sys_nFan( pStat, mAir, area, vAir, CRes, CFan, kFan ) ;
            pStat      = fan_sys_pStat(nFan, mAir, area, vAir, CRes, CFan, kFan ) ;
            mAir       = fan_sys_mAir( pStat, nFan, area, vAir, CRes, CFan, kFan ) ;
             }
#endif /* SPARK_TEXT */
#include "spark.h"

   double
fan_sys_nFan ( ARGS )
{
   ARGDEF(0,pStat) ;
   ARGDEF(1,mAir) ;
   ARGDEF(2,area) ;
   ARGDEF(3,vAir) ;
   ARGDEF(4,CRes) ;
   ARGDEF(5,CFan) ;
   ARGDEF(6,kFan) ;

   double nFan;
   nFan = pow (((pStat + ((vAir/(2*area*area)) + CRes + CFan ) * mAir*mAir)/kFan),0.5)  ;

   return nFan ;
}

   double
fan_sys_pStat ( ARGS )
{
   ARGDEF(0,nFan) ;
   ARGDEF(1,mAir) ;
   ARGDEF(2,area) ;
   ARGDEF(3,vAir) ;
   ARGDEF(4,CRes) ;
   ARGDEF(5,CFan) ;
   ARGDEF(6,kFan) ;

   double pStat;
   pStat = kFan * nFan*nFan - ((vAir/(2*area*area)) + CRes + CFan ) * mAir*mAir ;

   return pStat ;
}

   double
fan_sys_mAir ( ARGS )
{
   ARGDEF(0,pStat) ;
   ARGDEF(1,nFan) ;
   ARGDEF(2,area) ;
   ARGDEF(3,vAir) ;
   ARGDEF(4,CRes) ;
   ARGDEF(5,CFan) ;
   ARGDEF(6,kFan) ;

   double mAir;
   if ( kFan * nFan*nFan - pStat >=0)
            mAir =pow(( (kFan * nFan*nFan - pStat ) /  ((vAir/(2*area*area)) + CRes + CFan )  ), 0.5 );
   else
            cout<<"error! kFan*nFan*nFan less than pStat" <<endl;

  return mAir ;
}
```

```
/*+++
  Identification: fan heat gain.
  Abstract:

  Notes:
      None

  Interface:
     qLoss:          heat gain of the air stream through fan              [W]
     powerShaft:     fan shaft power                                      [W]
     effShaft:        fan efficiency                                      [kg/s]
     power:          Total motor power consumption                        [kg/s]
     motFrac:         Fraction of motor heat loss in air stream           [fraction] ;

  Acceptable input set:
     qLoss:          unknown            [W]
     powerShaft:     100                [W]
     effShaft:       0.8                [kg/s]
     power:          120                [kg/s]
     motFrac:         1                 [fraction] ;

  Recommended matches:
        None
  Suggested breaks:
        None
  Local variables:
  Equations:
            qLoss = (powerShaf)+(powerTot-powerShaft)*motFrac;

---*/

#ifdef SPARK_TEXT

port      qLoss        "heat gain of the air stream through fan"        [W];
port      powerShaft "fan shaft power"                                 [W];
port      effShaft     "fan efficiency"                                 [kg/s];
port      powerTot    "Total motor power consumption"                  [kg/s];
port      motFrac      "Fraction of motor heat loss in air stream"      [fraction] ;


EQUATIONS {
            qLoss = (powerShaft - powerShaft*effShaft)+(powerTot-powerShaft)*motFrac;
             }

// ==== FUNCTIONS ====
FUNCTIONS {
            qLoss = fan_qLoss_qLoss( powerShaft, effShaft, powerTot, motFrac) ;
             }
#endif /* SPARK_TEXT */
#include "spark.h"

   double
fan_qLoss_qLoss ( ARGS )
{
   ARGDEF(0,powerShaft) ;
   ARGDEF(1,effShaft) ;
   ARGDEF(2,powerTot) ;
   ARGDEF(3,motFrac) ;

   double qLoss;
   qLoss = (powerShaft)+(powerTot-powerShaft)*motFrac  ;

   return qLoss;
}
```

```
/*+++
 Identification: fan shaft efficiency model.
 Abstract:
 Notes:
     None

 Interface:
    effShaft:        fan efficiency                          [scalar]
    effShaftMax:     maximum fan efficiency                  [scalar]
    mAir:            air flow rate through the fan           [kg/s]
    mAirMax:         maximum air flow rate through the fan   [kg/s]
    CEff:            fan effiency constant                   [scalar]
    nFan:            fan speed                               [rpm]

 Acceptable input set:
    effShaft:        unknown                                 [scalar]
    effShaftMax:     0.98                                    [scalar]
    mAir:            1                                       [kg/s]
    mAirMax:         1.5                                     [kg/s]
    CEff:            0.2                                     [scalar]
    nFan:            1000                                    [rpm]

 Recommended matches:
       None

 Suggested breaks:
       None

 Local variables:

 Equations:
          effShaft = effShaftMax - CEff*((mAir-mAirMax)/nFan)^2;
---*/
#ifdef SPARK_TEXT

PORT     nFan  "fan speed "                                       [rpm];
PORT     effShaft  "fan efficiency"                               [scalar];
PORT     effShaftMax "maximum fan efficiency"                     [scalar];
PORT     mAir "air flow rate through the fan"                     [kg/s];
PORT     mAirMax "maximum air flow rate through the fan"          [kg/s];
PORT     CEff "fan effiency constant"                             [scalar];

EQUATIONS {
          effShaft = effShaftMax - CEff*((mAir-mAirMax)/nFan)^2 ;
           }

// ==== FUNCTIONS ====
FUNCTIONS {
          effShaft = fan_effShaft_effShaft( effShaftMax, CEff, mAir, mAirMax, nFan) ;
           }
#endif /* SPARK_TEXT */
#include "spark.h"

   double
fan_effShaft_effShaft ( ARGS )
{
   ARGDEF(0,effShaftMax) ;
   ARGDEF(1,CEff) ;
   ARGDEF(2,mAir) ;
   ARGDEF(3,mAirMax) ;
   ARGDEF(4,nFan) ;

   double effShaft;
   if( (mAir-mAirMax)<0 )
          effShaft = effShaftMax - CEff* pow( ((mAir-mAirMax)/nFan), 2)  ;
   else
          cout<<"  error! mAirMax less than mAir"  <<endl;
   return effShaft ;
}
```

Control valve

General description
A control valve varies the fluid flow rate in a circuit by varying its flow resistance. An external actuator is used to move a plug connected to the valve stem that restricts the flow to varying degrees depending on its position. There are three distinct valve flow types based on the geometry of the plug: quick opening, linear, and equal percentage. The equal percentage characteristic is used to compensate for the non-linear characteristic of heating and cooling coils and the effect of the series resistance of the coil.

The most common faults associated with control valves are: leakage, stuck valve/actuator, actuator/valve range mismatch and unstable control. In order to detect these faults, it is more important to model the valve behavior at each end of the operation than in the middle. However, as discussed in the Coil section, it is desirable to be able to predict the part load performance of coils in order to anticipate loss of peak capacity before it occurs. Since the water flow rate through a coil is not generally measured in HVAC systems, it is necessary to treat the behavior of the control valve at intermediate flow rates by modeling its inherent and installed characteristics in order to predict the water flow rate through the coil.

Model description
The water flow rate is a function of the valve position, the flow rate through the valve when fully open and the leakage. The flow characteristic is assumed to be parabolic, which is an adequate and convenient approximation to the equal percentage characteristic.

In order to model the installed characteristic of the valve, it is necessary to treat the effect of the series resistance of the coil and other components in the branch. This is conventionally expressed in terms of the authority of the valve. Authority is the ratio of the pressure drop across the valve when it is fully open to the pressure drop across the whole of the branch when the valve is fully open. When the authority is equal to unity, the pressure drop across the valve dominates the pressure drop in the branch and there is no distortion of the valve flow characteristics curve. When the authority is equal to zero, the pressure drop across the valve is negligible unless it is fully closed and so the valve has essentially no effect on the flow rate except when it is fully closed. A more detailed description of valve authority is given in the ASHRAE Handbook (HVAC Systems and Equipment, p41.7, 1996)

The model described here is a combination of the valve model in the ASHRAE Secondary Toolkit and relationships given in the ASHRAE Handbook. The class of the valve model is ***valve.cc***.

Governing equations

$$leak_{par} = \frac{m_{leak}}{m_{liq,open}}$$

$$f_{inher} = (1 - leak_{par}) \cdot pos^2 + leak_{par}$$

$$f_{install} = \frac{1}{\sqrt{\frac{a}{f_{inher}^2} + (1-a)}} \qquad (f_{inher} \neq 0)$$

$$f_{install} = 0 \qquad (f_{inher} = 0)$$

$$m_{liq} = f_{install} \cdot m_{liq,open}$$

Nomenclature

Variables		Description	Unit
A	A	Valve authority, between 0-1	Dimensionless
$leak_{par}$	Leakpar	Leakage parameter	Dimensionless
m_{liq}	mLiq	Mass flow rate	kg/s
$m_{liq,open}$	mLiqOpen	Mass flow rate for open valve	kg/s
m_{leak}	mLeak	Mass flow rate with closed valve	kg/s
pos	pos	Valve position, between 0-1	Dimensionless
$f_{install}$	fInstall	Installed flow rate factor	Dimensionless
f_{inher}	fInher	Inherit valve resistance ratio (valve resistance divided by valve resistance at full open)	Dimensionless

```
/*+++
  Identification: Flow circuit with non-linear/square valve and series flow
                              resistance.

  Abstract:

  Notes:
      None

  Interface:
     pos:     Valve position         (-)
     mLiq:    Mass flow rate         [Kg/s]
     A:       Valve authority        (-)
     mLiqOpen:Mass flow rate for open valve [kg/s]
     mLeak:   Mass flow rate for closed valve [Kg/s]

  Acceptable input set:
      pos = 0.5, A = 0.5, mLiqOpen = 1,  wf = 0.5, leak = 0.05

  Recommended matches:
        None

  Suggested breaks:
        None

  Local variables:
     Leakpar: Fraction of mLeak to mOpen
     fInher:  inherited valve resistance ratio
     fInstall: Installed flow rate factor

  Equations:
             Leakpar = mLeak/mLiqOpen;
             fInher = (1-Leakpar)*pos^2 + Leakpar) ;
             fInstall = 1/ (a/(fInher^2) + (1-a) )^0.5 (fInher !=0)
                        = 0 (fInher = 0);
             mLiq = mLiqOpen *fInstall;
---*/

#ifdef SPARK_TEXT
// ==== PORTS ====
port pos            "Valve position, between 0-1"          [scalar] ;
port mLiq           "Mass flow rate"                       [Kg/s];
port A              "Valve authority, between 0-1"         [scalar] ;
port mLiqOpen       "Mass flow rate for open valve"        [Kg/s] ;
port mLeak          "Fraction of m_open for closed valve"  [Kg/s] ;


EQUATIONS { mLiq = valve_mLiq (pos, A, mLiqOpen, mLeak) ;
              }

// ==== FUNCTIONS ====
FUNCTIONS {
             mLiq      = valve1_mLiq( pos, A, mLiqOpen, mLeak ) ;
             pos       = valve1_pos ( mLiq, A, mLiqOpen, mLeak ) ;
              }
#endif /* SPARK_TEXT */
#include "spark.h"

   double
valve1_mLiq ( ARGS )
{
   ARGDEF(0,pos) ;
   ARGDEF(1,A) ;
   ARGDEF(2,mLiqOpen) ;
   ARGDEF(3,mLeak) ;

   double Leakpar;
   double fInher ;
```

```
   double fInstall;
   double mLiq;

   Leakpar = mLeak/mLiqOpen;
   fInher = (1-Leakpar)*pos*pos + Leakpar ;

   if (fInher !=0)
   fInstall = 1/ pow ( (A/(fInher*fInher) + (1-A) ), 0.5 );
   else
   fInstall = 0;

   mLiq = mLiqOpen *fInstall;

   return mLiq;
}

 double
valve1_pos ( ARGS )
{
   ARGDEF(0,mLiq) ;
   ARGDEF(1,A) ;
   ARGDEF(2,mLiqOpen) ;
   ARGDEF(3,mLeak) ;

   double Leakpar;
   double fInher ;
   double fInstall;
   double pos;

   Leakpar = mLeak/mLiqOpen;
   fInstall = mLiq / mLiqOpen ;
   if (fInstall == 0)
      pos =0;
   else
            {
                        if (fInstall >1.0)
                                    cout<<"error! mLiq is larger than mLiqOpen"<<endl;
                        else
                        fInher  =pow( A / (1/ (fInstall*fInstall) - (1-A)), 0.5) ;

                        if (fInher < Leakpar)
                                    cout<<"error! mLeak is larger than mLiq"<<endl;
                        else
                  pos = pow ((fInher - Leakpar) /  (1-Leakpar), 0.5)  ;
            }
   return pos;
}
```

General description
A mixing box is the section of an air handling unit used to mix the return air flow with the outside air flow. It consists of three sets of dampers whose operation is coordinated to control the fraction of the outside air in the supply air while maintaining the supply air-flow rate approximately constant. Figure 2 is a simplified diagram of the mixing box simulated in the model. A variant of this design has a separate outside air damper that is adjusted to provide the minimum outside air flow required during occupancy. In an ideal mixing box (no damper leakage), the mixed air should consist of 100% return air when the control signal is 0 and consist of 100% outside air when the control signal is 1.

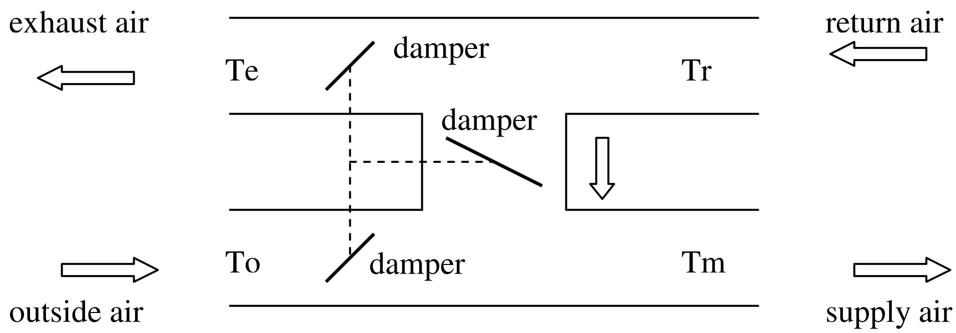

Figure 4 Diagram of the mixing box

Figure 5 shows ideal behavior and the range of acceptable behavior of a mixing box. The vertical axis is the outside air fraction. Under the ideal conditions, the outside air fraction should range from 0 to 1 when the damper position varies from 0 to 1. However, in general there is leakage of both the outside air and the return air dampers; the outside air fraction then ranges between a minimum value that is greater than 0 and a maximum value that is less than 1. In addition, the air-flow rate is not necessarily linearly related to the damper position and therefore the mixed air temperature and humidity ratio are not linearly related to damper position. After the mixing box has been commissioned, the results of the functional test can be used to calibrate a model of the actual behavior.

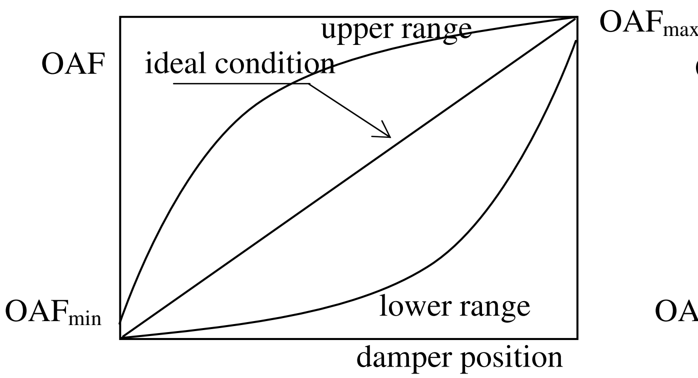

Figure 5 mixing box design curves

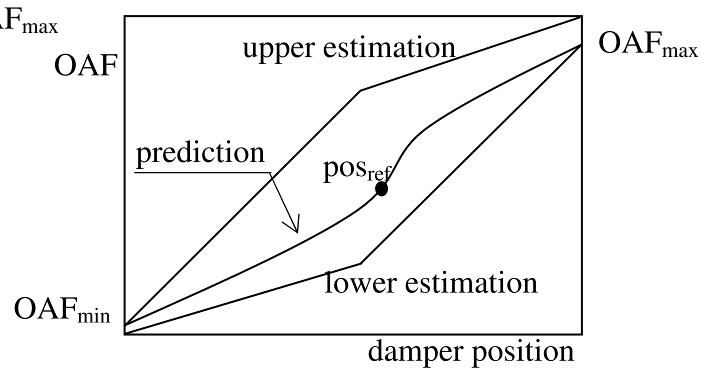

Figure 6 mixing box model curves

Model description

Theoretically, it is possible to determine the airflow rates of both the outside air and recirculation air streams as a function of damper position. The airflow rates can then be used to determine the outside air fraction and hence the mixed air temperature and humidity ratio. However, it is impractical to simulate the mixed air temperature accurately in that way, because the pressure boundary conditions change with fan speed and as a result of wind effects and because of the difficulty of estimating the authority of the dampers. This said, the behavior in the middle of the operating range is relatively unimportant compared to the behavior at the ends of the operating range.

Figure 4 shows the forms of the models to be used during commissioning and during routine operation following commissioning. In the model to be used at the commissioning stage, when only design information is available, the range of acceptable behavior is modeled. A 3:1 gain variation is used by default; when the damper position is 50%, the upper limit of the outside air fraction is 25% lower than its maximum and the lower limit is 25% above its minimum. The maximum acceptable deviations from 0 and 100% outside air fraction at each end of the operating range should be specified by the designer. Once the mixing box has been commissioned, the results of the functional test can be used to fit curves to the measured variation of outside air fraction with the control signal. There are two ways to fit into this curve, one is by simple polynomial, another one is by a more complex method that involved with a middle point representing where the curve reflects and an exponential constant (see equations below). In VAV systems, this relationship may depend on supply air-flow rate. If it is significant, this dependence may be treated by fitting two polynomials, one for maximum supply air flow rate and one for minimum supply air flow rate, and using these two polynomials to define the range of expected behavior.

Mixing box REFERENCE MODELS

The class ***mix.cm*** is the model of the mixing box. There are three mixed air temperature outputs, the upper and lower estimates of the mixed air temperature, and the predicted mixed air temperature by polynomial curve fitting. The atomic class ***OAFLow.cc*** is to predict the lower acceptable range of the mixed air temperature; the class ***OAFHigh.cc*** is to predict the upper range of the mixed air temperature; the class ***OAF.cc*** is the simulation model to predict the outside air fraction by 3rd order polynomial fitting. Atomic class ***tmix.cc*** models the mixed air temperature based on the outside air fraction.

Governing equations

Minimum and maximum of the outside air fraction:

$$OAF_{\min} = leak_{out}$$

$$OAF_{\max} = 1 - leak_{ret}$$

Upper and lower limit of the outside air fraction:

$$OAF_{lower} = \begin{cases} 2 \cdot pos \times OAF_{half} & (pos < 0.5) \\ 2 \cdot (pos - 0.5) \times (OAF_{\max} - OAF_{half}) + OAF_{half} & (pos > 0.5) \end{cases}$$

$$OAF_{higher} = \begin{cases} 2 \cdot pos \times (OAF_{half} - OAF_{\min}) + OAF_{\min} & (pos < 0.5) \\ 2 \cdot (pos - 0.5) \times (1 - OAF_{half}) + OAF_{half} & (pos > 0.5) \end{cases}$$

Predicted outside air fraction by polynomial curve fitting

$$OAF_{predic} = (OAF_{\max} - OAF_{\min})(C_1 pos + C_2 pos^2 + C_3 pos^3) + OAF_{\min}$$

Polynomial coefficients are related by following constraint:

$$C_1 + C_2 + C_3 = 1$$

Predicted outside air fraction by two exponential curves fitting linked at reference position

$$OAF_{predic} = OAF_{\min} + (OAF_{\max} - OAF_{\min})\left(\frac{pos_{ref}{}^{n} + (pos - pos_{ref})^{n}}{pos_{ref}{}^{n} + (1 - pos_{ref})^{n}}\right) \qquad (pos > pos_{ref})$$

$$OAF_{predic} = OAF_{\min} + (OAF_{\max} - OAF_{\min})(\frac{pos_{ref}{}^{n} - (pos_{ref} - pos)^{n}}{pos_{ref}{}^{n} + (1 - pos_{ref})^{n}}) \qquad (pos \le pos_{ref})$$

Mixed air temperature:

$$T_{mix,predic} = OAF_{predic} \cdot (T_{out} - T_{ret}) + T_{ret}$$

$$T_{mix,lower} = OAF_{lower} \cdot (T_{out} - T_{ret}) + T_{ret}$$

$$T_{mix,higher} = OAF_{higher} \cdot (T_{out} - T_{ret}) + T_{ret}$$

Nomenclature

Variables		Description	Unit
$leak_{ret}$	LeakRet	Installed return damper leakage (0-1)	Dimensionless
$leak_{out}$	LeakOut	Installed outside air damper leakage (0-1)	Dimensionless
pos	pos	Valve damper position(-) (0 to 1, 1 = 100% outside air, 0 = 100% return air)	Dimensionless
OAF	OAF	Outside air fraction	Dimensionless
OAF_{half}	OAFHalf	Outside air fraction when damper position is 0.5	Dimensionless
OAF_{min}	OAFMin	Minimum outside air fraction	Dimensionless
OAF_{max}	OAFMax	Maximum outside air fraction	Dimensionless
T_{ret}	TRet	Return air temperature	^oC
T_{out}	TOut	Out air temperature	^oC
$T_{mix,lower}$	TMixLow	Lower range of the mixed air temperature	^oC
$T_{mix,higher}$	TMixHigh	Upper range of the mixed air temperature	^oC
$T_{mix,predic}$	TMix	Predicted mixed air temperature"	^oC
C_1	C1	Polynomial constant 1 for curve fitting outside air fraction as a function of damper position	
C_2	C2	Polynomial constant 2 for curve fitting outside air fraction as a function of damper position	
C_3	C3	Polynomial constant 3 for curve fitting outside air fraction as a function of damper position	
n	n	Exponential constant in two exponential curve fitting	>0, Real number
pos_{ref}	RefPos	Reference position where the two curves reflect each other (Figure 6) (0-1)	Scalar

```
/*+++
 Identification: mixing air temperature
 Abstract:
 Notes:
     None

 Interface:
    pos:           damper position(-) "0 to 1, 1 = 100% outside air, 0 = 100% return air "  [scalar]
    TRet:          Return air temperature                                                     [deg_C]
    TOut:          Outside air temperature                                                    [deg_C]
    TMix:          mixing air temperature                                                     [deg_C]
    TMixHigh:      Lower estimation of mixing air temperature                                 [deg_C]
    TMixLow:       Higher estimation of mixing air temperature                                [deg_C]
    LeakRet:       installed return damper leakage 0-1                                        [scalar]
    LeakOut:       outside air damper leakage 0-1                                             [scalar]


 Acceptable input set:
     pos = 0, TRet = 20, TOut =30, LeakRet =0.01, LeatOut=0.01

 Recommended matches:
       None

 Suggested breaks:
       None

 Local variables:
    OAF:           Outside air fraction (0-1)
    OAFHigh:       High estimation of outside air fraction (0-1)
    OAFLow:        Low estimation of outside air fraction (0-1)
    OAFHalfHigh:   High estimation of outside air fraction when damper position equals to 0.5.
    OAFHalfLow:    Low estimation of outside air fraction when damper position equals to 0.5.
    OAFMax:        Maximum outside air fraction when damper postion equals to 1. (Leakage from return air damper)
    OAFMin:        Minimum outside air fraction when damper postion equals to 0. (Leakage from outside air damper)

 Equations:
           TMix = OAF * (TOut-TRet) + TRet;
            OAFMax = 1-LeakRet;
           OAFMin = LeakOut ;
           OAF = (OAFMax-OAFMin)* (C1*pos+C2*pos^2+C3*pos^3) + OAFMin

           OAFHalfHigh = OAFMin + 0.75*(OAFMax -OAFMin)
           OAFHigh = (OAFHalf-OAFMin) * (pos*2) + OAFMin                  ( pos <= 0.5)
           OAFHigh = (1 - OAFHalf) * ((pos -0.5)*2) + OAFHalfHigh         ( pos > 0.5)

           OAFHalfLow = OAFMin + 0.25*(OAFMax -OAFMin)
           OAFLow = OAFHalf * (pos*2)                                     ( pos <= 0.5)
           OAFLow = (OAFMax - OAFHalfLow) * ((pos -0.5)*2) + OAFHalfLow   ( pos > 0.5)
*/
//PORT
PORT    pos       "damper position(-) (0 to 1, 1 = 100% outside air, 0 = 100% return air) " ;
PORT    TRet      "Return air temperature"                                                   [deg_C];
PORT    TOut      "Outside air temperature"                                                  [deg_C]   ;
PORT    TMix      "mixing air temperature"                                                   [deg_C];
PORT    TMixHigh "Lower estimation of mixing air temperature"                                [deg_C];
PORT    TMixLow  "Higher estimation of mixing air temperature"                               [deg_C];
PORT    LeakRet  "installed return damper leakage 0-1"                                       [scalar];
PORT    LeakOut  "installed outside air damper leakage 0-1"                                  [scalar];
PORT      C1      "polynomial constant 1 for curve fitting the outside air fraction (C1+C2+C3=1)"  [scalar];
PORT      C2      "polynomial constant 2 for curve fitting the outside air fraction (C1+C2+C3=1)"  [scalar] ;
```

mix.cm Mix / SOURCE CODE

```
PORT      C3      "polynomial constant 3 for curve fitting the outside air fraction (C1+C2+C3=1)"  [scalar] ;
```

```
declare equal_link eq1 eq2 eq3;

//Mixed Air temperature
//predicted mixed air temperature
declare tmix tmix;
link     OAF            tmix.OAF                              eq1.a;
link     .TMix          tmix.TMix;

//Lower estimation of  mixed air temperature
declare tmix tmixLow;
link     OAFLow         tmixLow.OAF                           eq2.a;
link     .TmixLow       tmixLow.TMix;

//Higher estimation of  mixed air temperature
declare tmix tmixHigh;
link     OAFHigh        tmixHigh.OAF                          eq3.a;
link     .TMixHightmixHigh.TMix;

link     .TOut          tmix.TOut tmixLow.TOut    tmixHigh.TOut;
link     .TRet          tmix.TRet tmixLow.TRet    tmixHigh.TRet;

//Outside air fraction
//Predicted outside Air Fraction
declare OAF        OAF;
link     OAF0           OAF.OAF                               eq1.b;
link     .C1            OAF.C1;
link     .C2            OAF.C2;
link     .C3            OAF.C3;

//Lower estimation of outside Air Fraction
declare OAFLow OAFLow;
link     OAFLow0        OAFLow.OAFLow                         eq2.b;;

//Higher estimation of outside Air Fraction
declare OAFHigh OAFHigh     ;
link     OAFHigh0       OAFHigh.OAFHigh                       eq3.b;

link     .pos           OAF.pos OAFLow.pos OAFHigh.pos;
link     .LeakRet       OAF.LeakRet OAFHigh.LeakRet OAFLow.LeakRet;
link     .LeakOut       OAF.LeakOut OAFHigh.LeakOut OAFLow.LeakOut;
```

```
/*+++
 Identification: mixing air temperature
 Abstract:
 Notes:
     None

 Interface:
    pos:            damper position(-) "0 to 1, 1 = 100% outside air, 0 = 100% return air "
    TRet:           Return air temperature                                   [oF]
    TOut:           Outside air temperature                          [oF]
    TMix:           mixing air temperature                                   [oF]
    TMixHigh: Lower estimation of mixing air temperature    [oF]
    TMixLow:  Higher estimation of mixing air temperature   [oF]
    LeakRet: The ratio of the  mass flow rate of return air to outside air
                    when damper equals to 1 (100% outside air)
    LeakOut: The ratio of the  mass flow rate of outside to return air
                    when damper equals to 0 (100% return air)
    RefPos:  reflection position (0-1), the position where the curves start to reflect
    n: the exponential constants (>0, real number)



 Acceptable input set:
     pos = 0, TRet = 20, TOut =30, LeakRet =0.01, LeatOut=0.01, n =2, RefPos=0.5

 Recommended matches:
       None

 Suggested breaks:
       None

 Local variables:
    OAF:            Outside air fraction (0-1)
    OAFHigh:                High estimation of outside air fraction (0-1)
    OAFLow:                 Low estimation of outside air fraction (0-1)
    OAFHalfHigh:  High estimation of outside air fraction when damper position equals to 0.5.
    OAFHalfLow:   Low estimation of outside air fraction when damper position equals to 0.5.
    OAFMax:       Maximum outside air fraction when damper postion equals to 1. (Leakage from return air damper)
    OAFMin:       Minimum outside air fraction when damper postion equals to 0. (Leakage from outside air damper)

 Equations:
         TMix = OAF * (TOut-TRet) + TRet;
     OAFMax = 1-LeakRet;
         OAFMin = LeakOut ;
         OAF = (OAFMax-OAFMin)* (C1*pos+C2*pos^2+C3*pos^3) + OAFMin

         OAFHalfHigh = OAFMin + 0.75*(OAFMax -OAFMin)
         OAFHigh = (OAFHalf-OAFMin) * (pos*2) + OAFMin                    ( pos <= 0.5)
         OAFHigh = (1 - OAFHalf) * ((pos -0.5)*2) + OAFHalfHigh     ( pos > 0.5)

         OAFHalfLow = OAFMin + 0.25*(OAFMax -OAFMin)
         OAFLow = OAFHalf * (pos*2)                          ( pos <= 0.5)
         OAFLow = (OAFMax - OAFHalfLow) * ((pos -0.5)*2) + OAFHalfLow     ( pos > 0.5)
*/
//PORT
PORT     pos     "damper position(-) (0 to 1, 1 = 100% outside air, 0 = 100% return air) " ;
PORT     TRet    "Return air temperature"                                  [oF];
PORT     TOut    "Outside air temperature"                                 [oF]   ;
PORT     TMix    "mixing air temperature"                                  [oF];
PORT       TMixHigh "Lower estimation of mixing air temperature"         [oF];
PORT     TMixLow  "Higher estimation of mixing air temperature"          [oF];
PORT     LeakRet  "The ratio of the  mass flow rate of return air to outside air when damper equals to 1 (100% outside
air)" [];
PORT     LeakOut  "The ratio of the  mass flow rate of outside to return air when damper equals to 0 (100% return air)" [];
```



```
PORT       RefPos   "reflection position (0-1), the position where the curves start to reflect, 0-1" ;
```

```
PORT        n             "the exponential constants (>0, real number)        " ;
```

~~**mix_EXP.cm**~~ ~~**Mix / SOURCE CODE**~~

```
declare equal_link eq1 eq2 eq3;

//Mixed Air temperature
//predicted mixed air temperature
declare tmix tmix;
link        OAF        tmix.OAF                                  eq1.a;
link        .TMix      tmix.TMix;

//Lower estimation of  mixed air temperature
declare tmix tmixLow;
link        OAFLow  tmixLow.OAF                                          eq2.a;
link        .TMixLow tmixLow.TMix;

//Higher estimation of  mixed air temperature
declare tmix tmixHigh;
link        OAFHigh tmixHigh.OAF                                         eq3.a;
link        .TMixHightmixHigh.TMix;

link        .TOut      tmix.TOut tmixLow.TOut          tmixHigh.TOut;
link        .TRet      tmix.TRet tmixLow.TRet          tmixHigh.TRet;

//Outside air fraction
//Predicted outside Air Fraction
declare OAF_EXP   OAF;
link        "OAF"      OAF.OAF                                           eq1.b;
link    .n    OAF.n;
link    .RefPos        OAF.RefPos;

//Lower estimation of outside Air Fraction
declare OAFLow OAFLow;
link        "OAFLow"             OAFLow.OAFLow                           eq2.b;;

//Higher estimation of outside Air Fraction
declare OAFHigh OAFHigh      ;
link        "OAFHigh"            OAFHigh.OAFHigh                         eq3.b;

link        .pos       OAF.pos OAFLow.pos OAFHigh.pos;
link        .LeakRet OAF.LeakRet OAFHigh.LeakRet OAFLow.LeakRet;
link        .LeakOut OAF.LeakOut OAFHigh.LeakOut OAFLow.LeakOut;
```

OAF.cc Mix / SOURCE CODE

```
/*+++
 Identification: estimation of outside air fraction
 Abstract:
 Notes:
      None
 Interface:
    pos:            damper position(-) "change from 0 to 1, 1 = 100% outside air, 2 = 100% return air "
    LeakRet:        The installed return air damper leakage (0-1)
    LeakOut:        The installed outside air damper leakage (0-1)
 Acceptable input set:
     pos = 0, LeakRet =0.01, LeatOut=0.01
 Recommended matches:
        None
 Suggested breaks:
        None
 Local variables:
    OAF:            Outside air fraction (0-1)
    OAFHalf:        Outside air fraction when damper position equals to 0.5.
    OAFMax:         Maximum outside air fraction when damper postion equals to 1. (Leakage from return air damper)
    OAFMin:         Minimum outside air fraction when damper postion equals to 0. (Leakage from outside air damper)
 Equations:
            OAFMax = 1-LeakRet;
            OAFMin = LeakOut ;
            OAF = (OAFMax-OAFMin)* (C1*pos+C2*pos^2+C3*pos^3) + OAFMin
*/
#ifdef SPARK_TEXT
//PORT
PORT    pos       "Damper position(-) (0 to 1, 1 = 100% outside air, 0 = 100% return air) "                [scalar];
PORT    OAF       "Outside air fraction"                                                                  [scalar];
PORT    LeakRet   "Installed return air damper leakage (0-1)"                                             [scalar];
PORT    LeakOut   "Installed outside air damper leakage (0-1)"                                            [scalar];
PORT      C1      "polynomial constant 1 for curve fitting the outside air fraction (C1+C2+C3=1)"         [scalar];
PORT      C2      "polynomial constant 2 for curve fitting the outside air fraction (C1+C2+C3=1)"         [scalar];
PORT      C3      "polynomial constant 3 for curve fitting the outside air fraction (C1+C2+C3=1)"         [scalar];

EQUATIONS {
            OAFMax = 1-LeakRet;
            OAFMin = LeakOut ;
            OAF = (OAFMax-OAFMin)* (C1*pos+C2*pos^2+C3*pos^3) + OAFMin;
            }
// ==== FUNCTIONS ====
FUNCTIONS {
                    OAF = OAF(pos, LeakRet, LeakOut,C1, C2, C3);
                    }
#endif /* SPARK_TEXT */
#include "spark.h"

 double
OAF ( ARGS )
{
            ARGDEF(0,pos) ;
            ARGDEF(1,LeakRet) ;
            ARGDEF(2,LeakOut) ;
            ARGDEF(3,C1) ;
            ARGDEF(4,C2) ;
            ARGDEF(5,C3) ;

            double OAFMax;
            double OAFMin;
            double OAF;
            OAFMax = 1-LeakRet;
```

OAF_EXP.cc Mix / SOURCE CODE

```
            OAFMin = LeakOut;
            OAF = (OAFMax-OAFMin)* (C1*pos+C2*pow(pos,2)+C3*pow(pos,3)) + OAFMin ;

            return OAF;
}
```

~~OAF_EXP.cc~~ ~~Mix / SOURCE CODE~~

```
/*+++
  Identification: estimation of outside air fraction based on two exponential functions
  Abstract:
  Notes:
      None

  Interface:
     pos:     damper position(-) "change from 0 to 1, 1 = 100% outside air, 2 = 100% return air "
     LeakRet: The return air damper leakage (0-1)
     LeakOut: The outside air damper leakage (0-1)
     RefPos:  reflection position (0-1), the position where the curves start to reflect
     n: the exponential constants (>0, real number)

  Acceptable input set:
      pos = 0, LeakRet =0.01, LeatOut=0.01, RefPos=0.5

  Recommended matches:
        None

  Suggested breaks:
        None

  Local variables:
     OAF:  Outside air fraction (0-1)
     OAFMax:   Maximum outside air fraction when damper postion equals to 1. (Leakage from return air damper)
     OAFMin:   Minimum outside air fraction when damper postion equals to 0. (Leakage from outside air damper)

  Equations:
       OAFMax = 1-LeakRet;
            OAFMin = LeakOut ;
            OAF = OAFMin + (OAFMax-OAFMin)*(pow(RefPos,n)+pow((pos-RefPos),n))/(pow(RefPos,n)+pow((1-
RefPos),n)) (if pos> RefPos)
            OAF = OAFMin + (OAFMax-OAFMin)*(pow(RefPos,n)-pow((RefPos-pos),n))/(pow(RefPos,n)+pow((1-
RefPos),n)) (if pos<= RefPos)
*/
#ifdef SPARK_TEXT
//PORT
PORT     pos     "Damper position(-) (0 to 1, 1 = 100% outside air, 0 = 100% return air) "     [scalar];
PORT     OAF          "Outside air fraction"                                              [scalar];
PORT     LeakRet "Return air damper leakage (0-1)"                                        [scalar];
PORT     LeakOut "Outside air damper leakage (0-1)"                                       [scalar];
PORT       n            "the exponential constants (>0, real number)        " ;
PORT       RefPos      "reflection position (0-1), the position where the curves start to reflect" ;

EQUATIONS {
            OAFMax = 1-LeakRet;
            OAFMin = LeakOut ;
            OAF = OAFMin + (OAFMax-OAFMin)*(pow(RefPos,n)+pow((pos-RefPos),n))/(pow(RefPos,n)+pow((1-
RefPos),n)) (if pos> RefPos)                            ;
            OAF = OAFMin + (OAFMax-OAFMin)*(pow(RefPos,n)-pow((RefPos-pos),n))/(pow(RefPos,n)+pow((1-
RefPos),n)) (if pos<= RefPos)                ;
            }

// ==== FUNCTIONS ====
FUNCTIONS {
                      OAF = OAF(pos, LeakRet, LeakOut, n, RefPos);
                      }
#endif /* SPARK_TEXT */
#include "spark.h"
```

OAF_EXP.cc Mix / SOURCE CODE

```
  double
OAF ( ARGS )
{
            ARGDEF(0,pos) ;
            ARGDEF(1,LeakRet) ;
            ARGDEF(2,LeakOut) ;
```

```
ARGDEF(3,n) ;
ARGDEF(4,RefPos) ;
```



```
double OAFMax;
double OAFMin;

double OAF;

OAFMax = 1-LeakRet;
OAFMin = LeakOut;

if (pos> RefPos)
OAF = OAFMin + (OAFMax-OAFMin)*(pow(RefPos,n)+pow((pos-RefPos),n))/(pow(RefPos,n)+pow((1-
RefPos),n)) ;
else
OAF = OAFMin + (OAFMax-OAFMin)*(pow(RefPos,n)-pow((RefPos-pos),n))/(pow(RefPos,n)+pow((1-
RefPos),n)) ;

return OAF;

}
```

OAFHigh.cc Mix / SOURCE CODE

```
/*+++
  Identification: upper estimation of outside air fraction
  Abstract:
  Notes:
       None
  Interface:
     pos:               damper position(-) "change from 0 to 1, 1 = 100% outside air, 2 = 100% return air "
     LeakRet:           The installed return air damper leakage (0-1)
     LeakOut:           The installed outside air damper leakage (0-1)
  Acceptable input set:
     pos = 0, LeakRet =0.01, LeatOut=0.01
  Local variables:
     OAFHigh:           Lower estimation of outside air fraction (0-1)
     OAFHalf:           Outside air fraction when damper position equals to 0.5.
     OAFMax:            Maximum outside air fraction when damper postion equals to 1. (Leakage from return air damper)
     OAFMin:             Minimum outside air fraction when damper postion equals to 0. (Leakage from outside air damper)

  Equations:
            OAFMax = 1-LeakRet;
            OAFMin = LeakOut ;
            OAFHalf = OAFMin + 0.75*(OAFMax -OAFMin)
            OAFHigh = (OAFHalf-OAFMin) * (pos*2) + OAFMin                              ( pos <= 0.5)
            OAFHigh = (1 - OAFHalf) * ((pos -0.5)*2) + OAFHalf            ( pos > 0.5)
*/
#ifdef SPARK_TEXT
//PORT
PORT      pos          "Damper position(-) (0 to 1, 1 = 100% outside air, 0 = 100% return air) "            [scalar];
PORT      OAFHigh  "Lower estimation of Outside air fraction"                                               [scalar];
PORT      LeakRet    "Installed return air damper leakage (0-1)"                                                           [scalar];
PORT      LeakOut    "Installed outside air damper leakage (0-1)"                                                          [scalar];

EQUATIONS {
      OAFMax = 1-LeakRet;
            OAFMin = LeakOut ;
            OAFHalf = OAFMin + 0.75*(OAFMax -OAFMin)
            OAFHigh = (OAFHalf-OAFMin) * (pos*2) + OAFMin            ( pos <= 0.5)
            OAFHigh = (1 - OAFHalf) * ((pos -0.5)*2) + OAFHalf            ( pos > 0.5)
             }
// ==== FUNCTIONS ====
FUNCTIONS {
                        OAFHigh = OAFHigh(pos, LeakRet, LeakOut);
                        }
#endif /* SPARK_TEXT */
#include "spark.h"
  double
OAFHigh ( ARGS )
{
            ARGDEF(0,pos) ;
            ARGDEF(1,LeakRet) ;
            ARGDEF(2,LeakOut) ;

            double OAFMax;
            double OAFMin;
            double OAFHalf;
            double OAFHigh;

            OAFMax = 1-LeakRet;
            OAFMin = LeakOut;
            OAFHalf = OAFMin + 0.75*(OAFMax -OAFMin);
            if ( pos <= 0.5 )
            OAFHigh = (OAFHalf-OAFMin) * (pos*2) + OAFMin ;
      else
            OAFHigh = (1 - OAFHalf) * ((pos -0.5)*2) + OAFHalf ;

            return OAFHigh;

}
```

OAFLow.cc Mix / SOURCE CODE

```
/*+++
  Identification: lower estimation of outside air fraction
  Abstract:
  Notes:
      None
  Interface:
     pos:            damper position(-) "change from 0 to 1, 1 = 100% outside air, 2 = 100% return air "
     LeakRet:        The installed return air damper leakage (0-1)
     LeakOut:        The installed outside air damper leakage (0-1)
  Acceptable input set:
      pos = 0, LeakRet =0.01, LeatOut=0.01
 Local variables:
     OAFLow:         Lower estimation of outside air fraction (0-1)
     OAFHalf:        Outside air fraction when damper position equals to 0.5.
     OAFMax:         Maximum outside air fraction when damper postion equals to 1. (Leakage from return air damper)
     OAFMin:         Minimum outside air fraction when damper postion equals to 0. (Leakage from outside air damper)

  Equations:
            OAFMax = 1-LeakRet;
            OAFMin = LeakOut ;
            OAFHalf = OAFMin + 0.25*(OAFMax -OAFMin)
            OAFLow = OAFHalf * (pos*2)                                   ( pos <= 0.5)
            OAFLow = (OAFMax - OAFHalf) * ((pos -0.5)*2) + OAFHalf       ( pos > 0.5)
*/
#ifdef SPARK_TEXT
//PORT
PORT     pos        "Damper position(-) (0 to 1, 1 = 100% outside air, 0 = 100% return air) "         [scalar];
PORT     OAFLow   "Lower estimation of Outside air fraction"                                          [scalar];
PORT     LeakRet  "Installed return air damper leakage (0-1)"
         [scalar];
PORT     LeakOut   "Installed outside air damper leakage (0-1)"
         [scalar];

EQUATIONS {
      OAFMax = 1-LeakRet;
            OAFMin = LeakOut ;
            OAFHalf = OAFMin + 0.25*(OAFMax -OAFMin)
            OAFLow = OAFHalf * (pos*2)                                            ( pos <= 0.5)
            OAFLow = (OAFMax - OAFHalf) * ((pos -0.5)*2) + OAFHalf   ( pos > 0.5)
             }
// ==== FUNCTIONS ====
FUNCTIONS {
                  OAFLow = OAFLow(pos, LeakRet, LeakOut);
                  }
#endif /* SPARK_TEXT */
#include "spark.h"
  double
OAFLow ( ARGS )
{
            ARGDEF(0,pos) ;
            ARGDEF(1,LeakRet) ;
            ARGDEF(2,LeakOut) ;

            double OAFMax;
            double OAFMin;
            double OAFHalf;
            double OAFLow;

            OAFMax = 1-LeakRet;
            OAFMin = LeakOut;
            OAFHalf = OAFMin + 0.25*(OAFMax -OAFMin);

            if ( pos <= 0.5 )
                  OAFLow = OAFHalf * (pos*2) ;
             else
                  OAFLow = (OAFMax - OAFHalf) * ((pos -0.5)*2) + OAFHalf;
```

tmix.cc Mix / SOURCE CODE

```
        return OAFLow;
}
```

~~tmix.cc~~ ~~Mix / SOURCE CODE~~

```
/*+++
/* CLASS  tmix       "determine the mixed air temperature based on outside air fraction"

ABSTRACT

ABSTRACT_END
TEST_INPUT
      TRet = 1, TOut = 0, TMix = 0.5 ;
*/
#ifdef SPARK_TEXT
// ==== PORTS ====

PORT     OAF    "outside air fraction in the mixed air"       [scalar] ;
PORT     TRet   "return air temperature"                      [deg_C] ;
PORT     TOut   "outside air temperature"                     [deg_C] ;
PORT     TMix   "mixed air temperature"                       [deg_C] ;

EQUATIONS {
     TMix = OAF * (TOut - TRet) + TRet;
  }

// ==== FUNCTIONS ====
FUNCTIONS {
        OAF      = tmix_OAF( TRet, TOut, TMix ) ;
        TMix     = tmix_TMix (OAF, TRet, TOut );
          }
#endif /* SPARK_TEXT */
#include "spark.h"

  double
tmix_OAF ( ARGS )
{
   ARGDEF(0,TRet) ;
   ARGDEF(1,TOut) ;
   ARGDEF(2,TMix) ;

   double OAF;
   OAF = ( TMix - TRet ) / (TOut - TRet) ;
   return OAF;
}

  double
tmix_TMix ( ARGS )
{
   ARGDEF(0,OAF) ;
   ARGDEF(1,TRet) ;
   ARGDEF(2,TOut) ;

   double TMix;
   TMix = OAF * (TOut - TRet) + TRet;
   return TMix;
}
```

Air handling unit (AHU)

General description

An air-handling unit consists of a set of components that together provide conditioned air for distribution to occupied spaces. The components described above perform the functions of heating, cooling, dehumidification and ventilation. The fan system model implicitly treats the pressure drops due to the coils, filters and attenuators. These models can be connected together in different combinations to form models of different types of air handling units. Figure 7 shows an example, which consists of a mixing box, a cooling coil with a control valve, a heating coil with a control valve, and supply and return fans.

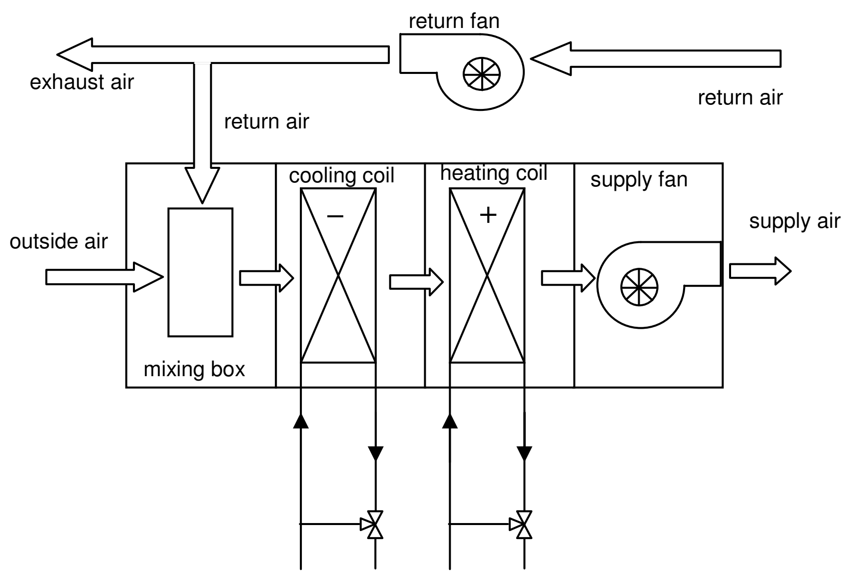

Figure 7 Schmetic of the modeled air handling unit (AHU)

Model description

The model of an air handling unit is built by linking all the above related component models together. The outlet air property of a particular component is the inlet property of the component immediately downstream. Class ***AHU_Example.cm*** is an example model of an AHU that has the configuration shown in Figure 7.

Governing equations:

$$m_{air,\sup plyfan} = m_{air,coolingcoil} = m_{air,heatingcoil}$$

$$m_{water,valveCC} = m_{water,coolingcoil}$$

$$m_{water,valveHC} = m_{water,heatingcoil}$$

$$T_{air,AHU,outside} = T_{air,mixing,outside}$$

$$T_{air,mixing,lvg} = T_{air,ccolingcoil,ent}$$

$$T_{air,coolingcoil,lvg} = T_{air,heatingcoil,ent}$$

$$T_{air,heatingcoil,lvg} = T_{air,\sup pplyfan,ent}$$
$$T_{air,\sup plyfan,lvg} = T_{air,AHU,\sup}$$
$$T_{air,AHU,ret} = T_{air,returnfan,ent}$$
$$T_{air,return,lvg} = T_{air,mixing,ret}$$

$$w_{air,AHU,outside} = w_{air,mixing,outside}$$
$$w_{air,mixing,lvg} = w_{air,ccolingcoil,ent}$$
$$w_{air,coolingcoil,lvg} = w_{air,AHU,\sup}$$

Nomenclature

Variables		Description	Unit
$m_{air,supplyfan}$	mAirSup	Supply fan air flow rate	[kg_dryAir/s]
$m_{water,coolingcoil}$	mLiqCC	Cooling coil Liquid flow rate	[kg/s]
$m_{water,valveCC}$	mLiqValCC	Cooling coil Liquid flow rate	[kg/s]
$m_{water,heatingcoil}$	mLiqHC	Heating coil Liquid flow rate	[kg/s]
$m_{water,valveHC}$	mLiqValHC	Heating coil Liquid flow rate	[kg/s]
$T_{air,AHU,outside}$	TAirOut	Outside air temperature	[deg_C]
$T_{air,AHU,ret}$	TAirRet	Return air dry bulb temperature	[deg_C]
$T_{air,AHU,sup}$	TAirSup	Supply air dry bulb temperature	[deg_C]
$T_{air,coolingcoil,lvg}$	TAirLvgCC	Cooling coil leaving air dry bulb temperature	[deg_C]
$T_{air,coolingcoil,ent}$	TAirEntCC	Cooling coil entering air dry bulb temperature	[deg_C]
$T_{air,heatingcoil,ent}$	TAirEntHC	Heating coil entering air dry bulb temperature	[deg_C]
$T_{air,heatingcoil,lvg}$	TAirLvgHC	Heating coil leaving air dry bulb temperature	[deg_C]
$T_{air,supplyfan,lvg}$	TAirLvgSfan	Supply fan leaving air dry bulb temperature	[deg_C]
$T_{air,supplyfan,ent}$	TAirEntSfan	Supply fan entering air dry bulb temperature	[deg_C]
$T_{air,returnfan,ent}$	TAirEntRfan	Return fan entering air dry bulb temperature	[deg_C]
$T_{air,returnfan,lvg}$	TAirLvgRfan	Return fan leaving air dry bulb temperature	[deg_C]
$T_{air,mixing,outside}$	TAirMixOut	Mixing box outside air temperature	[deg_C]
$T_{air,mixng,ret}$	TAirMixRet	Mixing box return air dry bulb temperature	[deg_C]
$T_{air,mixing,lvg}$	TAirMixLvg	Mixing box leaving air dry bulb temperature	[deg_C]

```
/*+++

/*+++
  Identification: AHU model for diagnosis.

  Abstract:


  Notes:
     None

  Interface:

        TAirLvgCC          Cooling coil leaving air dry bulb temperature                    [deg_C]
        wAirLvgCC          Cooling coil leaving air humidity  ratio                         [kg/kg_dryAir]
        TLiqEntCC          Cooling coil entering water temperature                          [deg_C]
        TLiqLvgCC          Cooling coil leaving water temperature                           [deg_C]
        AExtCC             Cooling coil heat transfer area                                  [m2]
        AIntCC             Cooling coil heat transfer area                                  [m2]
        CExtCC             Cooling coil air side heat transfer coefficient constant         [scalar]
        CIntCC             Cooling coil liquid side heat transfer coefficient constant      [scalar]
        PAtm               Atmospheric pressure
        [Pa]
        qSenCC             Cooling coil Sensible heat transfer rate.  Positive for air cooling.  [W]
        qLatCC             Cooling coil Latent heat transfer rate.  Positive for air cooling.    [W]
        qTotCC             Cooling coil Heat transfer rate.  Positive for air cooling.           [W]

        posValveCC         Cooling coil Valve position, between 0-1                         [scalar]
        AValveCC           Cooling coil Valve authority, between 0-1                        [scalar]
        mLiqOpenValveCC    Cooling coil Mass flow rate for open valve                       [Kg/s]
        mLeakValveCC       Cooling coil Mass flow rate of leakage                           [Kg/s]
        mLiqCC             Cooling coil Liquid flow rate                                    [kg/s]

        TAirLvgHC          heating coil leaving air dry bulb temperature                    [deg_C]
        TLiqEntHC          heating coil entering water temperature                          [deg_C]
        TLiqLvgHC          heating coil leaving water temperature                           [deg_C]
        AHC                heating coil heat transfer area                                  [m2]
        CHC                heating coil liquid side heat transfer coefficient constant      [scalar]
        qSenHC             heating coil Sensible heat transfer rate.  Positive for air cooling.  [W]

        posValveHC         heating coil Valve position, between 0-1                         [scalar]
        AValveHC           heating coil Valve authority, between 0-1                        [scalar]
        mLiqOpenValveHC    heating coil Mass flow rate for open valve                       [Kg/s]
        mLeakValveHC       heating coil Mass flow rate of leakage                           [Kg/s]
        mLiqHC             heating coil Liquid flow rate                                    [kg/s]

        TAirSup            Supply air dry bulb temperature                                  [deg_C]
        wAirSup            Supply air humidity  ratio                                       [kg/kg_dryAir]
         mAirSup           supply fan air flow rate                                         [kg_dryAir/s]
        powerTotSfan       supply fan motor power consumption                               [W]
        nSfan              supply fan  fan speed                                            [rpm]
        pStatSfan          supply fan static pressure setpoint                              [Pa]
         pSfan             supply fan  total pressure increase across fan                   [Pa]
        effMotSfan         supply fan  Efficiency of fan motor                              [scalar]
        motFracSfan        supply fan  Fraction of motor heat loss in air stream            [fraction]
        effShaftSfan       supply fan  fan efficiency                                       [scalar]
        effShaftMaxSfan    supply fan  fan maximum efficiency                               [scalar]
```

AHU_Example.cm AHU / SOURCE CODE

```
        mAirMaxSfan        supply fan  maximum air flow of the fan                          [kg_dryAir/s]
        CResSfan           supply fan  resistance charactristic constant                    [scalar]
        CSfan              supply fan  fan curve constant                                   [scalar]
        kSfan              supply fan  pressure-fanspeed constant                           [scalar]
        CEffSfan           supply fan  fan effiency constant                                [scalar]
        areaSPSfan         supply fan  duct work crossing section area                      [m2]
```

TAirRet	Return air dry bulb temperature	[deg_C]
wAirRet	Return air humidity ratio	[kg_water/kg_dryAir]
mAirRet	Return fan air flow rate	[kg_dryAir/s]
powerTotRfan	return fan motor power consumption	[W]
nRfan	return fan fan speed	[rpm]
pStatRfan	return fan static pressure setpoint	[Pa]
pRfan	return fan total pressure increase across fan	[Pa]
effMotRfan	return fan Efficiency of fan motor	[scalar]
motFracRfan	return fan Fraction of motor heat loss in air stream	scalar]
effShaftRfan	return fan fan efficiency	[scalar]
effShaftMaxRfan	return fan fan maximum efficiency	[scalar]
mAirMaxRfan	return fan maximum air flow of the fan	[kg_dryAir/s]
CResRfan	return fan resistance charactristic constant	[scalar]
CRfan	return fan fan curve constant	[scalar]
kRfan	return fan pressure-fanspeed constant	[scalar]
CEffRfan	return fan fan effiency constant	[scalar]
areaSPRfan	return fan duct work crossing section area	[m2]
TAirOut	Outside air temperature	[deg_C]
wAirOut	Outside humidity ratio	[kg/kg]
posDamper	damper position (0 to 1, 1 = 100% outside air, 0 = 100% return air)	[scalar]
LeakRetDamper	installed return damper leakage (0-1)	[scalar]
LeakOutDamper	installed outside air damper leakage (0-1)	[scalar]
mixC1	polynomial constant 1 for curve fitting the outside fraction	[scalar]
mixC2	polynomial constant 2 for curve fitting the outside fraction	[scalar]
mixC3	polynomial constant 3 for curve fitting the outside fraction	[scalar]

Acceptable input set:

TAirLvgCC	= unknown	[deg_C]
wAirLvgCC	= unknown	[kg_water/kg_dryAir]
TLiqEntCC	= 7	[deg_C]
TLiqLvgCC	= unknown	[deg_C]
AExtCC	= 1	[m2]
AIntCC	= 1	[m2]
CExtCC	= 1000	[scalar]
CIntCC	= 4000	[scalar]
PAtm	= 100000	[Pa]
qSenCC	= unknown	[W]
qLatCC	= unknown	[W]
qTotCC	= unknown	[W]

AHU_Example.cm AHU / SOURCE CODE

posValveCC	= 0.5	[scalar]
AValveCC	= 0.5	[scalar]
mLiqOpenValveCC	= 3	[Kg/s]
mLeakValveCC	= 0.1	[Kg/s]
mLiqCC	= unknown	[kg/s]
TAirLvgHC	= unknown	[deg_C]
TLiqEntHC	= 95	[deg_C]
TLiqLvgHC	= unknown	[deg_C]
AHC	= 1	[m2]
CHC	= 4000	[scalar]
qSenHC	= unknown	[W]
posValveHC	= 0.5	[scalar]
AValveHC	= 0.5	[scalar]
mLiqOpenValveHC	= 3	[Kg/s]
mLeakValveHC	= 0.1	[Kg/s]
mLiqHC	= unknown	[kg/s]
TAirSup	= unknown	[deg_C]
wAirSup	= unknown	[kg_water/kg_dryAir]

~~AHU_Example.cm AHU / SOURCE CODE~~

```
mAirSup          = unknown    [kg_dryAir/s]
powerTotSfan     = unknown    [W]
 nSfan           = unknown    [rpm]
 pStatSfan       = 20         [Pa]
pSfan            = unknown    [Pa]
effMotSfan       = 0.9        [scalar]
motFracSfan      = 1          [fraction]
effShaftSfan     = unknown    [scalar]
effShaftMaxSfan  = 0.9        [scalar]
mAirMaxSfan      = 5          [kg_dryAir/s]
CResSfan         = 0.1        [scalar]
CSfan            = 0.3        [scalar]
kSfan            = 0.00125    [scalar]
CEffSfan         = 0.0001     [scalar]
areaSPSfan       = 0.3        [m2]

TAirRet          = 25         [deg_C]
wAirRet          = 0.007      [kg_water/kg_dryAir]
 mAirRet         = unknown    [kg_dryAir/s]
powerTotRfan     = unknown    [W]
nRfan            = unknown    [rpm]
pStatRfan        = 20         [Pa]
 pRfan           = unknown    [Pa]
effMotRfan       = 0.9        [scalar]
motFracRfan      = 1          [fraction]
effShaftRfan     = unknown    [scalar]
effShaftMaxRfan = 0.9         [scalar]
mAirMaxRfan      = unknown    [kg_dryAir/s]
CResRfan         = 0.1        [scalar]
CRfan            = 0.3        [scalar]
kRfan            = 0.00125    [scalar]
CEffRfan         = 0.0001     [scalar]
areaSPRfan       = 0.3        [m2]

TAirOut          = 38         [deg_C]
wAirOut          = 0.009      [kg/kg]
posDamper        = 0.5        [scalar]
LeakRetDamper    = 0.01       [scalar]
LeakOutDamper    = 0.01       [scalar]
mixC1            = 0.8        [scalar]
mixC2            = 0.1        [scalar]
mixC3            = 0.1        [scalar]
```

AHU_Example.cm AHU / SOURCE CODE

```
 Recommended matches:
        None

 Suggested breaks:
        None

 Local variables:
                None
 Equations:
                    Objects: cooling coil, fan, valve, mixing box, air specific volume;
---*/

//cooling coil
PORT    TAirLvgCC    "Cooling coil leaving air dry bulb temperature"              [deg_C] ;
PORT    wAirLvgCC    "Cooling coil leaving air humidity  ratio"                   [kg_water/kg_dryAir];
PORT    TLiqEntCC    "Cooling coil entering water temperature"                    [deg_C] ;
PORT    TLiqLvgCC    "Cooling coil leaving water temperature"                     [deg_C] ;
PORT    AExtCC       "Cooling coil heat transfer area"                            [m2] ;
PORT    AIntCC       "Cooling coil heat transfer area"                            [m2] ;
PORT    CExtCC       "Cooling coil air side heat transfer coefficient constant"   [scalar] ;
PORT    CIntCC       "Cooling coil liquid side heat transfer coefficient constant" [scalar] ;
```

```
PORT    PAtm            "Atmospheric pressure"                                    [Pa];
PORT    qSenCC          "Cooling coil Sensible heat transfer rate.  Positive for air cooling."    [W];
```



```
PORT    qLatCC          "Cooling coil Latent heat transfer rate.  Positive for air cooling."     [W];
PORT    qTotCC          "Cooling coil Heat transfer rate.  Positive for air cooling."            [W];
//valve-cooling
port    posValveCC      "Cooling coil Valve position, between 0-1"                  [scalar] ;
port    AValveCC        "Cooling coil Valve authority, between 0-1"                 [scalar] ;
port    mLiqOpenValveCC "Cooling coil Mass flow rate for open valve"                [Kg/s] ;
port    mLeakValveCC    "Cooling coil Mass flow rate of leakage"                    [Kg/s] ;
PORT    mLiqCC          "Cooling coil Liquid flow rate "                            [kg/s] ;
//heating coil
PORT    TAirLvgHC       "heating coil leaving air dry bulb temperature"             [deg_C] ;
PORT    TLiqEntHC       "heating coil entering water temperature"                   [deg_C] ;
PORT    TLiqLvgHC       "heating coil leaving water temperature"                    [deg_C] ;
PORT    AHC             "heating coil heat transfer area"                           [m2] ;
PORT    CHC             "heating coil liquid side heat transfer coefficient constant"   [scalar] ;
PORT    qSenHC          "heating coil Sensible heat transfer rate.  Positive for air cooling."   [W];
//valve-heating
port    posValveHC      "heating coil Valve position, between 0-1"                  [scalar] ;
port    AValveHC        "heating coil Valve authority, between 0-1"                 [scalar] ;
port    mLiqOpenValveHC "heating coil Mass flow rate for open valve"                [Kg/s] ;
port    mLeakValveHC    "heating coil Mass flow rate of leakage"                    [Kg/s] ;
PORT    mLiqHC          "heating coil Liquid flow rate "                            [kg/s] ;
//fan-supply fan
PORT    TAirSup         "Supply air dry bulb temperature"                           [deg_C] ;
PORT    wAirSup         "Supply air humidity  ratio"                                [kg/kg_dryAir];
PORT    mAirSup         "supply fan air flow rate "                                 [kg_dryAir/s];
PORT    powerTotSfan    "supply fan motor power consumption"                        [W];
PORT    nSfan           "supply fan  fan speed "                                    [rpm];
PORT    pStatSfan       "supply fan static pressure setpoint "                      [Pa] ;
PORT    pSfan           "supply fan  total pressure increase across fan"            [Pa];
PORT    effMotSfan      "supply fan  Efficiency of fan motor"                       [scalar] ;
PORT    motFracSfan     "supply fan  Fraction of motor heat loss in air stream"     [fraction] ;
PORT    effShaftSfan    "supply fan  fan efficiency"                                [scalar];
PORT    effShaftMaxSfan "supply fan  fan maximum efficiency"                        [scalar];
PORT    mAirMaxSfan     "supply fan  maximum air flow of the fan"                   [kg_dryAir/s];
PORT    CResSfan        "supply fan  resistance charactristic constant"             [scalar];
PORT    CSfan           "supply fan  fan curve constant"                            [scalar];
PORT    kSfan           "supply fan  pressure-fanspeed constant"                    [scalar];
```

AHU_Example.cm AHU / SOURCE CODE

```
PORT    CEffSfan        "supply fan  fan effiency constant"                         [scalar];
PORT    areaSPSfan      "supply fan  duct work crossing section area"               [m2];

//fan-return fan
PORT    TAirRet         "Return air dry bulb temperature"                           [deg_C] ;
PORT    wAirRet         "Return air humidity  ratio"                                [kg/kg_dryAir];
PORT    mAirRet         "Return fan air flow rate "                                 [kg_dryAir/s];
PORT    powerTotRfan    "return fan motor power consumption"                        [W];
PORT    nRfan           "return fan  fan speed "                                    [rpm];
PORT    pStatRfan       "return fan static pressure setpoint "                      [Pa] ;
PORT    pRfan           "return fan  total pressure increase across fan"            [Pa];
PORT    effMotRfan      "return fan  Efficiency of fan motor"                       [scalar] ;
PORT    motFracRfan     "return fan  Fraction of motor heat loss in air stream"     [fraction] ;
PORT    effShaftRfan    "return fan  fan efficiency"                                [scalar];
PORT    effShaftMaxRfan "return fan  fan maximum efficiency"                        [scalar];
PORT    mAirMaxRfan     "return fan  maximum air flow of the fan"                   [kg_dryAir/s];
PORT    CResRfan        "return fan  resistance charactristic constant"             [scalar];
PORT    CRfan           "return fan  fan curve constant"                            [scalar];
PORT     kRfan          "return fan  pressure-fanspeed constant"                    [scalar];
PORT    CEffRfan        "return fan  fan effiency constant"                         [scalar];
PORT    areaSPRfan      "return fan  duct work crossing section area"               [m2];

//mixing box
PORT    TAirOut         "Outside air temperature"                                   [deg_C]   ;
PORT    wAirOut         "Outside humidity ratio"                                    [kg/kg]   ;
PORT    posDamper       "damper position(-) (0 to 1, 1 = 100% outside air, 0 = 100% return air) " [scalar];
```

```
PORT    LeakRetDamper        "installed return damper leakage (0-1)"                              [scalar];
```



```
PORT    LeakOutDamper        "outside air damper leakge (0-1)"                                   [scalar];
PORT      mixC1              "polynomial constant 1 for curve fitting the outside air fraction"  [scalar] ;
PORT      mixC2              "polynomial constant 2 for curve fitting the outside air fraction"  [scalar];
PORT      mixC3              "polynomial constant 3 for curve fitting the outside air fraction"  [scalar];


declare equal_link eq1 eq2 eq4 eq5 eq6 eq7 eq8 eq9 eq10 eq11 eq12 eq14 eq15 eq16;
//LINKS
declare coil_cooling_counter_flow cc;
link                        TAirEntCC       cc.TAirEnt              eq1.a;
link                        wAirEntCC       cc.wAirEnt              eq2.a;
link                        mAirCC          cc.mAir                 eq7.a     eq9.a;
link                        mLiqCC          cc.mLiq                 eq4.b ;
link      .TAirLvgCC                        cc.TAirLvg              eq5.a;
link      .wAirLvgCC                        cc.wAirLvg              eq8.a;
link      .TLiqEntCC                        cc.TLiqEnt;
link      .TLiqLvgCC                        cc.TLiqLvg;
link      .AExtCC                           cc.AExt   ;
link      .AIntCC                           cc.AInt;
link      .CExtCC                           cc.CExt;
link      .CIntCC                           cc.CInt;
link      .PAtm                             cc.PAtm                 eq12.b;
link      .qSenCC                           cc.qSen;
link      .qLatCC                           cc.qLat;
link      .qTotCC                           cc.qTot;

declare valve CCvalve;
```



```
link      .posValveCC                       CCvalve.pos;
link      .mLiqCC                           CCvalve.mLiq            eq4.a;
link      .AValveCC                         CCvalve.A;
link      .mLiqOpenValveCC                  CCvalve.mLiqOpen;
link      .mLeakValveCC                     CCvalve.mLeak       ;

declare coil_heating_cross_flow hc;
link                        mAirHC          hc.mAirEnt              eq7.b;
link                        mLiqHC          hc.mLiq                 eq6.a;
link                        TAirEntHC       hc.TAirEnt              eq5.b;
link                        wAirEntHC       hc.wAirEnt              eq8.b;
link                        mAirLvgHC       hc.mAirLvg;
link                        wAirLvgHC       hc.wAirLvg              eq11.a;
link      .TLiqEntHC                        hc.TLiqEnt;
link      .TAirLvgHC                        hc.TAirLvg              eq10.a;
link      .TLiqLvgHC                        hc.TLiqLvg              ;
link      .qSenHC                           hc.qSen   ;
link      .CHC                              hc.CHx    ;
link      .AHC                              hc.AHx      ;

declare valve HCvalve;
link      .posValveHC                       HCvalve.pos;
link      .mLiqHC                           HCvalve.mLiq            eq6.b;
link      .AValveHC                         HCvalve.A;
link      .mLiqOpenValveHC                  HCvalve.mLiqOpen;
link      .mLeakValveHC                     HCvalve.mLeak;

declare fan_system Sfan;
link                        TAirEntSfan     Sfan.TAirEnt            eq10.b;
link                        wAirEntSfan     Sfan.wAirEnt            eq11.b;
link      .TAirSup                          Sfan.TAirLvg;
link      .wAirSup                          Sfan.wAirLvg;
link      .powerTotSfan                     Sfan.powerTot;
link      .effMotSfan                       Sfan.effMot;
```

```
link        .motFracSfan                    Sfan.motFrac;
link        .effShaftSfan                   Sfan.effShaft;
link        .effShaftMaxSfan                Sfan.effShaftMax;
link        .mAirMaxSfan                    Sfan.mAirMax;
link        .nSfan                          Sfan.nFan;
```



```
link        .pStatSfan                      Sfan.pStat;
link        .pSfan                          Sfan.pFan;
link        .mAirSup                        Sfan.mAir                     eq9.b ;
link        .CResSfan                       Sfan.CRes  ;
link        .CSfan                          Sfan.CFan ;
link        .kSfan                          Sfan.kFan ;
link        .CEffSfan                       Sfan.CEff;
link        .areaSPSfan                     Sfan.area;
link                      PAtmSfan          Sfan.PAtm                     eq12.a eq16.b;

declare fan_system Rfan;
link        .TAirRet                        Rfan.TAirEnt                  ;
link        .wAirRet                        Rfan.wAirEnt                  ;
link                      TAirLvgRfan       Rfan.TAirLvg                  eq14.a;
link                      wAirLvgRfan       Rfan.wAirLvg                  eq15.a;
link        .powerTotRfan                   Rfan.powerTot;
link        .effMotRfan                     Rfan.effMot;
link        .motFracRfan                    Rfan.motFrac;
link        .effShaftRfan                   Rfan.effShaft;
link        .effShaftMaxRfan                Rfan.effShaftMax;
link        .mAirMaxRfan                    Rfan.mAirMax;
link        .nRfan                          Rfan.nFan;
link        .pStatRfan                      Rfan.pStat;
link        .pRfan                          Rfan.pFan;
link        .mAirRet                        Rfan.mAir             ;
link        .CResRfan                       Rfan.CRes  ;
link        .CRfan                          Rfan.CFan ;
link        .kRfan                          Rfan.kFan ;
link        .CEffRfan                       Rfan.CEff;
link        .areaSPRfan                     Rfan.area;
link                      PAtmRfan          Rfan.PAtm                     eq16.a;

declare mix mixT mixW;
link        .posDamper                      mixT.pos        mixW.pos    ;
link        .LeakRetDamper                  mixT.LeakRet    mixW.LeakRet ;
link        .LeakOutDamper                  mixT.LeakOut    mixW.LeakOut ;
link         .mixC1                         mixT.C1   mixW.C1;
link         .mixC2                         mixT.C2   mixW.C2;
link         .mixC3                         mixT.C3   mixW.C3;

link                      TAirRetMix        mixT.TRet                     eq14.b;
link        .TAirOut                        mixT.TOut   ;
link                      TMixLvg           mixT.TMix                     eq1.b  ;
link                      TMixLow           mixT.TMixLow;
link                      TMixHigh mixT.TMixHigh;
link                      wAirRetMix        mixW.TRet                     eq15.b;
link        .wAirOut                        mixW.TOut   ;
link                      wMixLvg           mixW.TMix                     eq2.b  ;
link                      wMixLow           mixW.TMixLow;
link                      wMixHigh          mixW.TMixHigh;
```

Chiller REFERENCE MODELS

General description

Chiller models can generally be divided into two categories: efficiency models and detailed mechanistic models. Efficiency models predict the power required to meet a particular load at particular operating conditions. These models can usually be extended to model capacity, i.e. the ability to meet that particular load. The input variables for these models are the water temperatures and flow rates. Mechanistic models predict refrigerant temperatures, pressures and flow rates and account explicitly for faults such as fouling and incorrect refrigerant charge.

Model description

Previously developed efficiency models were compared, specifically the DOE-2/CoolTools empirical model, the Gordon and Ng thermodynamic model and the ASHRAE Primary Toolkit model, which is a simplified mechanistic model (Sreedharan and Haves 2001). The Gordon and NG universal chiller model (2nd generation) was selected for use in the library. The model is based on both energy and entropy balances, thus incorporating both the first and second laws of thermodynamics. As in the ASHRAE Toolkit model, sensible heat exchange is not treated explicitly in either the condenser or the evaporator, which are modeled using the NTU-ε method assuming an infinite capacity rate on the refrigerant side. The performance equation is expressed in a form that is linear in physically meaningful parameters. The values of the model parameters, ΔS_T , $Q_{leak,eqv}$, R , are obtained by linear regression.

Governing equations:

$$q_{eva} = m_{liq,eva} c_{liq} (T_{eva,ent} - T_{eva,lvg})$$

$$q_{con} = m_{liq,con} c_{liq} (T_{con,lvg} - T_{con,ent})$$

$$\frac{T_{eva,ent}}{T_{con,ent}}\left(1+\frac{1}{COP}\right)-1 = \frac{T_{eva,ent}}{q_{eva}}\Delta S_T + Q_{leak,eqv}\frac{T_{con,ent}-T_{eva,ent}}{T_{con,ent}\times q_{eva}} + \frac{R\times q_{eva}}{T_{con,ent}}\left(1+\frac{1}{COP}\right)$$

$$W_{com} = \frac{q_{eva}}{COP}$$

$$q_{con} = W_{com} + q_{eva}$$

Nomenclature

Variables		Description	Unit
COP	COP	chiller COP	Dimensionless
c_{liq}	cLiq	water specific heat	kW/kg.K
$m_{liq,con}$	mLiqCon	Water mass flow rate at condenser	kg/s
$m_{liq,eva}$	mLiqEva	Water mass flow rate at evaporator	kg/s
q_{eva}	qEva	Heat exchange at evaporator	kW
q_{con}	qCon	Heat exchange at condenser	kW
Q_{leak}	QLeak	equvilant heat leak	kW
R	R	total heat exchanger thermal resistance =(1/C_con) + (1/C_eva)"	K/kW
St	St	total internal entropy production	K/kW
$T_{eva,ent}$	TEvaEnt	Entering water temperature at evaporator	K
$T_{eva,lvg}$	TEvaLvg	Leaving water temperature at evaporator	K
$T_{con,ent}$	TConEnt	Entering water temperature at condenser	K
$T_{con,lvg}$	TConLvg	Leaving water temperature at condenser	K
W_{com}	WCom	Compressor power consumption	kW

Chiller.cm Chiller / SOURCE CODE

```
/*+++
 Identification: Chiller model using Ng-Gordon method.

 Abstract:


 Notes:
     None

 Interface:
    mLiqEva:        Water mass flow rate at evaprator                   [Kg/s]
    mLiqCon:        Water mass flow rate at condenser                   [Kg/s]
    TEvaEnt:        Entering water temperature at evaprator             [K]
    TEvaLvg:        Leaving water temperature at evaprator              [K]
    TConEnt:        Entering water temperature at condenser             [K]
    TEvaLvg:        Leaving water temperature at condenser              [K]
    qEva:           Heat exchange at evaporator                         [kW]
    qCon:           Heat exchange at condenser                          [kW]
    WCom:           Compressor power consumption                        [kW]
    St:             total internal entropy production                   [K/kW]
    R:              total heat exchanger thermal resistance
                    =(1/C_con) + (1/C_eva)                              [K/kW]
    QLeak:          equvilant heat leak                                 [kW]
    cLiq:           water specific heat                                 [kW/kg.K]

 Acceptable input set:
    mLiqEva:        0.5
    mLiqCon:        0.5
    TEvaEnt:        293
    TEvaLvg:        283
    TConEnt:        310
    St:             0.005
    R:              2.5
    QLeak:          0.2
    cLiq:            4.182

 Recommended matches:
       None

 Suggested breaks:
       None

 Local variables:
          COP:  coefficient of performance         [scalar]
          cLiq:   water specific heat             [kW/kg.K]

 Equations:
          qEva=mLiqEva*cLiq*(TEvaEnt-TEvaLvg);
          qCon=mLiqCon*cLiq*(TConLvg-TConEnt);
          COP=(1-TEvaEnt/TConEnt+TEvaEnt*St/qEva+QLeak*(TConEnt-
                TEvaEnt)/TConEnt/qEva+qEva/TConEnt/UA)^-1*(TEvaEnt-qEva/UAEva)/TConEnt;
          WCom=1/COP*qEva;
          qCon =Wcom + qEva;

---*/
PORT     mLiqEva  "Water mass flow rate at evaprator"          [Kg/s];
PORT     mLiqCon  "Water mass flow rate at condenser "          [Kg/s];
PORT     TEvaEnt  "Entering water temperature at evaprator"     [K];
PORT     TEvaLvg  "Leaving water temperature at evaprator"      [K];
PORT     TConEnt  "Entering water temperature at condenser"     [K];
PORT     TConLvg  "Leaving water temperature at condenser"      [K];
PORT     qEva     "Heat exchange at evaporator"                 [kW];
PORT     qCon      "Heat exchange at condenser"                 [kW];
PORT     WCom     "Compressor power consumption"                [kW];
PORT     St       "total internal entropy production"           [K/kW];
PORT     R        "total heat exchanger thermal resistance
                  =(1/C_con) + (1/C_eva)"                       [K/kW];
PORT     QLeak    "equvilant heat leak "                        [kW];
```

```
PORT      cLiq   "water specific heat      "                          [kW/kg.K];
PORT      COP   "chiller COP"                                         [scalar];


declare equal_link eq1 eq2 eq3 eq4 eq5 eq6 eq7 eq8;
//Evaporator
//qEva=mLiqEva*cLiq*(TEvaEnt-TEvaLvg);
DECLARE cond Eva;
DECLARE safprod pd1;
LINK      .TEvaEnt          Eva.T1                    eq1.a;
LINK      .TEvaLvg          Eva.T2    ;
LINK      .cLiq             pd1.a                     eq2.a;
LINK      .mLiq             Eva pd1.b;
LINK                        pd1.c Eva.U12;
LINK                        qEva Eva.q                eq5.b;

//Condenser
//qCon=mLiqCon*cLiq*(TConLvg-TConEnt);
DECLARE cond Con;
DECLARE safprod pd2;
LINK      .TConEnt          Con.T2                    eq4.a;
LINK      .TConLvg          Con.T1;
LINK                        cLiq pd2.a                eq2.b;
LINK      .mLiqCon          pd2.b;
LINK                        pd2.c Con.U12;
LINK      .qCon             Con.q                     eq6.a;

//COP=(1-TEvaEnt/TConEnt+TEvaEnt*St/qEva+QLeak*(TConEnt-TEvaEnt)/TConEnt/qEva+qEva/TConEnt/UAEva)^-
1*(TEvaEnt-qEva/UAEva)/TConEnt;
declare chiller_COP COP;
LINK                        TEvaEnt COP.TEvaEnt       eq1.b;
LINK                        TConEnt COP.TConEnt       eq4.b;
LINK      .St               COP.St;
LINK      .qEva             COP.qEva                  eq3.a eq5.a;
LINK      .QLeak            COP.QLeak;
LINK      .R                COP.R ;

//        WCom=qEva/COP;
declare safquot sq;
LINK                        qEva1 sq.a                eq3.b eq7.a;
LINK      .COP              COP.COP sq.b;
LINK      .Wcom              sq.c                     eq8.a;

//        qCon =Wcom + qEva;
declare sum sum1;
LINK                        WCom sum1.a               eq8.b;
LINK                        qEva2 sum1.b              eq7.b;
LINK                        qCon sum1.c               eq6.b;
```

Chiller_COP.cc Chiller / SOURCE CODE

```
/* CLASS  ch_COP  "COP of chillers"

ABSTRACT

            NG-GORDON method

ABSTRACT_END
TEST_INPUT
       TEvaEnt = 290, TConEnt = 370, St=0.005, qEva=10, QLeak=0.2, R=2.5;
*/
#ifdef SPARK_TEXT
// ==== PORTS ====

PORT    TEvaEnt         "Entering water temperature at evaprator"        [K];
PORT    TConEnt         "Entering water temperature at condenser"        [K];
PORT    St              "Total internal entropy production"              [K/kW] ;
PORT    qEva            "Heat exchange at evaporator"                              [kW];
PORT    QLeak           "Equvilant heat leak"                                      [kW];
PORT    R               "total heat exchanger thermal resistance
                 =(1/C_con) + (1/C_eva)"                                           [K/kW];
PORT    COP             "chiller COP"                                              [scalar];

EQUATIONS {

COP=(1-TEvaEnt/TConEnt+TEvaEnt*St/qEva+QLeak*(TConEnt-TEvaEnt)/TConEnt/qEva+qEva/TConEnt/UAEva)^-
1*(TEvaEnt-qEva/UAEva)/TConEnt;

             }

// ==== FUNCTIONS ====
FUNCTIONS {
            COP       = chiller_COP( TEvaEnt, TConEnt, St, qEva, QLeak, R ) ;

             }
#endif /* SPARK_TEXT */
#include "spark.h"

  double
chiller_COP ( ARGS )
{
   ARGDEF(0,TEvaEnt) ;
   ARGDEF(1,TConEnt) ;
   ARGDEF(2,St) ;
   ARGDEF(3,qEva) ;
   ARGDEF(4,QLeak) ;
   ARGDEF(5,R) ;

   double COP ;

   COP = 1/(1-TEvaEnt/TConEnt+TEvaEnt*St/qEva+QLeak*(TConEnt-TEvaEnt)/TConEnt/qEva+ R*qEva/TConEnt) *
(TEvaEnt-R*qEva)/TConEnt;

   return  COP;
}
```

REFERENCES

Brandemuehl, M. J., Gabel, S., I. Andersen. 1993. A toolkit for secondary HVAC system energy calculations, HVAC2 Toolkit. Prepared for The American Society of Heating, Refrigerating and Air Conditioning Engineers. TC 4.7 Energy Calculations. Atlanta, GA. ASHRAE.

Sreedharan, P. and Haves, P**.** 2001. Comparison of Chiller Models for use in Model-Based Fault Detection, *Proc. International Conference for Enhancing Building Operations,* Austin, TX, July

Ng, K. C., H. T. Chua, W.Ong, S. S. Lee, J. M. Gordon. 1997. Diagnostics and optimization of reciprocating chillers: theory and experiment, Applied Thermal Engineering, vol. 17, no.3, pp. 263-276.

SPARK 2003. Simulation Problem Analysis and Research Kernel. Lawrence Berkeley National Laboratory and Ayres Sowell Associates, Inc. Downloadable from http://simulationresearch.lbl.gov/