APPENDIX A ASCEND IV MODELS AND LIBRARIES

A.1 MODELS

All the code presented in this appendix is in routine use by the ASCEND group at Carnegie Mellon, it is not a collection of examples used only for this thesis. Since this is working code, we ask the reader contact us if they find any errors or have any suggestions for improving our modeling libraries. This code is available in its latest revision via http://www.cs.cmu.edu/~ascend/Home.html.

The models included here have been provoked by or evolved from the models created by many ASCEND III and ASCEND IV users. The users and developers we have interacted with to produce these models are, alphabetically, Kirk Abbott, Tom Epperly, Chad Farschman, Steinar Hauan, Robert Huss, Ajay Lakshmanan, Jon Ingar Monsen, Jahn Olaf Olsen, Jennifer Perry, Boyd Safrit, Craig Schmidt, Mark Thomas, Ken Tyner, Vicente Rico-Ramirez, Art Westerberg, and Joe Zaher. These models are part of an evolving language system, and parties interested in these or other ASCEND IV models later than March 1997 should contact Professor Westerberg[1] for more information. Example models in ASCEND are normally associated with a script which demonstrates their use in the ASCEND system. Because this thesis focuses primarily on the ASCEND IV language rather than the graphic user interface, we omit these scripts.

An index of the model types defined is given on page A-147.

1. Art Westerberg, Carnegie Mellon, aw0a@andrew.cmu.edu.

A.1.1 NEWISOM

Please note that this model assumes the ASCEND III style thermodynamics library is loaded. This is the reason for the large number of **IS_REFINED_TO** statements.

Appendix A Code 1
```
(* isomerization2 benchmark input. *)
MODEL newkinetics REFINES bvp_point;
      n_eq        :== 5;
      state       IS_A Pitzer_mixture;
      state.components:== [‘H2’, ‘C2’, ‘C3’, ‘iC5’, ‘nC5’];
      state.data[‘H2’]        IS_REFINED_TO hydrogen;
      state.data[‘C2’]        IS_REFINED_TO ethane;
      state.data[‘C3’]        IS_REFINED_TO propane;
      state.data[‘iC5’]       IS_REFINED_TO i_pentane;
      state.data[‘nC5’]       IS_REFINED_TO n_pentane;
      catalyst    IS_A mass;
      F           IS_A molar_rate;
      x           IS_REFINED_TO factor;
      dydx[1..4]  IS_REFINED_TO factor;
      dydx[5]     IS_REFINED_TO molar_energy;

      y[1], state.y[‘C2’] ARE_THE_SAME;
      y[2], state.y[‘C3’] ARE_THE_SAME;
      y[3], state.y[‘iC5’] ARE_THE_SAME;
      y[4], state.y[‘nC5’] ARE_THE_SAME;
      y[5], state.H ARE_THE_SAME;

      dydx[1]*state.V*F = 1.0{cm^3/g/s}
            *catalyst
            *(exp(7.3 - 10000.0{R}/state.T)*state.y[‘iC5’] +
            exp(7.1 - 11000.0{R}/state.T)*state.y[‘nC5’])
            /(1.0 + 1.0{1/psia}*state.P
            *(0.63726-0.0010452{1/R}*state.T +
            4.2182e-7{1/R^2}*state.T^2))^2;
      dydx[2], dydx[1] ARE_THE_SAME;
      dydx[3]*state.V*F = 1.0{cm^3/g/s}
            *catalyst
            *(exp(9.9 - 8500.0{R}/state.T)*state.y[‘nC5’] -
            exp(10.7 - 10100.0{R}/state.T)*state.y[‘iC5’] -
            exp(7.3 - 10000.0{R}/state.T)*state.y[‘iC5’])
            /(1.0 + 1.0{1/psia}*state.P
            *(0.63726-0.0010452{1/R}*state.T+
            4.2182e-7{1/R^2}*state.T^2))^2;
      dydx[4]*state.V*F = 1.0{cm^3/g/s}
            *catalyst
            *(exp(10.7 - 10100.0{R}/state.T)*state.y[‘iC5’] -
            exp(9.9 - 8500.0{R}/state.T)*state.y[‘nC5’] -
            exp(7.1 - 11000.0{R}/state.T)*state.y[‘nC5’])
            /(1.0 + 1.0{1/psia}*state.P
```

```
            *(0.63726-0.0010452{1/R}*state.T+
            4.2182e-7{1/R^2}*state.T^2))^2;
      state.T := 980.0{R};
      state.P := 280.0{psia};
      state.y[‘H2’] := 0.50;
      state.y[‘C2’] := 0.04;
      state.y[‘C3’] := 0.02;
      state.y[‘iC5’] := 0.11;
      state.y[‘nC5’] := 0.33;
      catalyst := 10000.0{lbm};
      F := 800.0{lb_mole/hour};
METHODS
      METHOD clear;
            RUN state.clear;
            catalyst.fixed := FALSE;
            F.fixed := FALSE;
            x.fixed := FALSE;
            y[1..n_eq].fixed := FALSE;
            dydx[1..n_eq].fixed := FALSE;
      END clear;
      METHOD specify;
            RUN state.specify;
            state.T.fixed := FALSE;
            state.H.fixed := TRUE;
            state.y[state.components].fixed := TRUE;
            state.y[‘H2’].fixed := FALSE;
            catalyst.fixed := TRUE;
            F.fixed := TRUE;
            x.fixed := TRUE;
            dydx[1..n_eq].fixed := FALSE;
            dydx[5] := 0.0{BTU/lb_mole};
            dydx[5].fixed := TRUE;
      END specify;
END newkinetics;
MODEL isomerization2;
      nstep IS_A integer_constant;
      nstep :== 8;
      npoint IS_A integer_constant;
      npoint :== 2;
      grid[0..(nstep*npoint)] IS_A newkinetics;
      stepsize IS_A generic_real;

      (* pass our grid into the bvp integrator package *)
      collocate IS_A
      integration(nstep,npoint,’midpoint’,grid,stepsize,grid[0].n_eq);

      initial ALIASES grid[0];
      final ALIASES grid[nstep*npoint];


```

```
MAXIMIZE 92*final.state.y[‘iC5’] + 63*final.state.y[‘nC5’];

graph, temperature_graphIS_A plt_plot_integer;
graph.title :== ‘Isomerization’;
graph.XLabel :== ‘Fraction of Catalyst’;
graph.YLabel :== ‘Mole fraction’;
graph.ncurve :== 4;
graph.curve[1].legend :== ‘ethane’;
graph.curve[2].legend :== ‘propane’;
graph.curve[3].legend :== ‘isopentane’;
graph.curve[4].legend :== ‘normal pentane’;
graph.curve[1..graph.ncurve].npnt :== nstep + 1;

temperature_graph.title :== ‘Isomerization’;
temperature_graph.XLabel :== ‘Fraction of Catalyst’;
temperature_graph.YLabel :== ‘Temperature (K)’;
temperature_graph.ncurve :== 1;
temperature_graph.curve[1].legend :== ‘temperature’;
temperature_graph.curve[1].npnt :== nstep + 1;

METHODS
METHOD values;
      initial.x := 0.00;
      stepsize := 0.125;
      RUN collocate.values;
END values;
METHOD clear;
      RUN collocate.clear;
END clear;
METHOD specify;
      RUN collocate.specify;
      initial.state.T.fixed := TRUE;
      initial.state.H.fixed := FALSE;
END specify;
METHOD plot;
      FOR i IN [1..graph.ncurve] DO
      FOR j IN [1..nstep] DO
            graph.curve[i].pnt[j].x :=
                  grid[(j-1)*npoint].x;
            graph.curve[i].pnt[j].y :=
                  grid[(j-1)*npoint].y[i];
      END;
      graph.curve[i].pnt[nstep+1].x := final.x;
      graph.curve[i].pnt[nstep+1].y := final.y[i];
      END;
END plot;
METHOD temperature_plot;
      FOR j IN [1..nstep] DO
      temperature_graph.curve[1].pnt[j].x :=
                  grid[(j-1)*npoint].x;
```

```
            temperature_graph.curve[1].pnt[j].y :=
                        grid[(j-1)*npoint].state.T;
            END;
            temperature_graph.curve[1].pnt[nstep+1].x := final.x;
            temperature_graph.curve[1].pnt[nstep+1].y :=
            final.state.T;
      END temperature_plot;

END isomerization2;

```

A.1.2 TEST_PROFLASH

Appendix A Code 2

```
(* Ed’s test file. *)
MODEL test_vapor_liquid_flash;
      flash IS_A vapor_liquid_flash(feed, vapout, liqout, flash_state);

      feed IS_A liquid_stream(feed_state);
      vapout IS_A vapor_stream(vapor_state);
      liqout IS_A liquid_stream(liquid_state);
      flash_state IS_A
      td_VLE_mixture(P, T,  vapor_state, liquid_state, equilibrated);

      P      IS_A pressure;
      T      IS_A temperature;
      equilibrated IS_A boolean;
      feed_state IS_A liquid_mixture(feed_P, feed_T, liquid_options);
      liquid_state IS_A liquid_mixture(P, T, liquid_options);
      vapor_state IS_A vapor_mixture(P, T, vapor_options);

      feed_PIS_A pressure;
      feed_T IS_A temperature;
      vapor_options IS_A vapor_phase_options(ds,’Pitzer’,’Pitzer’);
      liquid_options IS_A liquid_phase_options(ds,’Rackett’,’UNIFAC’);

      ds    IS_A
      td_component_data_set([‘methanol’,’water’],’water’);

METHODS
METHOD clear;
      RUN flash.clear;
      equilibrated := TRUE;
END clear;

METHOD values;
      flash_state.alpha[‘methanol’] := 2;
      flash_state.alpha[‘n_hexane’] := 1;
      T := 298 {K};
      P := 1{atm};
      feed.f[‘methanol’,’water’] := 3{mole/s};
END values;

END test_vapor_liquid_flash;
```

A.1.3 PLOTCOL

These models create useful interfaces to the basic column library. With these models, the user needs only to specify the components and tray numbers and they get column models with built-in T,P, K-value, and composition profile plots.

Appendix A Code 3(* plotcol.par.asc *)

```
MODEL demo_plot_column(
      components IS_A set OF symbol_constant;
      reference IS_A symbol_constant;
      n_trays IS_A integer_constant;
      feed_location IS_A integer_constant;
) WHERE (
      reference IN components == TRUE;
      n_trays > 5;
      feed_location > 2;
      feed_location < n_trays - 2;
) REFINES demo_column;

      z_eq_tray_set IS_A set OF integer_constant;
      z_eq_tray_set :== Column.VLE_set;
      z_eq_tray_coord[z_eq_tray_set] IS_A real;
      z_icmp IS_A set OF integer_constant;
      z_icmp :== [1..CARD[components]];

      FOR i IN [components] CREATE
            x_curves[i] IS_A plt_curve(
                  Column.zx[i],
                  Column.Profile_x[i],
                  z_eq_tray_coord
            );
            y_curves[i] IS_A plt_curve(
                  Column.zy[i],
                  Column.Profile_y[i],
                  z_eq_tray_coord
            );
            k_curves[i] IS_A plt_curve(
                  Column.zk[i],
                  Column.Profile_kvalues_when_full_thermo[i],
                  z_eq_tray_coord
            );
      END;
      t_curve IS_A plt_curve(Column.zT,Column.Profile_T,
                              z_eq_tray_coord);
      t_curves[1] ALIASES t_curve;
      p_curve IS_A plt_curve(Column.zP, Column.Profile_P,
                              z_eq_tray_coord);
      p_curves[2] ALIASES p_curve;
```

```
      Plot_x IS_A plt_plot_symbol(components,x_curves);
      Plot_y IS_A plt_plot_symbol(components,y_curves);
      Plot_K IS_A plt_plot_symbol(components,k_curves);
      Plot_T IS_A plt_plot_integer([1],t_curves);
      Plot_P IS_A plt_plot_integer([2],p_curves);


METHODS
METHOD values;
      RUN demo_column::values;
      FOR i IN z_eq_tray_set DO
            z_eq_tray_coord[i] := i;
      END;
      t_curve.legend := ‘temperature’;
      p_curve.legend := ‘pressure’;
      FOR i IN components DO
            y_curves[i].legend := i;
            x_curves[i].legend := i;
            k_curves[i].legend := i;
      END;
      Plot_x.title := ‘Liquid Compositions’;
      Plot_y.title := ‘Vapor Compositions’;
      Plot_K.title := ‘Volatilities’;
      Plot_T.title := ‘Temperature’;
      Plot_P.title := ‘Pressure’;
      Plot_P.XLabel := ‘tray number’;
      Plot_T.XLabel := ‘tray number’;
      Plot_y.XLabel := ‘tray number’;
      Plot_x.XLabel := ‘tray number’;
      Plot_K.XLabel := ‘tray number’;
      Plot_T.YLabel := ‘{K}’;
      Plot_P.YLabel := ‘{Pascal}’;
      Plot_y.YLabel := ‘mole fraction’;
      Plot_x.YLabel := ‘mole fraction’;
      Plot_K.YLabel := ‘K value’;
      (* make composition plots cover 0 - 1 *)
      Plot_x.Ylow := 0;
      Plot_x.Yhigh := 1;
      Plot_y.Ylow := 0;
      Plot_y.Yhigh := 1;
END values;

END demo_plot_column;

MODEL mw_demo_plot_column;
        mw_column IS_A demo_plot_column([‘methanol’,’water’],
                                  ’water’,13,7);
END mw_demo_plot_column;

MODEL abc_demo_plot_column;
```

```
        abc_column IS_A
        demo_plot_column([‘benzene’,’chloroform’,’acetone’],
                        ’benzene’,13,7);
END abc_demo_plot_column;

MODEL c567_demo_plot_column;
        c567_column IS_A
        demo_plot_column([‘n_pentane’,’n_hexane’,’n_heptane’],
                        ’n_heptane’,13,7);
END c567_demo_plot_column;
```

A.2 LIBRARIES

We include here all the libraries required for the examples in this appendix and elsewhere in this thesis. We have stripped the copyright notice and other distracting headers from these libraries so that the reader can more easily focus on the content.

A.2.1 SYSTEM

System defines the basic variable types expected by current solver clients of the ASCEND environment. If a client requires more information about variables than is in the current definition of solver_var, this library is easily edited to add the needed variable attributes without requiring any of the other libraries to be modified.

Appendix A Code 4

```
(* system.lib *)
RELATION_DEFINITION
      included IS_A boolean;
      included := TRUE;
END;

LOGREL_DEFINITION
      included IS_A boolean;
      included := TRUE;
END;

ATOM solver_var REFINES real DEFAULT 0.5 {?};
      fixed       IS_A boolean;
      lower_bound IS_A real;
      nominal     IS_A real;
      upper_bound IS_A real;

      fixed := FALSE;
      lower_bound := -1e20 {?};
      upper_bound := 1e20 {?};
      nominal := 0.5 {?};
END solver_var;

ATOM boolean_var REFINES boolean
      DEFAULT TRUE;
      nominalIS_A boolean;
      fixed IS_A boolean;
      fixed := FALSE;
      nominal := TRUE;
END boolean_var;

ATOM generic_real REFINES solver_var DIMENSIONLESS DEFAULT 0.5;
      lower_bound := -1e20;
      upper_bound := 1e20;
      nominal := 0.5;
END generic_real;

ATOM solver_int REFINES solver_var
      DIMENSIONLESS
      DEFAULT 0.0;
```

```
      relaxed IS_A boolean;
      lower_bound := 0.0;
      upper_bound := 1000000.0;
      nominal := 0.5;
END solver_int;

ATOM solver_binary REFINES solver_int DIMENSIONLESS DEFAULT 0.0;
      lower_bound := 0.0;
      upper_bound := 1.0;
      nominal := 0.5;
END solver_binary;

ATOM solver_semi REFINES solver_var
      DEFAULT 1.0 {?};
      is_zero IS_A boolean;
      relaxed IS_A boolean;
      lower_bound := 1 {?};
      upper_bound := 1e20 {?};
      nominal := 1.0 {?};
END solver_semi;
```

A.2.2 ATOMS

Appendix A Code 5(* atoms.lib *)

```
(*  G E N E R I C   C O N S T A N T S  ( C H E M .  E .   B I A S )
  -------------------------------------------------------------  *)

      CONSTANT constant
      (* any sloppiness about what is a constant will yield a
      wild real constant *)
      REFINES real_constant;

      (* dimensionless *)

      CONSTANT critical_compressibility REFINES real_constant
      DIMENSIONLESS;

      CONSTANT acentric_factor REFINES real_constant DIMENSIONLESS;

      CONSTANT UNIFAC_size REFINES real_constant DIMENSIONLESS;

      CONSTANT Wilson_constant REFINES real_constant DIMENSIONLESS;

      CONSTANT vapor_pressure_constant REFINES real_constant
      DIMENSIONLESS;

      CONSTANT factor_constant REFINES real_constant DIMENSIONLESS;

      (* molecular weight *)
      CONSTANT molar_weight_constant
      REFINES real_constant DIMENSION M/Q;

      (* temperatures *)
      CONSTANT temperature_constant
      REFINES real_constant DIMENSION TMP;

      CONSTANT boiling_temperature
      REFINES temperature_constant;

      CONSTANT critical_temperature
      REFINES temperature_constant;

      CONSTANT reference_temperature
      REFINES temperature_constant;

      CONSTANT UNIFAC_a
      REFINES temperature_constant;

      (* pressures *)
      CONSTANT pressure_constant
      REFINES real_constant DIMENSION M/L/T^2;
```

```
CONSTANT critical_pressure
REFINES pressure_constant;

CONSTANT reference_pressure
REFINES pressure_constant;

(* molar volumes *)
CONSTANT molar_volume_constant
REFINES real_constant DIMENSION L^3/Q;

CONSTANT critical_volume
REFINES molar_volume_constant;

CONSTANT reference_molar_volume
REFINES molar_volume_constant;

(* mass densities *)
CONSTANT reference_mass_density
REFINES real_constant DIMENSION M/L^3;

(* molar energies *)
CONSTANT molar_energy_constant
REFINES real_constant DIMENSION M*L^2/T^2/Q;

CONSTANT reference_molar_energy
REFINES molar_energy_constant;

CONSTANT enthalpy_of_formation_constant
REFINES molar_energy_constant;

CONSTANT free_energy_of_formation_constant
REFINES molar_energy_constant;

CONSTANT heat_of_vaporization_constant
REFINES molar_energy_constant;

CONSTANT Wilson_energy_constant
REFINES molar_energy_constant;

(* molar entropies *)
CONSTANT molar_entropy_constant
REFINES real_constant DIMENSION M*L^2/T^2/Q/TMP;

CONSTANT reference_molar_entropy
REFINES molar_entropy_constant;

(* other strange correlation coefficients *)
CONSTANT heat_capacity_constant
REFINES real_constant;
```

```
      CONSTANT heat_capacity_a
      REFINES heat_capacity_constant DIMENSION M*L^2/T^2/Q/TMP;

      CONSTANT heat_capacity_b
      REFINES heat_capacity_constant DIMENSION M*L^2/T^2/Q/TMP^2;

      CONSTANT heat_capacity_c
      REFINES heat_capacity_constant DIMENSION M*L^2/T^2/Q/TMP^3;

      CONSTANT heat_capacity_d
      REFINES heat_capacity_constant DIMENSION M*L^2/T^2/Q/TMP^4;

(* template:
      CONSTANT
      REFINES real_constant DIMENSION;
*)

(*  U N I V E R S A LC O N S T A N T S
  --------------------------------------  *)

      UNIVERSAL CONSTANT gas_constant
      (* DIMENSION M*L^2/T^2/Q/TMP *)
      REFINES real_constant :== 1{GAS_C};

      UNIVERSAL CONSTANT gravity_constant
      (* DIMENSION L/T^2 *)
      REFINES real_constant :== 1{EARTH_G};

      UNIVERSAL CONSTANT circle_constant
      REFINES real_constant :== 1{PI};

      UNIVERSAL CONSTANT speed_of_light
      REFINES real_constant :== 1{LIGHT_C};

      UNIVERSAL CONSTANT planck_constant
      (* DIMENSION M*L^2/T *)
      REFINES real_constant :== 1{PLANCK_C};

      UNIVERSAL CONSTANT avogadro_constant
      REFINES real_constant :== 1{AVOGADRO_C};

      UNIVERSAL CONSTANT permittivity_constant
      (* DIMENSION E^2*T^4/M/L^3 *)
      REFINES real_constant :== 1{EPSILON0};

      UNIVERSAL CONSTANT permeability_constant
      REFINES real_constant :== 1{MU0};

      UNIVERSAL CONSTANT electron_charge
```

```
      REFINES real_constant :== 1{eCHARGE};

      UNIVERSAL CONSTANT electron_mass
      REFINES real_constant :== 1{eMASS};

      UNIVERSAL CONSTANT proton_mass
      REFINES real_constant :== 1{pMASS};

   (* B O O L E A N S   *)

ATOM start_true REFINES boolean
      DEFAULT  TRUE;
END start_true;

ATOM start_false REFINES boolean
      DEFAULT  FALSE;
END start_false;


   (* P A R A M E T E R S *)

UNIVERSAL ATOM scaling_constant REFINES real
      (* not really a constant but a parameter *)
      DIMENSIONLESS
      DEFAULT 1e8;
END scaling_constant;

UNIVERSAL ATOM ode_counter REFINES integer
      DIMENSIONLESS
      DEFAULT 1;
END ode_counter;

UNIVERSAL ATOM obs_counter REFINES integer
      DIMENSIONLESS
      DEFAULT 1;
END obs_counter;

ATOM real_parameter REFINES real;
END real_parameter;

ATOM length_parameter REFINES real_parameter
      DIMENSION  L
      DEFAULT 1.0 {m};
END length_parameter;


(*  D I M E N S I O N L E S S  Q U A N T I T I E S
  ---------------------------------------------  *)

      ATOM positive_variable REFINES solver_var
```

```
    (* one for the gams folks *)
          DIMENSIONLESS
          DEFAULT 1.0;
          lower_bound := 0.0;
          upper_bound := 1e20;
          nominal := 1.0;
    END positive_variable;

    ATOM factor REFINES solver_var
          DIMENSIONLESS
          DEFAULT 1.0;
          lower_bound := -1e50;
          upper_bound := 1e50;
          nominal := 1.0;
    END factor;

    ATOM variable REFINES solver_var
    DIMENSIONLESS;
    END variable;


    ATOM fraction REFINES solver_var
          DIMENSIONLESS
          DEFAULT 0.5;
          nominal := 1.0;
          lower_bound := 0.0;
          upper_bound := 1.0;
    END fraction;

    ATOM positive_factor REFINES factor;
          lower_bound := 0.0;
    END positive_factor;

    ATOM small_factor REFINES factor;
          lower_bound := -10.0;
          upper_bound := 10.0;
    END small_factor;

    ATOM small_positive_factor REFINES factor;
          lower_bound := 0.0;
          upper_bound := 10.0;
    END small_positive_factor;

    ATOM reduced_pressure REFINES factor;
    END reduced_pressure;

(*  S U B S T I T U T I O N   V A R I A B L E S
  -----------------------------------------  *)
    ATOM exp_sub REFINES factor ;
          lower_bound := -1e50;
```

```
            upper_bound := 100;
            nominal := 1.0;
      END exp_sub;

      ATOM power_sub REFINES factor;
            lower_bound := -25;
            upper_bound := 25;
            nominal := 1.0;
      END power_sub;

(*  T E M P E R A T U R E
  ---------------------  *)

      ATOM temperature REFINES solver_var
            DIMENSION TMP
            DEFAULT 298.0{K};
            lower_bound := 1.0e-6{K};
            upper_bound := 10000{K};
            nominal := 298.0{K};
      END temperature;

      ATOM inverse_temperature REFINES solver_var
            DIMENSION 1/TMP
            DEFAULT 0.00366099{1/K};
            lower_bound := 0.0{1/K};
            upper_bound := 1e50{1/K};
            nominal := 0.00366099{1/K};
      END inverse_temperature;

(*  P R E S S U R E
  ---------------  *)

      ATOM pressure REFINES solver_var
            DIMENSION M/L/T^2
            DEFAULT 1.0{atm};
            lower_bound := 0.001{Pa};
            upper_bound := 5000{atm};
            nominal := 1.0{atm};
      END pressure;

      ATOM vapor_pressure REFINES pressure
            DIMENSION M/L/T^2
            DEFAULT 1.0{atm};
            lower_bound := 0.001{Pa};
            upper_bound := 5000{atm};
            nominal := 0.5{atm};
      END vapor_pressure;

      ATOM k_constant REFINES solver_var
      (* what IS this ? ? ? *)
```

```
        DIMENSION T^2/L^5
        DEFAULT 1.0 {s^2/ft^5};
        lower_bound := 0.001 {s^2/ft^5};
        upper_bound := 5000 {s^2/ft^5};
        nominal := 1.0 {s^2/ft^5};
    END k_constant;


(*  M A S S / M O L E  Q U A N T I T I E S
  --------------------------------------  *)

    ATOM molar_mass REFINES solver_var
        DIMENSION M/Q
        DEFAULT 100.0{g/g_mole};
        lower_bound := 0.0{g/g_mole};
        upper_bound := 1e9{g/g_mole};
        nominal := 100.0{g/g_mole};
    END molar_mass;

    ATOM mass REFINES solver_var
        DIMENSION M
        DEFAULT 10.0{kg};
        lower_bound := 0.0{kg};
        upper_bound := 1e50{kg};
        nominal := 10.0{kg};
    END mass;

    ATOM mole REFINES solver_var
        DIMENSION Q
        DEFAULT 10.0{lb_mole};
        lower_bound := 0.0{lb_mole};
        upper_bound := 1e50{lb_mole};
        nominal := 10.0{lb_mole};
    END mole;

    ATOM mass_rate REFINES solver_var
        DIMENSION M/T
        DEFAULT 50{g/s};
        lower_bound := 0.0{g/s};
        upper_bound := 1e50{g/s};
        nominal := 100.0{g/s};
    END mass_rate;

    ATOM molar_rate REFINES solver_var
        DIMENSION Q/T
        DEFAULT 100.0{lb_mole/hour};
        lower_bound := 0.0{lb_mole/hour};
        upper_bound := 1e50{lb_mole/hour};
        nominal := 100.0{lb_mole/hour};
    END molar_rate;
```

```
    ATOM mole_fraction REFINES fraction
          DIMENSIONLESS
          DEFAULT 0.5;
          lower_bound := 0.0;
          nominal := 0.3;
          upper_bound := 1.0;
    END mole_fraction;

    ATOM mass_fraction REFINES fraction
          DIMENSIONLESS
          DEFAULT 0.5;
          lower_bound := 0.0;
          nominal := 0.3;
          upper_bound := 1.0;
    END mass_fraction;


(*  V O L U M E  Q U A N T I T I E S
  --------------------------------  *)

    ATOM molar_volume REFINES solver_var
          DIMENSION L^3/Q
          DEFAULT 1000.0{cm^3/g_mole};
          lower_bound := 0.0{cm^3/g_mole};
          upper_bound := 1e50{cm^3/g_mole};
          nominal := 1000.0{cm^3/g_mole};
    END molar_volume;

    ATOM volume REFINES solver_var
          DIMENSION L^3
          DEFAULT 100.0{ft^3};
          lower_bound := 0.0{ft^3};
          upper_bound := 1e50{ft^3};
          nominal := 100.0{ft^3};
    END volume;

    ATOM volume_rate REFINES solver_var
          DIMENSION L^3/T
          DEFAULT 100.0{gpm};
          lower_bound := 0.0{gpm};
          upper_bound := 1e50{gpm};
          nominal := 100.0{gpm};
    END volume_rate;

    ATOM volume_rate_square REFINES solver_var
          DIMENSION L^6/T^2
          DEFAULT 100.0{ft^6/s^2};
          lower_bound := 0.0{ft^6/s^2};
          upper_bound := 1e50{ft^6/s^2};
```

```
        nominal := 100 {ft^6/s^2};
    END volume_rate_square;

    ATOM volume_expansivity REFINES solver_var
        DIMENSION 1/TMP
        DEFAULT 0.001{1/K};
        lower_bound := 0.0{1/K};
        upper_bound := 1e50{1/K};
        nominal := 0.001{1/K};
    END volume_expansivity;


(*  D E N S I T Y  Q U A N T I T I E S
  ---------------------------------  *)

    ATOM molar_density REFINES solver_var
        DIMENSION Q/L^3
        DEFAULT 0.1{mole/m^3};
        lower_bound := 0.0{mole/m^3};
        upper_bound := 1e50{mole/m^3};
        nominal := 0.1{mole/m^3};
    END molar_density;

    ATOM mass_density REFINES solver_var
        DIMENSION M/L^3
        DEFAULT 1.0{g/cm^3};
        lower_bound := 0.0{g/cm^3};
        upper_bound := 1e50{g/cm^3};
        nominal := 1.0{g/cm^3};
    END mass_density;


(*  E N E R G Y  Q U A N T I T I E S
  -----------------------------------  *)

    ATOM molar_energy REFINES solver_var
        DIMENSION M*L^2/T^2/Q
        DEFAULT 10000.0{BTU/lb_mole};
        lower_bound := -1e50{BTU/lb_mole};
        upper_bound := 1e50{BTU/lb_mole};
        nominal := 10000.0{BTU/lb_mole};
    END molar_energy;

    ATOM energy REFINES solver_var
        DIMENSION M*L^2/T^2
        DEFAULT 100000.0{BTU};
        lower_bound := -1e50{BTU};
        upper_bound := 1e50{BTU};
        nominal := 100000.0{BTU};
    END energy;
```

```
    ATOM energy_rate REFINES solver_var
          DIMENSION M*L^2/T^3
          DEFAULT 100000.0{BTU/hour};
          lower_bound := -1e50{BTU/hour};
          upper_bound := 1e50{BTU/hour};
          nominal := 100000.0{BTU/hour};
    END energy_rate;

    ATOM heat_capacity REFINES solver_var
          DIMENSION M*L^2/T^2/Q/TMP
          DEFAULT 1.00e5{J/mole/K};
          lower_bound := 0.0{J/mole/K};
          upper_bound := 1e60{J/mole/K};
          nominal := 1.00e5{J/mole/K};
    END heat_capacity;

    ATOM molar_energy_rate REFINES solver_var
          DIMENSION M*L^2/T^3/Q
          DEFAULT 0 {BTU/lb_mole/hr};
          lower_bound := -1e50 {BTU/lb_mole/hr};
          upper_bound := 1e50 {BTU/lb_mole/hr};
          nominal := 10000.0 {BTU/lb_mole/hr};
    END molar_energy_rate;

(*  E N T R O P Y   Q U A N T I T I E S
  ----------------------------------  *)

    ATOM molar_entropy REFINES solver_var
          DIMENSION M*L^2/T^2/Q/TMP
          DEFAULT 100.0{BTU/lb_mole/R};
          lower_bound := -1e50{BTU/lb_mole/R};
          upper_bound := 1e50{BTU/lb_mole/R};
          nominal := 100.0{BTU/lb_mole/R};
    END molar_entropy;

    ATOM entropy REFINES solver_var
          DIMENSION M*L^2/T^2/TMP
          DEFAULT 1000.0{BTU/R};
          lower_bound := -1e50{BTU/R};
          upper_bound := 1e50{BTU/R};
          nominal := 1000.0{BTU/R};
    END entropy;

    ATOM entropy_rate REFINES solver_var
          DIMENSION M*L^2/T^3/TMP
          DEFAULT 1000.0{BTU/hour/R};
          lower_bound := -1e50{BTU/hour/R};
          upper_bound := 1e50{BTU/hour/R};
          nominal := 1000.0{BTU/hour/R};
```

```
      END entropy_rate;



(*  E Q U I L I B R I U M  Q U A N T I T I E S
  ------------------------------------------  *)

      ATOM partition_coefficient REFINES factor (* new *)
            DEFAULT 1.0;
            lower_bound := 1.0e-10;
            upper_bound := 30.0;
            nominal := 1.0;
      END partition_coefficient;

      ATOM relative_volatility REFINES partition_coefficient;
      END relative_volatility;


(*  M O N E T A R Y  Q U A N T I T I E S
  ------------------------------------  *)

      ATOM monetary_unit REFINES solver_var
            DIMENSION C
            DEFAULT 100.0{USdollar};
            lower_bound := -1e50{USdollar};
            upper_bound := 1e50{USdollar};
            nominal := 100.0{USdollar};
      END monetary_unit;

ATOM cost_per_volume REFINES solver_var
            DIMENSION C/L^3
            DEFAULT 1.0{USdollar/gallon};
            lower_bound := 0.0{USdollar/gallon};
            upper_bound := 1e50{USdollar/gallon};
            nominal := 1.0{USdollar/gallon};
END cost_per_volume;

ATOM cost_per_mass REFINES solver_var
            DIMENSION C/M
            DEFAULT 1.0{USdollar/lbm};
            lower_bound := 0.0{USdollar/lbm};
            upper_bound := 1e50{USdollar/lbm};
            nominal := 1.0{USdollar/lbm};
END cost_per_mass;

ATOM cost_per_time REFINES solver_var
            DIMENSION C/T
            DEFAULT 1.0{USdollar/min};
            lower_bound := 0.0{USdollar/min};
            upper_bound := 1e50{USdollar/min};
```

```
            nominal := 1.0{USdollar/min};
END cost_per_time;

(*  S U R V E Y I N G   Q U A N T I T I E S
  ---------------------------------------  *)

      ATOM distance REFINES solver_var
            DIMENSION L
            DEFAULT 10.0{ft};
            lower_bound := 0.0{ft};
            upper_bound := 1e50{ft};
            nominal := 10.0{ft};
      END distance;

      ATOM area REFINES solver_var
            DIMENSION L^2
            DEFAULT 10.0{ft^2};
            lower_bound := 0.0{ft^2};
            upper_bound := 1e50{ft^2};
            nominal := 10.0{ft^2};
      END area;

      ATOM angle REFINES solver_var
            DIMENSION P
            DEFAULT 1 {rad};
            lower_bound := -1e50 {rad};
            upper_bound := 1e50 {rad};
            nominal := 1 {rad};
      END angle;

      ATOM solid_angle REFINES solver_var
            DIMENSION S
            DEFAULT 1 {srad};
            lower_bound := -1e50 {srad};
            upper_bound := 1e50 {srad};
            nominal := 1 {srad};
      END solid_angle;

(*  M O T I O N   Q U A N T I T I E S
  ---------------------------------  *)

      ATOM time REFINES solver_var
            DIMENSION T
            DEFAULT 60.0{s};
            lower_bound := -1e50{s};
            upper_bound := 1e50{s};
            nominal := 60.0{s};
      END time;

      ATOM speed REFINES solver_var
```

```
        DIMENSION L/T
        DEFAULT 3.0{ft/s};
        lower_bound := -1e50{m/s};
        upper_bound := 1e50{m/s};
        nominal := 1.0{m/s};
    END speed;

    ATOM acceleration REFINES solver_var
        DIMENSION L/T^2
        DEFAULT 9.8{m/s^2};
        lower_bound := -1e50{m/s^2};
        upper_bound := 1e50{m/s^2};
        nominal := 9.8{m/s^2};
    END acceleration;

    ATOM frequency REFINES solver_var
        DIMENSION 1/T
        DEFAULT 60.0{1/s};
        lower_bound := 0.0{1/s};
        upper_bound := 1e50{1/s};
        nominal := 60.0{1/s};
    END frequency;


(*  T R A N S P O R T   Q U A N T I T I E S
  ---------------------------------------  *)

    ATOM viscosity REFINES solver_var
        DIMENSION M/L/T
        DEFAULT 1.0{cP};
        lower_bound := 0.0{cP};
        upper_bound := 1e50{cP};
        nominal := 1.0{cP};
    END viscosity;

    ATOM thermal_conductivity REFINES solver_var
        DIMENSION M*L/T^3/TMP
        DEFAULT 0.1{BTU/hour/ft/R};
        lower_bound := 0.0{BTU/hour/ft/R};
        upper_bound := 1e50{BTU/hour/ft/R};
        nominal := 0.1{BTU/hour/ft/R};
    END thermal_conductivity;

    ATOM diffusivity REFINES solver_var
        DIMENSION L^2/T
        DEFAULT 1.0{cm^2/s};
        lower_bound := 0.0{cm^2/s};
        upper_bound := 1e50{cm^2/s};
        nominal := 1.0{cm^2/s};
    END diffusivity;
```

```
(*  E L E C T R O - M A G N E T I C   Q U A N T I T I E S
  -----------------------------------------------------  *)

      ATOM voltage REFINES solver_var
            DIMENSION M*L^2/(T^3*E)
            DEFAULT 10.0{volt};
            lower_bound := -1e50{volt};
            upper_bound := 1e50{volt};
            nominal := 10.0{volt};
      END voltage;

      ATOM resistance REFINES solver_var
            DIMENSION M*L^2/(E^2*T^3)
            DEFAULT 10.0{ohm};
            lower_bound := -1e50{ohm};
            upper_bound := 1e50{ohm};
            nominal := 10.0{ohm};
      END resistance;

      ATOM current REFINES solver_var
            DIMENSION E
            DEFAULT 1.0{amp};
            lower_bound := -1e50{amp};
            upper_bound := 1e50{amp};
            nominal := 1.0{amp};
      END current;

      ATOM capacitance REFINES solver_var
            DIMENSION E^2*T^4/M/L^2
            DEFAULT 1.0{coulomb/volt};
            lower_bound := -1e50{coulomb/volt};
            upper_bound := 1e50{coulomb/volt};
            nominal := 1.0{coulomb/volt};
      END capacitance;

      ATOM inductance REFINES solver_var
            DIMENSION M*L^2/E^2/T^2
            DEFAULT 1.0{volt*s/amp};
            lower_bound := -1e50{volt*s/amp};
            upper_bound := 1e50{volt*s/amp};
            nominal := 1.0{volt*s/amp};
      END inductance;

      ATOM magnetic_field REFINES solver_var
            DIMENSION E/L
            DEFAULT 1.0{amp/m};
            lower_bound := -1e50{amp/m};
            upper_bound := 1e50{amp/m};
```

```
      nominal := 1.0{amp/m};
END magnetic_field;

ATOM electric_field REFINES solver_var
      DIMENSION M*L/E/T^3
      DEFAULT 1.0{volt/m};
      lower_bound := -1e50{volt/m};
      upper_bound := 1e50{volt/m};
      nominal := 1.0{volt/m};
END electric_field;
```

A.2.3 COMPONENTS

This library contains only four reusable types: UNIFAC_constants, component_constants, td_component_constants, and td_component_data_set. `td_component_constants` is our properties database model. This library demonstrates the use of a UNIVERSAL model, NOTES and large SELECT statements. As noted in the library models, we do not claim this library is error free.

Appendix A Code 6

```
(* components.lib *)
UNIVERSAL MODEL UNIFAC_constants;

NOTES 'reference' SELF {
The reference for the bulk of this MODEL is
Reid, Prausnitz & Poling, Properties of Gases and Liquids,
1986, Chapter 8.
}
'disclaimer' SELF {
The authors of ASCEND and Carnegie Mellon University make
absolutely NO WARRANTY about the accuracy of this transcription
of the RPP data or of the original data itself, nor do they
provide any guarantee that the data here represented is
suitable for any purpose academic or commercial.
}
'applicability' SELF {
The frontiers of group contribution methods have moved on
well past the data presented here and the correlations
it is used with. Users with a penchant for thermodynamic
detail are advised to MODEL their own favorite correlations.
}
'usage' SELF {
The local name of every instance of this MODEL
(since it is UNIVERSAL) should always be uc.
In the comments that follow, we will use uc where applicable.
}
END NOTES;

      groups IS_A set OF integer_constant;
      groups   :== [1..9,11,23];

NOTES
'is' groups {
uc.groups is the main group list (the ones we support) from RPP
Table 8.21 column 1.
Currently supported groups are (in RPP Main index)
1,2,3,4,5,6,7,8,9,23, and 11
}
'is' sub {
```

```
uc.sub[i IN uc.groups] is the set of secondary groups for the ith
main group. This is the column 2 data of RPP Table 8-21.
The UNION of uc.sub[i] is uc.subgroups, basically the set of
rows in Table 8-21 that we have entered.
}
END NOTES;

      sub[groups]              IS_A set OF symbol_constant;
      sub[1]   :== [‘CH3’, ‘CH2’, ‘CH’, ‘C’];
      sub[2]   :== [‘CH2=CH’, ‘CH=CH’, ‘CH2=C’, ‘CH=C’, ‘C=C’];
      sub[3]   :== [‘ACH’, ‘AC’];
      sub[4]   :== [‘ACCH3’, ‘ACCH2’, ‘ACCH’];
      sub[5]   :== [‘OH’];
      sub[6]   :== [‘CH3OH’];
      sub[7]   :== [‘H2O’];
      sub[8]   :== [‘ACOH’];
      sub[9]   :== [‘CH3CO’, ‘CH2CO’];
      sub[23]  :== [‘CHCl3’, ‘CCl3’];
      sub[11]  :== [‘CH3COO’, ‘CH2COO’];
      subgroups                  IS_A set OF symbol_constant;

      subgroups “the total list of secondary groups we accomodate”
      :== UNION[sub[i] | i IN groups];
      group[subgroups]IS_A integer_constant;
NOTES
‘is’ groups {
Next we have a horribly named array, group, that lets us look up the
main group given a particular subgroup. So, for example,
if we want to know the main group of ‘CHCl3’, then
group[‘CHCl3’] will give it to us: ‘CCl3’.
Note, then, that the assignments which follow in most
cases expand to have several left-hand-sides.
}
END NOTES;

      FOR i IN groups CREATE
            group[sub[i]]:== i;
      END;

      a[groups][groups] IS_A UNIFAC_a;
NOTES
‘is’ groups {
Now we have uc.a[m][n], a portion of RPP Table 8-22, the main group
interaction matrix which is asymmetric and in principle
dense but in practice rather sparse.
Adding a main group to this MODEL means adding a row and
column to uc.a, which is a non-trivial exercise until we
get a proper tabular assignment syntax implemented.
Note that such an syntax has been completely scoped but
we’ve just not yet got round to implementation.
```

```
When we do implement, this will be the first test MODEL.
a[i][j] should have i and j being integer_constant!
}
END NOTES;

(* we would like the ascend TABLE syntax or a database
 * external lookup to handle this data entry.
 *)
(* row 1 *)
      a[1][1]     :==   0.0{K};
      a[1][2]     :==   86.020{K};
      a[1][3]     :==   61.13{K};
      a[1][4]     :==   76.50{K};
      a[1][5]     :==   986.5{K};
      a[1][6]     :==   697.2{K};
      a[1][7]     :==   1318.0{K};
      a[1][8]     :==   1333.0{K};
      a[1][9]     :==   476.4{K};
      a[1][23]:== 24.90{K};
      a[1][11]:== 232.1{K};

(* row 2 *)
      a[2][1]     :==   2520{K};
      a[2][2]     :==   0.0{K};
      a[2][3]     :==   340.7{K};
      a[2][4]     :==   4102{K};
      a[2][5]     :==   693.9{K};
      a[2][6]     :==   1509.0{K};
      a[2][7]     :==   634.2{K};
      a[2][8]     :==   547.4{K};
      a[2][9]     :==   524.5{K};
      a[2][23]:== 4584.0{K};
      a[2][11]:== 71.23{K};

(* row 3 *)
      a[3][1]     :==   -11.12{K};
      a[3][2]     :==   3.446{K};
      a[3][3]     :==   0.0{K};
      a[3][4]     :==   167.0{K};
      a[3][5]     :==   636.10{K};
      a[3][6]     :==   637.3{K};
      a[3][7]     :==   903.8{K};
      a[3][8]     :==   1329.0{K};
      a[3][9]     :==   25.77{K};
      a[3][23]:== -231.9{K};
      a[3][11]:== 5.994{K};

(* row 4 *)
      a[4][1]     :==   -69.70{K};
      a[4][2]     :==   -113.6{K};
```

```
    a[4][3]     :==   -146.80{K};
    a[4][4]     :==   0.0{K};
    a[4][5]     :==   803.20{K};
    a[4][6]     :==   603.2{K};
    a[4][7]     :==   5695.00{K};
    a[4][8]     :==   547.4{K};
    a[4][9]     :==   -52.10{K};
    a[4][23]:== -12.14{K};
    a[4][11]:== 5688.0{K};

(* row 5 *)
    a[5][1]     :==   156.40{K};
    a[5][2]     :==   475.000{K};
    a[5][3]     :==   89.60{K};
    a[5][4]     :==   25.82{K};
    a[5][5]     :==   0.0{K};
    a[5][6]     :==   -137.1{K};
    a[5][7]     :==   353.50{K};
    a[5][8]     :==   -259.7{K};
    a[5][9]     :==   84.0{K};
    a[5][23]:== -98.12{K};
    a[5][11]:== 101.1{K};

(* row 6 *)
    a[6][1]     :==   16.51{K};
    a[6][2]     :==   -12.520{K};
    a[6][3]     :==   -50.00{K};
    a[6][4]     :==   -44.50{K};
    a[6][5]     :==   249.1{K};
    a[6][6]     :==   0.0{K};
    a[6][7]     :==   -181.0{K};
    a[6][8]     :==   -101.7{K};
    a[6][9]     :==   23.39{K};
    a[6][23]:== -139.4{K};
    a[6][11]:== -10.72{K};

(* row 7 *)
    a[7][1]     :==   300.00{K};
    a[7][2]     :==   496.100{K};
    a[7][3]     :==   362.30{K};
    a[7][4]     :==   377.60{K};
    a[7][5]     :==   -229.10{K};
    a[7][6]     :==   289.6{K};
    a[7][7]     :==   0.0{K};
    a[7][8]     :==   324.5{K};
    a[7][9]     :==   -195.40{K};
    a[7][23]:== 353.7{K};
    a[7][11]:== 14.42{K};

(* row 8 *)
```

```
      a[8][1]     :==   275.8{K};
      a[8][2]     :==   217.5{K};
      a[8][3]     :==   25.34{K};
      a[8][4]     :==   244.2{K};
      a[8][5]     :==   -451.6{K};
      a[8][6]     :==   -265.2{K};
      a[8][7]     :==   -601.8{K};
      a[8][8]     :==   0.0{K};
      a[8][9]     :==   -356.1{K};
      a[8][23]:== 0.0{K};
      a[8][11]:== -449.4{K};

(* row 9 *)
      a[9][1]     :==   26.76{K};
      a[9][2]     :==   42.920{K};
      a[9][3]     :==   140.10{K};
      a[9][4]     :==   365.80{K};
      a[9][5]     :==   164.5{K};
      a[9][6]     :==   108.7{K};
      a[9][7]     :==   472.5{K};
      a[9][8]     :==   -133.1{K};
      a[9][9]     :==   0.0{K};
      a[9][23]:== -354.6{K};
      a[9][11]:== -213.7{K};

(* row 23 *)
      a[23][1]:== 36.70{K};
      a[23][2]:== 51.060{K};
      a[23][3]:== 288.5{K};
      a[23][4]:== 33.61{K};
      a[23][5]    :==   742.1{K};
      a[23][6]:== 649.1{K};
      a[23][7]:== 826.7{K};
      a[23][8]:== 0.0{K};
      a[23][9]:== 552.1{K};
      a[23][23]:==0.0{K};
      a[23][11]:==176.5{K};

(* row 11 *)
      a[11][1]:== 114.8{K};
      a[11][2]:== 132.1{K};
      a[11][3]:== 85.84{K};
      a[11][4]:== -170.0{K};
      a[11][5]    :==   245.4{K};
      a[11][6]:== 249.6{K};
      a[11][7]:== 10000.0{K};
      a[11][8]:== -36.72{K};
      a[11][9]:== 372.2{K};
      a[11][23]:==-209.7{K};
      a[11][11]:==0.0{K};
```

```
      R[subgroups], Q[subgroups]   IS_A UNIFAC_size;
NOTES
'is' R,Q {
R and Q are the unifac volume and surface contributions,
indexed by subgroup, columns 4 & 5 in RPP Table 8-21
}
END NOTES;
      R['CH3']     :==   0.9011;
      R['CH2']     :==   0.6744;
      R['CH']            :==   0.4469;
      R['C']             :==   0.2195;
      R['CH2=CH'] :==   1.3454;
      R['CH=CH']  :==   1.1167;
      R['CH2=C']  :==   1.1173;
      R['CH=C']   :==   0.8886;
      R['C=C']    :==   0.6605;
      R['ACH']    :==   0.5313;
      R['AC']            :==   0.3652;
      R['ACCH3']  :==   1.2663;
      R['ACCH2']  :==   1.0396;
      R['ACCH']   :==   0.8121;
      R['OH']            :==   1.000;
      R['CH3OH']  :==   1.4311;
      R['H2O']    :==   0.9200;
      R['ACOH']   :==   0.8952;
      R['CH3CO']  :==   1.6724;
      R['CH2CO']  :==   1.4457;
      R['CHCl3']  :==   2.87007;
      R['CCl3']   :==   2.6401;
      R['CH3COO'] :==   1.9031;
      R['CH2COO'] :==   1.6764;

      Q['CH3']     :==   0.848;
      Q['CH2']     :==   0.540;
      Q['CH']            :==   0.228;
      Q['C']             :==   0.0;
      Q['CH2=CH'] :==   1.176;
      Q['CH=CH']  :==   0.867;
      Q['CH2=C']  :==   0.988;
      Q['CH=C']   :==   0.676;
      Q['C=C']    :==   0.485;
      Q['ACH']    :==   0.40;
      Q['AC']            :==   0.120;
      Q['ACCH3']  :==   0.968;
      Q['ACCH2']  :==   0.660;
      Q['ACCH']   :==   0.348;
      Q['OH']            :==   1.200;
      Q['CH3OH']  :==   1.432;
      Q['H2O']    :==   1.400;
```

```
      Q[‘ACOH’]   :==   0.680;
      Q[‘CH3CO’]  :==   1.488;
      Q[‘CH2CO’]  :==   1.180;
      Q[‘CHCl3’]  :==   2.410;
      Q[‘CCl3’]   :==   2.184;
      Q[‘CH3COO’] :==   1.728;
      Q[‘CH2COO’] :==   1.420;

END UNIFAC_constants;


MODEL component_constants;

      formula            IS_A  symbol_constant;
      subgroups          IS_A  set OF symbol_constant;
      groups             IS_A  set OF integer_constant;

      wilson_set  IS_A  set OF symbol_constant;
      uc                 IS_A  UNIFAC_constants;
      mw                 IS_A  molar_weight_constant;
      Tb                 IS_A  boiling_temperature;
      Tc                 IS_A  critical_temperature;
      Pc                 IS_A  critical_pressure;
      Vc                 IS_A  critical_volume;
      Zc                 IS_A  critical_compressibility;
      omega              IS_A  acentric_factor;
      cpvapa             IS_A  heat_capacity_a;
      cpvapb             IS_A  heat_capacity_b;
      cpvapc             IS_A  heat_capacity_c;
      cpvapd             IS_A  heat_capacity_d;
      Hf                 IS_A  enthalpy_of_formation_constant;
      Gf                 IS_A  free_energy_of_formation_constant;
      vpa, vpb, vpc, vpdIS_A  vapor_pressure_constant;
      vp_correlation     IS_A  integer_constant;
      Hv                 IS_A  heat_of_vaporization_constant;
      Tliq               IS_A  reference_temperature;
      Vliq               IS_A  reference_molar_volume;
      lden               IS_A  reference_mass_density;
      T0                 IS_A  reference_temperature;
      P0                 IS_A  reference_pressure;
      H0, G0             IS_A  reference_molar_energy;
      S0                 IS_A  reference_molar_entropy;

      T0 :== 298.15{K};
      P0 :== 1.0{atm};

(* data checked by: Jennifer L Perry 1/97 *)

END component_constants;
```

```
MODEL td_component_constants(
      component_name IS_A symbol_constant;
) WHERE (
      (component_name ==  ‘hydrogen’) OR
      (component_name ==  ‘carbon_dioxide’) OR
      (component_name ==  ‘water’) OR
      (component_name ==  ‘chloroform’) OR
      (component_name ==  ‘methane’) OR
      (component_name ==  ‘methanol’) OR
      (component_name ==  ‘ethylene’) OR
      (component_name ==  ‘ethane’) OR
      (component_name ==  ‘ethanol’) OR
      (component_name ==  ‘propylene’) OR
      (component_name ==  ‘acetone’) OR
      (component_name ==  ‘propane’) OR
      (component_name ==  ‘propadiene’) OR
      (component_name ==  ‘n_propanol’) OR
      (component_name ==  ‘i_propanol’) OR
      (component_name ==  ‘ethylene_glycol’) OR
      (component_name ==  ‘n_butane’ ) OR
      (component_name ==  ‘i_butane’ ) OR
      (component_name ==  ‘n_butanol’ ) OR
      (component_name ==  ‘i_butanol’ ) OR
      (component_name ==  ‘n_pentane’ ) OR
      (component_name ==  ‘i_pentane’ ) OR
      (component_name ==  ‘benzene’ ) OR
      (component_name ==  ‘n_hexane’ ) OR
      (component_name ==  ‘toluene’ ) OR
      (component_name ==  ‘n_heptane’ ) OR
      (component_name ==  ‘styrene’ ) OR
      (component_name ==  ‘ethylbenzene’ ) OR
      (component_name ==  ‘n_octane’ ) OR
      (component_name ==  ‘diethylbenzene’ ) OR
      (component_name ==  ‘n_decane’ ) OR
      (component_name ==  ‘diphenyl’ ) OR
      (component_name ==  ‘pentanol_1’ ) OR
      (component_name ==  ‘chloro_benzene’ ) OR
      (component_name ==  ‘oxygen’ ) OR
      (component_name ==  ‘nitrogen’ );
(* note: keep this list consistent with td_component_data_set.
*     we really would prefer an external database lookup.
*)
) REFINES component_constants;

(* each component name we support should be in the SELECT below *)
      nu[subgroups]IS_A integer_constant;
      lambda[wilson_set]IS_AWilson_constant;
      del_ip[wilson_set]IS_A Wilson_energy_constant;
      groups :== UNION[[uc.group[i]] | i IN subgroups];

```

```
      Vliq :== mw/lden;
      S0 :== (H0-G0)/T0;
      H0 :== Hf;
      G0 :== Gf;

      SELECT(component_name)
(*No. 75 equilibrium_hydrogen*)
      CASE 'hydrogen':
            formula                 :== 'H2';
            subgroups         :== [];
            wilson_set        :== [];
            mw :== 2.016 {g/g_mole};
            Tb :== 20.3 {K};
            Tc :== 33.0 {K};
            Pc :== 12.9 {bar};
            Vc :== 64.3 {cm^3/g_mole};
            Zc :== 0.303;
            omega :== -0.216;
            cpvapa :== 27.14 {J/g_mole/K};
            cpvapb :== 9.274e+1 {J/g_mole/K^2};
            cpvapc :== -1.381e-5 {J/g_mole/K^3};
            cpvapd :== 7.645e-9 {J/g_mole/K^4};
            Hf :== 0.00 {J/g_mole};
            Gf :== 0.00 {J/g_mole};
            vpa :== -5.57929;
            vpb :== 2.60012;
            vpc :== -0.85506;
            vpd :== 1.70503;
            vp_correlation :== 1;
            Hv :== 216 {cal/g_mole};(*Reid, et.al 2nd ed.*)
            Tliq :== 20.0 {K};
            lden :== 0.071 {g/cm^3};

(*No. 99*)
      CASE 'carbon_dioxide':
            formula                 :== 'CO2';
            subgroups         :== [];
            wilson_set        :== [];
            mw :== 44.010 {g/g_mole};
            Tb :== 194.7 {K}; (*Reid, et.al 2nd ed.*)
            Tc :== 304.1 {K};
            Pc :== 73.8 {bar};
            Vc :== 93.9 {cm^3/g_mole};
            Zc :== 0.274;
            omega :== 0.239;
            cpvapa :== 19.80 {J/g_mole/K};
            cpvapb :== 7.344e-2 {J/g_mole/K^2};
            cpvapc :== -5.602e-5 {J/g_mole/K^3};
            cpvapd :== 1.715e-8 {J/g_mole/K^4};
            Hf :== -3.938e+5 {J/g_mole};
```

```
            Gf :== -3.946e+5 {J/g_mole};
            vpa :== -6.95626;
            vpb :== 1.19695;
            vpc :== -3.12614;
            vpd :== 2.99448;
            vp_correlation :== 1;
            Hv :== 4100 {cal/g_mole}; (*Reid, et.al 2nd ed.*)
            Tliq :== 293 {K}; (*Reid, et.al 2nd ed.*)
            lden :== 0.777; (*Reid, et.al 2nd ed.*)

(*No. 77*)
      CASE ‘water’:
            formula     :== ‘H2O’;
            subgroups:== [‘H2O’];
            wilson_set:==
            [‘CH3OH’,’(CH3)2CO’,’(CH3)2CHOH’,’C2H6O2’,’H2O’];
            mw :== 18.015 {g/g_mole};
            Tb :== 373.2 {K};
            Tc :== 647.3 {K};
            Pc :== 221.2 {bar};
            Vc :== 57.1 {cm^3/g_mole};
            Zc :== 0.235;
            omega :== 0.344;
            cpvapa :== 32.24 {J/g_mole/K};
            cpvapb :== 1.924e-3 {J/g_mole/K^2};
            cpvapc :== 1.055e-5 {J/g_mole/K^3};
            cpvapd :== -3.596e-9 {J/g_mole/K^4};
            Hf :== -2.420e+5 {J/g_mole};
            Gf :== -2.288e+5 {J/g_mole};
            vpa :== -7.76451;
            vpb :== 1.45838;
            vpc :== -2.77580;
            vpd :== -1.23303;
            vp_correlation :== 1;
            Hv :== 9717.0 {cal/g_mole}; (*Reid, et.al 2nd ed.*)
            Tliq :== 293.0 {K};
            lden :== 0.998 {g/cm^3};
            nu[‘H2O’]   :==   1;
            lambda[‘CH3OH’]:==0.94934;
            lambda[‘(CH3)2CO’]:==0.40640;
            lambda[‘(CH3)2CHOH’]:==0.78363;
            lambda[‘C2H6O2’]:==1.35289;
            lambda[‘H2O’]:==  1.0;
            del_ip[‘(CH3)2CHOH’]:==5085.76 {J/g_mole};
            del_ip[‘(CH3)2CO’]:==5997.845 {J/g_mole};
            del_ip[‘CH3OH’]:==2.0760E+002 {J/g_mole};
            del_ip[‘C2H6O2’]:==3946.12 {J/g_mole};
            del_ip[‘H2O’]:==  0.0 {J/g_mole};
(* No. 103 *)
      CASE ‘chloroform’:
```

```
            formula     :== ‘CHCl3’;
            subgroups:== [‘CHCl3’];
            wilson_set:== [‘CHCl3’,’C6H6’,’(CH3)2CO’];
            mw :== 119.378 {g/g_mole};
            Tb :== 334.3 {K};
            Tc :== 536.4 {K};
            Pc :== 53.7 {bar};
            Vc :== 238.9 {cm^3/g_mole};
            Zc :== 0.293;
            omega :== 0.218;
            cpvapa :== 24.00 {J/g_mole/K};
            cpvapb :== 1.893e-1 {J/g_mole/K^2};
            cpvapc :== -1.841e-4 {J/g_mole/K^3};
            cpvapd :== 6.657e-8 {J/g_mole/K^4};
            Hf :== -1.013e+5 {J/g_mole};
            Gf :== -6.858e+4 {J/g_mole};
            vpa :== -6.95546;
            vpb :== 1.16625;
            vpc :== -2.13970;
            vpd :== -3.44421;
            vp_correlation :== 1;
            Hv :== 7100 {cal/g_mole}; (*Reid, et.al 2nd ed.*)
            Tliq :== 293.15 {K};
            lden :== 1.489 {g/cm^3};

            nu[‘CHCl3’] :==   1;
            lambda[‘(CH3)2CO’]:==1.49288;
            lambda[‘C6H6’]:== 1.92309;
            lambda[‘CHCl3’]:==1.0;
            del_ip[‘(CH3)2CO’]:==-1359.32 {J/g_mole};
            del_ip[‘C6H6’]:== -1638.28 {J/g_mole};
            del_ip[‘CHCl3’]:==0.0 {J/g_mole};

(* No. 116 *)
      CASE ‘methane’:

            formula     :== ‘CH4’;
            subgroups:== [‘CH3’];
            wilson_set:== [];

            mw :== 16.043 {g/g_mole};
            Tb :== 111.6 {K};
            Tc :== 190.4 {K};
            Pc :== 46.0 {bar};
            Vc :== 99.0 {cm^3/g_mole};
            Zc :== 0.288;
            omega :== 0.011;
            cpvapa :== 19.25 {J/g_mole/K};
            cpvapb :== 5.213e-2 {J/g_mole/K^2};
```

```
            cpvapc :== 1.197e-5 {J/g_mole/K^3};
            cpvapd :== -1.132e-8 {J/g_mole/K^4};
            Hf :== -7.490e+4 {J/g_mole};
            Gf :== -5.087e+4 {J/g_mole};
            vpa :== -6.00435;
            vpb :== 1.18850;
            vpc :== -0.83408;
            vpd :== -1.22833;
            vp_correlation :== 1;
            Hv :== 1955 {cal/g_mole}; (*Reid, et.al 2nd ed.*)
            Tliq :== 111.7 {K};
            lden :== 0.425 {g/cm^3};

            nu[‘CH3’]    :==   1;

(* No. 117 *)
      CASE ‘methanol’:

            formula     :== ‘CH3OH’;
            subgroups:== [‘CH3’, ‘OH’];
            wilson_set :== [‘H2O’,’(CH3)2CO’,’CH3OH’];

            mw :== 32.042 {g/g_mole};
            Tb :== 337.7 {K};
            Tc :== 512.6 {K};
            Pc :== 80.9 {bar};
            Vc :== 118.0 {cm^3/g_mole};
            Zc :== 0.224;
            omega :== 0.556;
            cpvapa :== 21.15 {J/g_mole/K};
            cpvapb :== 7.092e-2 {J/g_mole/K^2};
            cpvapc :== 2.587e-5 {J/g_mole/K^3};
            cpvapd :== -2.852e-8 {J/g_mole/K^4};
            Hf :== -2.0133e+5 {J/g_mole};
            Gf :== -1.626e+5 {J/g_mole};
            vpa :== -8.54796;
            vpb :== 0.76982;
            vpc :== -3.10850;
            vpd :== 1.54481;
            vp_correlation :== 1;
            Hv :== 8426 {cal/g_mole}; (*Reid, et.al 2nd ed.*)
            Tliq :== 293 {K};
            lden :== 0.791 {g/cm^3};

            nu[‘CH3’]    :==   1;
            nu[‘OH’]     :==   1;
            lambda[‘H2O’]:==  0.43045;
            lambda[‘(CH3)2CO’]:==0.77204;
            lambda[‘CH3OH’]:==1.0;
            del_ip[‘(CH3)2CO’]:==2.6493E+002 {J/g_mole};
```

```
            del_ip['H2O']:==  1.1944E+002 {J/g_mole};
            del_ip['CH3OH']:==0.0 {J/g_mole};


(* No. 155 *)
      CASE 'ethylene':

            formula           :== 'C2H4';
            subgroups   :== ['CH2=CH'];
            wilson_set  :== [];
            mw :== 28.054 {g/g_mole};
            Tb :== 169.3 {K};
            Tc :== 282.4 {K};
            Pc :== 50.4 {bar};
            Vc :== 130.4 {cm^3/g_mole};
            Zc :== 0.280;
            omega :== 0.089;
            cpvapa :== 3.806 {J/g_mole/K};
            cpvapb :== 1.566e-1 {J/g_mole/K^2};
            cpvapc :== -8.348e-5 {J/g_mole/K^3};
            cpvapd :== 1.755e-8 {J/g_mole/K^4};
            Hf :== 5.234e+4 {J/g_mole};
            Gf :== 6.816e+4 {J/g_mole};
            vpa :== -6.32055;
            vpb :== 1.16819;
            vpc :== -1.55935;
            vpd :== -1.83552;
            vp_correlation :== 1;
            Hv :== 3237 {cal/g_mole}; (*Reid, et.al 2nd ed.*)
            Tliq :== 163{K};
            lden :== 0.577 {g/cm^3};
            nu['CH2=CH']:==   1;


(* No. 168 *)
      CASE 'ethane':

            formula           :== 'C2H6';
            subgroups   :== ['CH3'];
            wilson_set  :== [];
            mw :== 30.070 {g/g_mole};
            Tb :== 184.6 {K};
            Tc :== 305.4 {K};
            Pc :== 48.8 {bar};
            Vc :== 148.3 {cm^3/g_mole};
            Zc :== 0.285;
            omega :== 0.099;
            cpvapa :== 5.409 {J/g_mole/K};
            cpvapb :== 1.781e-1 {J/g_mole/K^2};
            cpvapc :== -6.938e-5 {J/g_mole/K^3};
```

```
          cpvapd :== 8.713e-9 {J/g_mole/K^4};
          Hf :== -8.474e+4 {J/g_mole};
          Gf :== -3.295e+4 {J/g_mole};
          vpa :== -6.34307;
          vpb :== 1.01630;
          vpc :== -1.19116;
          vpd :== -2.03539;
          vp_correlation :== 1;
          Hv :== 3515 {cal/g_mole}; (*Reid, et.al 2nd ed.*)
          Tliq :== 183.0 {K};
          lden :== 0.548 {g/cm^3};

          nu[‘CH3’]   :==   2;


(* No. 170 *)
     CASE ‘ethanol’:

          formula           :== ‘C2H5OH’;
          subgroups   :== [‘CH3’, ‘CH2’, ‘OH’];
          wilson_set  :== [];

          mw :== 46.069 {g/g_mole};
          Tb :== 351.4 {K};
          Tc :== 513.9 {K};
          Pc :== 61.4 {bar};
          Vc :== 167.1 {cm^3/g_mole};
          Zc :== 0.240;
          omega :== 0.644;
          cpvapa :== 9.014 {J/g_mole/K};
          cpvapb :== 2.141e-1 {J/g_mole/K^2};
          cpvapc :== -8.390e-5 {J/g_mole/K^3};
          cpvapd :== 1.373e-9 {J/g_mole/K^4};
          Hf :== -2.350e+5 {J/g_mole};
          Gf :== -1.684e+5 {J/g_mole};
          vpa :== -8.51838;
          vpb :== 0.34163;
          vpc :== -5.73683;
          vpd :== 8.32581;
          vp_correlation :== 1;
          Hv :== 9260 {cal/g_mole}; (*Reid, et.al 2nd ed.*)
          Tliq :== 293.15 {K};
          lden :== 0.789 {g/cm^3};

          nu[‘CH3’]   :==   1;
          nu[‘CH2’]   :==   1;
          nu[‘OH’]    :== 1;

(* No. 194 *)
     CASE ‘propylene’:
```

```
            formula           :== ‘C3H6’;
            subgroups   :== [‘CH3’, ‘CH2’, ‘CH’];
            wilson_set  :== [‘C3H4’,’C3H8’,’C3H6’];

            mw :== 42.081 {g/g_mole};
            Tb :== 225.5 {K};
            Tc :== 364.9 {K};
            Pc :== 46.0 {bar};
            Vc :== 181.0 {cm^3/g_mole};
            Zc :== 0.274;
            omega :== 0.144;
            cpvapa :== 3.710 {J/g_mole/K};
            cpvapb :== 2.345e-1 {J/g_mole/K^2};
            cpvapc :== -1.160e-4 {J/g_mole/K^3};
            cpvapd :== 2.205e-8 {J/g_mole/K^4};
            Hf :== 2.043e+4 {J/g_mole};
            Gf :== 6.276e+4 {J/g_mole};
            vpa :== -6.64231;
            vpb :== 1.21857;
            vpc :== -1.81005;
            vpd :== -2.48212;
            vp_correlation :== 1;
            Hv :== 4400 {cal/g_mole}; (*Reid, et.al 2nd ed.*)
            Tliq :== 223.0 {K};
            lden :== 0.612 {g/cm^3};

            nu[‘CH3’]   :==   1;
            nu[‘CH2’]   :==   1;
            nu[‘CH’]    :==   1;

            lambda[‘C3H4’]:== 1.150;
            lambda[‘C3H8’]:== 1.057;
            lambda[‘C3H6’]:== 1.0;

            del_ip[wilson_set] :== MAX_REAL * 1e-12{J/g_mole};

(* No. 196 *)
      CASE ‘acetone’:

            formula           :== ‘(CH3)2CO’;
            subgroups   :== [‘CH3’, ‘CH3CO’];
            wilson_set :== [‘H2O’,’CH3OH’,’CHCl3’,’C6H6’,’(CH3)2CO’];

            mw :== 58.080 {g/g_mole};
            Tb :== 329.2 {K};
            Tc :== 508.1 {K};
            Pc :== 47.0 {bar};
            Vc :== 209.0 {cm^3/g_mole};
            Zc :== 0.232;
```

```
        omega :== 0.304;
        cpvapa :== 6.301 {J/g_mole/K};
        cpvapb :== 2.606e-1 {J/g_mole/K^2};
        cpvapc :== -1.253e-4 {J/g_mole/K^3};
        cpvapd :== 2.038e-8 {J/g_mole/K^4};
        Hf :== -2.177e+5 {J/g_mole};
        Gf :== -1.532e+5 {J/g_mole};
        vpa :== -7.45514;
        vpb :== 1.20200;
        vpc :== -2.43926;
        vpd :== -3.35590;
        vp_correlation :== 1;
        Hv :== 6960 {cal/g_mole}; (*Reid, et.al 2nd ed.*)
        Tliq :== 293.15{K};
        lden :== 0.790 {g/cm^3};

        nu[‘CH3’]  :== 1;
        nu[‘CH3CO’] :==   1;

        lambda[‘H2O’]:==  0.16924;
        lambda[‘CH3OH’]:==0.65675;
        lambda[‘CHCl3’]:==1.29452;
        lambda[‘C6H6’]:== 0.75632;
        lambda[‘(CH3)2CO’]:==1.0;

        del_ip[‘CH3OH’]:==23.2321 {J/g_mole};
        del_ip[‘H2O’]:==  1674.817 {J/g_mole};
        del_ip[‘C6H6’]:== 1364.63 {J/g_mole};
        del_ip[‘CHCl3’]:==-468.831 {J/g_mole};
        del_ip[‘(CH3)2CO’]:==0.0 {J/g_mole};


(* No. 206 *)
    CASE ‘propane’:

        formula           :== ‘C3H8’;
        subgroups   :== [‘CH3’, ‘CH2’];
        wilson_set  :== [‘C3H4’,’C3H8’,’C3H6’];

        mw :== 44.094 {g/g_mole};
        Tb :== 231.1 {K};
        Tc :== 369.8 {K};
        Pc :== 42.5 {bar};
        Vc :== 203.0 {cm^3/g_mole};
        Zc :== 0.281;
        omega :== 0.153;
        cpvapa :== -4.224 {J/g_mole/K};
        cpvapb :== 3.063e-1 {J/g_mole/K^2};
        cpvapc :== -1.586e-4 {J/g_mole/K^3};
        cpvapd :== 3.215e-8 {J/g_mole/K^4};
```

```
        Hf :== -1.039e+5 {J/g_mole};
        Gf :== -2.349e+4 {J/g_mole};
        vpa :== -6.72219;
        vpb :== 1.33236;
        vpc :== -2.13868;
        vpd :== -1.38551;
        Hv :== 4487 {cal/g_mole}; (*Reid, et.al 2nd ed.*)
        Tliq :== 231.0 {K};
        lden :== 0.582 {g/cm^3};

        nu[‘CH3’]   :==   2;
        nu[‘CH2’]   :==   1;

        lambda[‘C3H4’]:== 1.439;
        lambda[‘C3H8’]:== 1.0;
        lambda[‘C3H6’]:== 1.062;
    (* we don’t have this data yet *)
    FOR w IN wilson_set CREATE
        del_ip[w] :== MAX_REAL*1e-12{J/g_mole};
    END;

(* No. 185 *)
    CASE ‘propadiene’:


        formula           :== ‘C3H4’;
        subgroups   :== [‘CH2’, ‘CH2=C’];
        wilson_set  :== [‘C3H4’,’C3H8’,’C3H6’];

        mw :== 40.065 {g/g_mole};
        Tb :== 238.7 {K};
        Tc :== 393.0 {K};
        Pc :== 54.7 {bar};
        Vc :== 162.0 {cm^3/g_mole};
        Zc :== 0.271;
        omega :== 0.313;
        cpvapa :== 9.906 {J/g_mole/K};
        cpvapb :== 1.977e-1 {J/g_mole/K^2};
        cpvapc :== -1.182e-4 {J/g_mole/K^3};
        cpvapd :== 2.782e-8 {J/g_mole/K^4};
        Hf :== 1.923e5 {J/g_mole};
        Gf :== 2.025e5 {J/g_mole};
        vpa :== -40.588;
        vpb :== 97.0407;
        vpc :== -262.325;
        vpd :== 1564.05;
        vp_correlation :== 3;
        Hv :== 4450.0 {cal/g_mole};  (*Reid, et.al 2nd ed.*)
        lden :== 0.658 {g/cm^3};
        Tliq :== 238.0 {K};
```

```
          nu[‘CH2’]    :==   2;
          nu[‘CH2=C’] :==   2;

          lambda[‘C3H4’]:== 1.0;
          lambda[‘C3H8’]:== 1.353;
          lambda[‘C3H6’]:== 1.136;

      FOR w IN wilson_set CREATE
      (* we don’t have this data yet *)
          del_ip[w] :== MAX_REAL * 1e-12{J/g_mole};
      END;

(* No. 207 l_propanol *)
      CASE ‘n_propanol’:

          formula           :== ‘C3H7OH’;
          subgroups    :== [‘CH3’, ‘CH2’, ‘OH’];
          wilson_set   :== [];

          mw :== 60.096 {g/g_mole};
          Tb :== 370.4 {K};
          Tc :== 536.7 {K};
          Pc :== 51.7 {bar};
          Vc :== 218.5 {cm^3/g_mole};
          Zc :== 0.253;
          omega :== 0.623;
          cpvapa :== 2.470 {J/g_mole/K};
          cpvapb :== 3.325e-1 {J/g_mole/K^2};
          cpvapc :== -1.855e-4 {J/g_mole/K^3};
          cpvapd :== 4.296e-8 {J/g_mole/K^4};
          Hf :== -2.566e+5 {J/g_mole};
          Gf :== -1.619e+5 {J/g_mole};
          vpa :== -8.05594;
          vpb :== 0.0425183;
          vpc :== -7.51296;
          vpd :== 6.89004;
          vp_correlation :== 1;
          Hv :== 9980 {cal/g_mole}; (*Reid, et.al 2nd ed.*)
          lden :== 0.804 {g/cm^3};
          Tliq :== 293.0 {K};

          nu[‘CH3’]    :==   1;
          nu[‘CH2’]    :==   2;
          nu[‘OH’]     :==   1;

(* No. 208 isopropyl alcohol *)
      CASE ‘i_propanol’:

          formula      :== ‘(CH3)2CHOH’;
```

```
        subgroups:== [‘CH3’, ‘CH’, ‘OH’];
        wilson_set:== [‘H2O’,’C2H6O2’,’(CH3)2CHOH’];
        mw :== 60.096 {g/g_mole};
        Tb :== 355.4 {K};
        Tc :== 508.3 {K};
        Pc :== 47.6 {bar};
        Vc :== 220.0 {cm^3/g_mole};
        Zc :== 0.248;
        omega :== 0.665;
        cpvapa :== 32.43 {J/g_mole/K};
        cpvapb :== 1.885e-1 {J/g_mole/K^2};
        cpvapc :== 6.406e-5 {J/g_mole/K^3};
        cpvapd :== -9.261e-8 {J/g_mole/K^4};
        Hf :== -2.726e+5 {J/g_mole};
        Gf :== -1.177e+5 {J/g_mole};
        vpa :== -8.16927;
        vpb :== -0.0943213;
        vpc :== -8.10040;
        vpd :== 7.85000;
        vp_correlation :== 1;
        Hv :== 9520 {cal/g_mole}; (*Reid, et.al 2nd ed.*)
        lden :== 0.786 {g/cm^3};
        Tliq :== 293.0 {K};

        nu[‘CH3’]   :==   2;
        nu[‘CH’]    :==   1;
        nu[‘OH’]    :==   1;
        lambda[‘H2O’]:==  0.03094;
        lambda[‘C2H6O2’]:==1.80740;
        lambda[‘(CH3)2CHOH’]:==1.0;
        del_ip[‘H2O’]:==  6255.66 {J/g_mole};
        del_ip[‘C2H6O2’]:==-1319.20 {J/g_mole};
        del_ip[‘(CH3)2CHOH’]:==0.0 {J/g_mole};

(* No 171 *)
    CASE ‘ethylene_glycol’:

        formula     :== ‘C2H6O2’;
        subgroups:== [‘CH2’,’OH’];
        wilson_set:== [‘H2O’,’(CH3)2CHOH’,’C2H6O2’];

        mw :== 62.069 {g/g_mole};
        Tb :== 470.5 {K};
        Tc :== 645 {K};
        Pc :== 77 {bar};
        Vc :== 55.71723{cm^3/g_mole};
        Zc :== 0.270;
        omega :== (0.9145 + 1.20)/2;
        (* omega Calculated using Perry’s, p. 3-257*)
        cpvapa :== 3.570e+1 {J/g_mole/K};
```

```
        cpvapb :== 2.483e-1 {J/g_mole/K^2};
        cpvapc :== -1.497e-4 {J/g_mole/K^3};
        cpvapd :== 3.010e-8 {J/g_mole/K^4};
        Hf :== -3.896e5 {J/g_mole};
        Gf :== -3.047e5 {J/g_mole};
        vpa :== 13.6299;
        vpb :== 6022.18;
        vpc :== -28.25;
        vpd :== 0;
        vp_correlation :== 3;
        Hv :== 12550.0 {cal/g_mole}; (*Reid, et.al 2nd ed.*)
        lden :== 1.114 {g/cm^3};
        Tliq :== 293.0 {K};

        nu[‘CH2’]   :==   2;
        nu[‘OH’]    :==   2;

        lambda[‘H2O’]:==  0.42861;
        lambda[‘(CH3)2CHOH’]:==0.23018;
        lambda[‘C2H6O2’]:==1.0;

        del_ip[‘H2O’]:==  -2661.746 {J/g_mole};
        del_ip[‘(CH3)2CHOH’]:==2167.164 {J/g_mole};
        del_ip[‘C2H6O2’]:==0.0 {J/g_mole};


(* No. 258 *)
    CASE ‘n_butane’:

        formula                 :== ‘C4H10’;
        subgroups         :== [‘CH3’, ‘CH2’];
        wilson_set        :== [];

        mw :== 58.124 {g/g_mole};
        Tb :== 272.7 {K};
        Tc :== 425.2 {K};
        Pc :== 38.0 {bar};
        Vc :== 255 {cm^3/g_mole};
        Zc :== 0.274;
        omega :== 0.199;
        cpvapa :== 9.487 {J/g_mole/K};
        cpvapb :== 3.313e-1 {J/g_mole/K^2};
        cpvapc :== -1.108e-4 {J/g_mole/K^3};
        cpvapd :== -2.822e-9 {J/g_mole/K^4};
        Hf :== -1.262e+5 {J/g_mole};
        Gf :== -1.610e+4 {J/g_mole};
        vpa :== -6.88709;
        vpb :== 1.15157;
        vpc :== -1.99873;
        vpd :== -3.13003;
```

```
            vp_correlation :== 1;
            Hv :== 5352 {cal/g_mole}; (*Reid, et.al 2nd ed.*)
            lden :== 0.579 {g/cm^3};
            Tliq :== 293.0 {K};

            nu['CH3']   :==   2;
            nu['CH2']   :==   2;

(* No. 259 isobutane*)
      CASE 'i_butane':

            formula     :== '(CH3)3CH';
            subgroups:== ['CH3', 'CH'];
            wilson_set:== [];

            mw :== 58.124 {g/g_mole};
            Tb :== 261.4 {K};
            Tc :== 408.2 {K};
            Pc :== 36.5 {bar};
            Vc :== 263 {cm^3/g_mole};
            Zc :== 0.283;
            omega :== 0.183;
            cpvapa :== -1.390 {J/g_mole/K};
            cpvapb :== 3.847e-1 {J/g_mole/K^2};
            cpvapc :== -1.846e-4 {J/g_mole/K^3};
            cpvapd :== 2.895e-8 {J/g_mole/K^4};
            Hf :== -1.346e+5 {J/g_mole};
            Gf :== -2.090e+4 {J/g_mole};
            vpa :== -6.95579;
            vpb :== 1.50090;
            vpc :== -2.52717;
            vpd :== -1.49776;
            vp_correlation :== 1;
            Hv :== 5090 {cal/g_mole}; (*Reid, et.al 2nd ed.*)
            lden :== 0.557 {g/cm^3};
            Tliq :== 293.0 {K};

            nu['CH3']   :==   3;
            nu['CH']    :==   1;


(* No. 260*)
      CASE 'n_butanol':

            formula     :== 'C4H9OH';
            subgroups:== ['CH3', 'CH2', 'OH'];
            wilson_set:== [];

            mw :== 74.123 {g/g_mole};
            Tb :== 390.9 {K};
```

```
            Tc :== 563.1 {K};
            Pc :== 44.2 {bar};
            Vc :== 274.0 {cm^3/g_mole};
            Zc :== 0.259;
            omega :== 0.593;
            cpvapa :== 3.266 {J/g_mole/K};
            cpvapb :== 4.180e-1 {J/g_mole/K^2};
            cpvapc :== -2.242e-4 {J/g_mole/K^3};
            cpvapd :== 4.685e-8 {J/g_mole/K^4};
            Hf :== -2.749e+5 {J/g_mole};
            Gf :== -1.509e+5 {J/g_mole};
            vpa :== -8.00756;
            vpb :== 0.53783;
            vpc :== -9.34240;
            vpd :== 6.68692;
            vp_correlation :== 1;
            Hv :== 10300 {cal/g_mole}; (*Reid, et.al 2nd ed.*)
            lden :== 0.810;
            Tliq :== 293 {K};

            nu['CH3']   :==   1;
            nu['CH2']   :==   3;
            nu['OH']  :== 1;


(* No. 262 isobutanol*)
      CASE 'i_butanol':

            formula     :== '(CH3)2CHCH2OH';
            subgroups:== ['CH3','CH2','CH','OH'];
            wilson_set:== [];

            mw :== 74.123 {g/g_mole};
            Tb :== 381.0 {K};
            Tc :== 547.8 {K};
            Pc :== 43.0 {bar};
            Vc :== 273 {cm^3/g_mole};
            Zc :== 0.257;
            omega :== 0.592;
            cpvapa :== -7.708 {J/g_mole/K};
            cpvapb :== 4.689e-1 {J/g_mole/K^2};
            cpvapc :== -2.884e-4 {J/g_mole/K^3};
            cpvapd :== 7.231e-8 {J/g_mole/K^4};
            Hf :== -2.834e+5 {J/g_mole};
            Gf :== -1.674e+5 {J/g_mole};
            vpa :== 10.2510;
            vpb :== 2874.73;
            vpc :== -100.3;
            vpd :== 0;
            vp_correlation :== 3;
```

```
            Hv :== 10050 {cal/g_mole}; (*Reid, et.al 2nd ed.*)
            lden :== 0.802 {g/cm^3};
            Tliq :== 293.0 {K};

            nu['CH3']   :==   2;
            nu['CH2']   :==   1;
            nu['CH']  :== 1;
            nu['OH']  :== 1;


(* No. 311 *)
      CASE 'n_pentane':


            formula     :== 'C5H12';
            subgroups:== ['CH3', 'CH2'];
            wilson_set:== [];

            mw :== 72.151 {g/g_mole};
            Tb :== 309.2 {K};
            Tc :== 469.7 {K};
            Pc :== 33.7 {bar};
            Vc :== 304 {cm^3/g_mole};
            Zc :== 0.263;
            omega :== 0.251;
            cpvapa :== -3.626 {J/g_mole/K};
            cpvapb :== 4.873e-1 {J/g_mole/K^2};
            cpvapc :== -2.580e-4 {J/g_mole/K^3};
            cpvapd :== 5.305e-8 {J/g_mole/K^4};
            Hf :== -1.465e+5 {J/g_mole};
            Gf :== -8.370e+3 {J/g_mole};
            vpa :== -7.28936;
            vpb :== 1.53679;
            vpc :== -3.08367;
            vpd :== -1.02456;
            vp_correlation :== 1;
            Hv :== 6160 {cal/g_mole}; (*Reid, et.al 2nd ed.*)
            lden :== 0.626 {g/cm^3};
            Tliq :== 293.0 {K};

            nu['CH3']   :==   2;
            nu['CH2']   :==   3;


(* No. 312 2-methyl butane*)
      CASE 'i_pentane':

            formula     :== 'C2H5CH(CH3)2';
            subgroups:== ['CH3', 'CH2', 'CH'];
            wilson_set:== [];
```

```
        mw :== 72.151 {g/g_mole};
        Tb :== 301.0 {K};
        Tc :== 460.4 {K};
        Pc :== 33.9 {bar};
        Vc :== 306 {cm^3/g_mole};
        Zc :== 0.271;
        omega :== 0.227;
        cpvapa :== -9.525 {J/g_mole/K};
        cpvapb :== 5.066e-1 {J/g_mole/K^2};
        cpvapc :== -2.729e-4 {J/g_mole/K^3};
        cpvapd :== 5.723e-8 {J/g_mole/K^4};
        Hf :== -1.546e+5 {J/g_mole};
        Gf :== -1.482e+4 {J/g_mole};
        vpa :== -7.12727;
        vpb :== 1.38996;
        vpc :== -2.54302;
        vpd :== -2.45657;
        vp_correlation :== 1;
        Hv :== 5900 {cal/g_mole}; (*Reid, et.al 2nd ed.*)
        lden :== 0.620 {g/cm^3};
        Tliq :== 293.0{K};

        nu[‘CH3’]   :==   3;
        nu[‘CH2’]   :==   1;
        nu[‘CH’]    :==   1;


(* No. 343 *)
    CASE ‘benzene’:

        formula     :== ‘C6H6’;
        subgroups:== [‘ACH’];
        wilson_set:== [‘CHCl3’,’C6H6’,’(CH3)2CO’];

        mw :== 78.114 {g/g_mole};
        Tb :== 353.2 {K};
        Tc :== 562.2 {K};
        Pc :== 48.9 {bar};
        Vc :== 259 {cm^3/g_mole};
        Zc :== 0.271;
        omega :== 0.212;
        cpvapa :== -3.392e+1 {J/g_mole/K};
        cpvapb :== 4.739e-1 {J/g_mole/K^2};
        cpvapc :== -3.017e-4 {J/g_mole/K^3};
        cpvapd :== 7.130e-8 {J/g_mole/K^4};
        Hf :== 8.298e+4 {J/g_mole};
        Gf :== 1.297e+5 {J/g_mole};
        vpa :== -6.98273;
        vpb :== 1.33213;
```

```
            vpc :== -2.62863;
            vpd :== -3.33399;
            vp_correlation :== 1;
            Hv :== 7352 {cal/g_mole}; (*Reid, et.al 2nd ed.*)
            lden :== 0.885 {g/cm^3};
            Tliq :== 289.0{K};

            nu['ACH']   :==    6;
            lambda['CHCl3']:== 0.52431;
            lambda['(CH3)2CO']:== 0.87629;
            lambda['C6H6']  :== 1.0;
            del_ip['CHCl3'] :== 1614.0 {J/g_mole};
            del_ip['(CH3)2CO']:== -156.458 {J/g_mole};
            del_ip['C6H6']  :== 0.0 {J/g_mole};


(* No. 379 *)
      CASE 'n_hexane':

            formula           :== 'C6H14';
            subgroups   :== ['CH3', 'CH2'];
            wilson_set  :== [];


            mw :== 86.178 {g/g_mole};
            Tb :== 341.9 {K};
            Tc :== 507.5 {K};
            Pc :== 30.1 {bar};
            Vc :== 370 {cm^3/g_mole};
            Zc :== 0.264;
            omega :== 0.299;
            cpvapa :== -4.413 {J/g_mole/K};
            cpvapb :== 5.820e-1 {J/g_mole/K^2};
            cpvapc :== -3.119e-4 {J/g_mole/K^3};
            cpvapd :== 6.494e-8 {J/g_mole/K^4};
            Hf :== -1.673e+5 {J/g_mole};
            Gf :== -1.670e+2 {J/g_mole};
            vpa :== -7.46765;
            vpb :== 1.44211;
            vpc :== -3.28222;
            vpd :== -2.50941;
            vp_correlation :== 1;
            Hv :== 6896 {cal/g_mole};  (*Reid, et.al 2nd ed.*)
            lden :== 0.659 {g/cm^3};
            Tliq :== 293.0{K};

            nu['CH3']   :==   2;
            nu['CH2']   :==   4;
```

```
(* No. 400*)
     CASE 'toluene':

           formula           :== 'C6H5CH3';
           subgroups   :== ['ACH', 'ACCH3'];
           wilson_set  :== [];

           mw :== 92.141 {g/g_mole};
           Tb :== 383.8 {K};
           Tc :== 591.8 {K};
           Pc :== 41.0 {bar};
           Vc :== 316 {cm^3/g_mole};
           Zc :== 0.263;
           omega :== 0.263;
           cpvapa :== -2.435e+1 {J/g_mole/K};
           cpvapb :== 5.125e-1 {J/g_mole/K^2};
           cpvapc :== -2.765e-4 {J/g_mole/K^3};
           cpvapd :== 4.911e-8 {J/g_mole/K^4};
           Hf :== 5.003e+4 {J/g_mole};
           Gf :== 1.221e+5 {J/g_mole};
           vpa :== -7.28607;
           vpb :== 1.38091;
           vpc :== -2.83433;
           vpd :== -2.79168;
           vp_correlation :== 1;
           Hv :== 7930 {cal/g_mole};  (*Reid, et.al 2nd ed.*)
           Tliq :== 293.0{K};
           lden :== 0.867 {g/cm^3};

           nu['ACH']   :==   5;
           nu['ACCH3'] :==   1;


(* No. 429 *)
     CASE 'n_heptane':

           formula           :== 'C7H16';
           subgroups   :== ['CH3', 'CH2'];
           wilson_set  :== [];

           mw :== 100.205 {g/g_mole};
           Tb :== 371.6 {K};
           Tc :== 540.3 {K};
           Pc :== 27.4 {bar};
           Vc :== 432 {cm^3/g_mole};
           Zc :== 0.263;
           omega :== 0.349;
           cpvapa :== -5.146 {J/g_mole/K};
           cpvapb :== 6.762e-1 {J/g_mole/K^2};
           cpvapc :== -3.651e-4 {J/g_mole/K^3};
```

```
            cpvapd :== 7.658e-8 {J/g_mole/K^4};
            Hf :== -1.879e+5 {J/g_mole};
            Gf :== 8.000e+3 {J/g_mole};
            vpa :== -7.67468;
            vpb :== 1.37068;
            vpc :== -3.53620;
            vpd :== -3.20243;
            vp_correlation :== 1;
            Hv :== 7576 {cal/g_mole}; (*Reid, et.al 2nd ed.*)
            lden :== 0.684 {g/cm^3};
            Tliq :== 293.0{K};

            nu['CH3']   :==   2;
            nu['CH2']   :==   5;


(* No. 440 *)
      CASE 'styrene':

            formula           :== 'C6H5C2H3';
            subgroups   :== ['ACH', 'AC', 'CH2=CH'];
            wilson_set  :== [];

            mw :== 104.152 {g/g_mole};
            Tb :== 418.3 {K};
            Tc :== 647 {K};
            Pc :== 39.9 {bar};
            Vc :== 350.55 {cm^3/g_mole};(*Not sure about Vc Zc*)
            Zc :== 0.260;
            omega :== 0.257;
            cpvapa :== -2.825e+1 {J/g_mole/K};
            cpvapb :== 6.159e-1 {J/g_mole/K^2};
            cpvapc :== -4.023e-4 {J/g_mole/K^3};
            cpvapd :== 9.935e-8 {J/g_mole/K^4};
            Hf :== 1.475e+5 {J/g_mole};
            Gf :== 2.139e+5 {J/g_mole};
            vpa :== -7.15981;
            vpb :== 1.78861;
            vpc :== -5.10359;
            vpd :== 1.63749;
            vp_correlation :== 1;
            Hv :== 8800 {cal/g_mole}; (*Reid, et.al 2nd ed.*)
            lden :== 0.906 {g/cm^3};
            Tliq :== 293.15{K};

            nu['ACH']   :==   5;
            nu['AC']    :==   1;
            nu['CH2=CH']:==   1;
```

```
(* No. 447 *)
      CASE 'ethylbenzene':

            formula           :== 'C6H5C2H5';
            subgroups   :== ['ACH', 'ACCH2', 'CH3'];
            wilson_set  :== [];

            mw :== 106.168 {g/g_mole};
            Tb :== 409.3 {K};
            Tc :== 617.2 {K};
            Pc :== 36.0 {bar};
            Vc :== 374 {cm^3/g_mole};
            Zc :== 0.262;
            omega :== 0.302;
            cpvapa :== -4.310e1 {J/g_mole/K};
            cpvapb :== 7.072e-1 {J/g_mole/K^2};
            cpvapc :== -4.811e-4 {J/g_mole/K^3};
            cpvapd :== 1.301e-7 {J/g_mole/K^4};
            Hf :== 2.981e4 {J/g_mole};
            Gf :== 1.307e5 {J/g_mole};
            vpa :== -7.48645;
            vpb :== 1.45488;
            vpc :== -3.37538;
            vpd :== -2.23048;
            vp_correlation :== 1;
            Hv :== 8500 {cal/g_mole}; (*Reid, et.al 2nd ed.*)
            lden :== 0.867 {g/cm^3};
            Tliq :== 293{K};

            nu['ACH']   :==   5;
            nu['ACCH2'] :==   1;
            nu['CH3']   :==   1;


(* No. 483 *)
      CASE 'n_octane':

            formula           :== 'C8H18';
            subgroups   :== ['CH3', 'CH2'];
            wilson_set  :== [];

            mw :== 114.232 {g/g_mole};
            Tb :== 398.8 {K};
            Tc :== 568.8 {K};
            Pc :== 24.9 {bar};
            Vc :== 492.0 {cm^3/g_mole};
            Zc :== 0.259;
            omega :== 0.398;
            cpvapa :== -6.096 {J/g_mole/K};
            cpvapb :== 7.712e-1 {J/g_mole/K^2};
```

```
            cpvapc :== -4.195e-4 {J/g_mole/K^3};
            cpvapd :== 8.855e-8 {J/g_mole/K^4};
            Hf :== -2.086e5 {J/g_mole};
            Gf :== 1.640e4 {J/g_mole};
            vpa :== -7.91211;
            vpb :== 1.38007;
            vpc :== -3.80435;
            vpd :== -4.50132;
            vp_correlation :== 1;
            Hv :== 8225 {cal/g_mole}; (*Reid, et.al 2nd ed.*)
            lden :== 0.703 {g/cm^3};
            Tliq :== 293.0{K};

            nu['CH3']   :==   2;
            nu['CH2']   :==   6;


(* No. 552 *)
      CASE 'diethylbenzene':

            formula           :== 'C6H4(C2H5)2';
            subgroups   :== ['ACH', 'ACCH2', 'CH3'];
            wilson_set  :== [];

            mw :== 134.222 {g/g_mole};
            Tb :== 456.9 {K};
            Tc :== 657.9 {K};
            Pc :== 28.0 {bar};
            Vc :== 480.0 {cm^3/g_mole};(*Not sure about Vc Zc*)
            Zc :== 0.25;
            omega :== 0.404;
            cpvapa :== -3.742e+1 {J/g_mole/K};
            cpvapb :== 8.671e-1 {J/g_mole/K^2};
            cpvapc :== -5.560e-4 {J/g_mole/K^3};
            cpvapd :== 1.411e-7 {J/g_mole/K^4};
            Hf :== -2.227e+4 {J/g_mole};
            Gf :== 1.380e+5 {J/g_mole};
            vpa :== -8.11413;
            vpb :== 1.77697;
            vpc :== -4.43960;
            vpd :== -1.47477;
            vp_correlation :== 1;
            Hv :== 9410 {cal/g_mole}; (*Reid, et.al 2nd ed.*)
            lden :== 0.862 {g/cm^3};
            Tliq :== 293 {K};

            nu['ACH']   :==   4;
            nu['ACCH2'] :==   2;
            nu['CH3']   :==   2;
```

```
(* No. 567 *)
      CASE ‘n_decane’:

            formula            :== ‘C10H22’;
            subgroups   :== [‘CH3’, ‘CH2’];
            wilson_set  :== [];

            mw :== 142.286 {g/g_mole};
            Tb :== 447.3 {K};
            Tc :== 617.7 {K};
            Pc :== 21.2 {bar};
            Vc :== 603.0 {cm^3/g_mole};
            Zc :== 0.249;
            omega :== 0.489;
            cpvapa :== -7.913 {J/g_mole/K};
            cpvapb :== 9.609e-1 {J/g_mole/K^2};
            cpvapc :== -5.288e-4 {J/g_mole/K^3};
            cpvapd :== 1.131e-7 {J/g_mole/K^4};
            Hf :== -2.498e5 {J/g_mole};
            Gf :== 3.324e4 {J/g_mole};
            vpa :== -8.56523;
            vpb :== 1.97756;
            vpc :== -5.81971;
            vpd :== -0.29982;
            vp_correlation :== 1;
            Hv :== 9388 {cal/g_mole}; (*Reid, et.al 2nd ed.*)
            lden :== 0.730 {g/cm^3};
            Tliq :== 293.0 {K};

            nu[‘CH3’]   :==   2;
            nu[‘CH2’]   :==   8;


(* No. 579 *)
      CASE ‘diphenyl’:

            formula            :== ‘(C6H5)2’;
            subgroups   :== [‘ACH’, ‘AC’];
            wilson_set  :== [];

            mw :== 154.212 {g/g_mole};
            Tb :== 529.3 {K};
            Tc :== 789.0 {K};
            Pc :== 38.5 {bar};
            Vc :== 502.0 {cm^3/g_mole};
            Zc :== 0.295;
            omega :== 0.372;
            cpvapa :== -9.707e+1 {J/g_mole/K};
            cpvapb :== 1.106 {J/g_mole/K^2};
```

```
            cpvapc :== -8.855e-4 {J/g_mole/K^3};
            cpvapd :== 2.790e-7 {J/g_mole/K^4};
            Hf :== 1.822e+5 {J/g_mole};
            Gf :== 2.803e+5 {J/g_mole};
            vpa :== -7.6740;
            vpb :== 1.23008;
            vpc :== -3.67908;
            vpd :== -2.29172;
            vp_correlation :== 1;
            Hv :== 10900 {cal/g_mole}; (*Reid, et.al 2nd ed.*)
            lden :== 0.990 {g/cm^3};
            Tliq :== 347 {K};
            nu['ACH']   :==   10;
            nu['AC']    :==   2;


(* No. 314 1_pentanol*)
      CASE 'pentanol_1':

            formula           :== 'C5H12O';
            subgroups   :== ['CH3', 'CH2', 'OH'];
            wilson_set  :== [];

            mw :== 88.150 {g/g_mole};
            Tb :== 411.1 {K};
            Tc :== 588.2 {K};
            Pc :== 39.1 {bar};
            Vc :== 326.0 {cm^3/g_mole};
            Zc :== 0.260;
            omega :== 0.579;
            cpvapa :== 3.869 {J/g_mole/K};
            cpvapb :== 0.5045 {J/g_mole/K^2};
            cpvapc :== -2.639e-04 {J/g_mole/K^3};
            cpvapd :== 5.120e-08 {J/g_mole/K^4};
            Hf :== -2.989e05 {J/g_mole};
            Gf :== -1.461e05 {J/g_mole};
            vpa :== -8.97725;
            vpb :== 2.99791;
            vpc :== -12.9596;
            vpd :== 8.84205;
            Hv :== 10600.0 {cal/g_mole}; (*Reid, et.al 2nd ed.*)
            lden :== 0.815;
            Tliq :== 293.0 {K};

            nu['CH2']   :==   4;
            nu['CH3']   :==   1;
            nu['OH']    :==   1;


(* No. 340*)
```

```
      CASE ‘chloro_benzene’:

            formula           :== ‘C6H5Cl’;
            subgroups   :== [‘ACH’, ‘AC’]; (* bug! *)
            wilson_set  :== [];

            mw :== 112.559 {g/g_mole};
            Tb :== 404.9 {K};
            Tc :== 632.4 {K};
            Pc :== 45.2 {bar};
            Vc :== 308.0 {cm^3/g_mole};
            Zc :== 0.265;
            omega :== 0.249;
            cpvapa :== -33.89 {J/g_mole/K};
            cpvapb :== 0.5631 {J/g_mole/K^2};
            cpvapc :== -4.522e-04 {J/g_mole/K^3};
            cpvapd :== 1.426e-07 {J/g_mole/K^4};
            Hf :== 5.187e04 {J/g_mole};
            Gf :== 9.923e04 {J/g_mole};
            vpa :== -7.5870;
            vpb :== 2.26551;
            vpc :== -4.09418;
            vpd :== 0.17038;
            Hv :== 8735.0 {cal/g_mole}; (*Reid, et.al 2nd ed.*)
            lden :== 1.106 {g/cm^3};
            Tliq :== 293.0 {K};

            nu[‘ACH’]   :==   5;
            nu[‘AC’]    :==   1;


(* added from art *)
      CASE ‘oxygen’:

            formula           :== ‘O2’;
            subgroups   :== [];
            wilson_set  :== [];

            mw :== 31.999 {g/g_mole};
            Tb :== 90.2 {K};
            Tc :== 154.6 {K};
            Pc :== 50.4 {bar};
            Vc :== 73.4 {cm^3/g_mole};
            Zc :== 0.288;
            omega :== 0.025;
            cpvapa :== 2.811e1 {cal/g_mole/K};
            cpvapb :== -3.680e-6 {cal/g_mole/K^2};
            cpvapc :== 1.746e-5 {cal/g_mole/K^3};
            cpvapd :== -1.065e-8 {cal/g_mole/K^4};
            Hf :== 0.00 {J/g_mole};
```

```
            Gf :== 0.00 {J/g_mole};
            vpa :== -6.28275;
            vpb :== 1.73619;
            vpc :== -1.81349;
            vpd :== -2.53645e-2;
            Hv :== 1630 {cal/g_mole};
            Tliq :== 90 {K};
            lden:== 1.149 {g/cm^3};


      CASE 'nitrogen':

            formula           :== 'N2';
            subgroups   :== [];
            wilson_set  :== [];

            mw :== 28.013 {g/g_mole};
            Tb :== 77.4 {K};
            Tc :== 126.2 {K};
            Pc :== 33.9 {bar};
            Vc :== 89.8 {cm^3/g_mole};
            Zc :== 0.290;
            omega :== 0.039;
            cpvapa :== 3.115e1 {cal/g_mole/K};
            cpvapb :== -1.357e-2 {cal/g_mole/K^2};
            cpvapc :== 2.680e-5 {cal/g_mole/K^3};
            cpvapd :== -1.168e-8 {cal/g_mole/K^4};
            Hf :== 0.00 {J/g_mole};
            Gf :== 0.00 {J/g_mole};
            vpa :== -6.09676;
            vpb :== 1.13670;
            vpc :== -1.04072;
            vpd :== -1.93306;
            Hv :== 1333 {cal/g_mole};
            Tliq :== 78 {K};
            lden :== 0.804 {g/cm^3};

      END; (* select component_name*)

END td_component_constants;

MODEL td_component_data_set(
      components IS_A set OF symbol_constant;
      reference IS_A symbol_constant;
) WHERE (
      CARD[INTERSECTION[[reference],components]] == 1;
      FOR i IN components CREATE
            (i ==  'hydrogen' ) OR
            (i ==  'carbon_dioxide' ) OR
            (i ==  'water' ) OR
```

```
            (i ==  ‘chloroform’ ) OR
            (i ==  ‘methane’ ) OR
            (i ==  ‘methanol’ ) OR
            (i ==  ‘ethylene’ ) OR
            (i ==  ‘ethane’ ) OR
            (i ==  ‘ethanol’ ) OR
            (i ==  ‘propylene’ ) OR
            (i ==  ‘acetone’ ) OR
            (i ==  ‘propane’ ) OR
            (i ==  ‘propadiene’ ) OR
            (i ==  ‘n_propanol’ ) OR
            (i ==  ‘i_propanol’ ) OR
            (i ==  ‘ethylene_glycol’ ) OR
            (i ==  ‘n_butane’ ) OR
            (i ==  ‘i_butane’ ) OR
            (i ==  ‘n_butanol’ ) OR
            (i ==  ‘i_butanol’ ) OR
            (i ==  ‘n_pentane’ ) OR
            (i ==  ‘i_pentane’ ) OR
            (i ==  ‘benzene’ ) OR
            (i ==  ‘n_hexane’ ) OR
            (i ==  ‘toluene’ ) OR
            (i ==  ‘n_heptane’ ) OR
            (i ==  ‘styrene’ ) OR
            (i ==  ‘ethylbenzene’ ) OR
            (i ==  ‘n_octane’ ) OR
            (i ==  ‘diethylbenzene’ ) OR
            (i ==  ‘n_decane’ ) OR
            (i ==  ‘diphenyl’ ) OR
            (i ==  ‘pentanol_1’ ) OR
            (i ==  ‘chloro_benzene’ ) OR
            (i ==  ‘oxygen’ ) OR
            (i ==  ‘nitrogen’ );
(*note: keep this list consistent with td_component_constants CASES
*)
      END;
);
      FOR i IN components CREATE
            data[i] IS_A td_component_constants(i);
      END;
      other IS_A set OF symbol_constant;
      other :== [components - [reference]];
END td_component_data_set;

MODEL test_components;
      ds IS_A td_component_data_set([‘methanol’,’water’],’water’);
END test_components;
```

A.2.4 PROPERTY_OPTIONS

This library is a subject of controversy in the ASCEND research group. This is not necessarily the best style for handling the distribution of consistent property option information to all the units in a flowsheet.

Appendix A Code 7

```
(* property_options.lib *)
(********************* Single phase option sets ********************)

MODEL single_phase_options(
      ds WILL_BE td_component_data_set;
        phase IS_A symbol_constant;
        component_thermo_correlation IS_A symbol_constant;
        mixture_thermo_correlation IS_A symbol_constant;
);
NOTES
'usage' SELF {
This is a template. Never use this type in the body of a MODEL.
Instead use a final refinement like
UNIFAC_Rackett_liquid_options
Wilson_Rackett_liquid_options
Pitzer_vapor_options
or material_only_options.
}
END NOTES;
END single_phase_options;

MODEL material_only_options (
      ds WILL_BE td_component_data_set;
) REFINES single_phase_options(
      phase :== 'material';
      component_thermo_correlation :== 'ideal';
      mixture_thermo_correlation :== 'ideal';
);
END material_only_options;

MODEL liquid_phase_options (
      ds WILL_BE td_component_data_set;
      component_thermo_correlation IS_A symbol_constant;
      mixture_thermo_correlation IS_A symbol_constant;
) WHERE (
      component_thermo_correlation == 'Rackett';
      (mixture_thermo_correlation == 'UNIFAC') OR
      (mixture_thermo_correlation == 'Wilson');
) REFINES single_phase_options (
      phase :== 'liquid';
);
NOTES
```

```
'usage' SELF {
This is a template. Never use this type in the body of a MODEL.
Instead use a final refinement like
UNIFAC_Rackett_liquid_options or Wilson_Rackett_liquid_options
}
END NOTES;
END liquid_phase_options;


MODEL vapor_phase_options (
      ds WILL_BE td_component_data_set;
      component_thermo_correlation IS_A symbol_constant;
      mixture_thermo_correlation IS_A symbol_constant;
) WHERE (
      component_thermo_correlation == 'Pitzer';
      mixture_thermo_correlation == 'Pitzer';
) REFINES single_phase_options(
      phase :== 'vapor';
);
NOTES
'usage' SELF {
This is a template. Avoid using this type in the body of a MODEL.
Instead use a final refinement like Pitzer_vapor_options.
}
END NOTES;
END vapor_phase_options;

MODEL UNIFAC_Rackett_liquid_options (
      ds WILL_BE td_component_data_set;
) WHERE (
      component_thermo_correlation == 'Rackett';
      (mixture_thermo_correlation == 'UNIFAC') OR
      (mixture_thermo_correlation == 'Wilson');
) REFINES liquid_phase_options(
      component_thermo_correlation :== 'Rackett';
      mixture_thermo_correlation :== 'UNIFAC';
);
END UNIFAC_Rackett_liquid_options;

MODEL Wilson_Rackett_liquid_options (
      ds WILL_BE td_component_data_set;
) WHERE (
      component_thermo_correlation == 'Rackett';
      (mixture_thermo_correlation == 'UNIFAC') OR
      (mixture_thermo_correlation == 'Wilson');
) REFINES liquid_phase_options(
      component_thermo_correlation :== 'Rackett';
      mixture_thermo_correlation :== 'Wilson';
);
END Wilson_Rackett_liquid_options;
```

```

MODEL Pitzer_vapor_options (
      ds WILL_BE td_component_data_set;
) WHERE (
      component_thermo_correlation == ‘Pitzer’;
      mixture_thermo_correlation == ‘Pitzer’;
) REFINES vapor_phase_options(
      component_thermo_correlation :== ‘Pitzer’;
      mixture_thermo_correlation :== ‘Pitzer’;
);

END Pitzer_vapor_options;

(******************** Multiphase option sets ********************)
MODEL generic_mixture_options(
      phases IS_A set OF symbol_constant;
      phase[phases] WILL_BE single_phase_options;
      reference_phase IS_A symbol_constant;
);
      n_phases IS_A integer_constant;
      n_phases :== CARD[phases];
      other_phases IS_A set OF symbol_constant;
      other_phases :== phases - [reference_phase];
END generic_mixture_options;

MODEL any_two_phase_options(
      phase1opt WILL_BE single_phase_options;
      phase2opt WILL_BE single_phase_options;
);
      phases IS_A set OF symbol_constant;
      phases :== UNION[[phase1opt.phase],[phase2opt.phase]];
END any_two_phase_options;

MODEL vle_options(
      phase1opt WILL_BE liquid_phase_options;
      phase2opt WILL_BE vapor_phase_options;
) WHERE(
      phase1opt.ds, phase2opt.ds WILL_BE_THE_SAME;
) REFINES any_two_phase_options;

      phase[pnames] ALIASES (phase1opt,phase2opt)
      WHERE pnames IS_A set OF symbol_constant
      WITH_VALUE (‘liquid’,’vapor’);
      pnames, phases ARE_THE_SAME;

      ds ALIASES phase1opt.ds;

      reference_phase IS_A symbol_constant;
      reference_phase :== ‘liquid’;
END vle_options;
```

```
MODEL Pitzer_Rackett_Wilson_options(
      phase1opt WILL_BE Wilson_Rackett_liquid_options;
      phase2opt WILL_BE Pitzer_vapor_options;
) WHERE(
      phase1opt.ds, phase2opt.ds WILL_BE_THE_SAME;
) REFINES vle_options;
END Pitzer_Rackett_Wilson_options;

MODEL Pitzer_Rackett_UNIFAC_options(
      phase1opt WILL_BE UNIFAC_Rackett_liquid_options;
      phase2opt WILL_BE Pitzer_vapor_options;
) WHERE(
      phase1opt.ds, phase2opt.ds WILL_BE_THE_SAME;
) REFINES vle_options;
END Pitzer_Rackett_UNIFAC_options;
```

A.2.5 NEW_THERMO

This library is derived from the ASCEND IIIc H_G_thermodynamics.lib. H_G_thermodynamics.lib is infinitely flexible, but very difficult to reuse. This library is somewhat less flexible but far more usable.

Appendix A Code 8(* newthermo.lib *)

```
MODEL thermodynamic_properties(
      P WILL_BE pressure;
      T WILL_BE temperature;
);

      V     IS_A molar_volume;
      H     IS_A molar_energy;
      G     IS_A molar_energy;
      scale IS_A scaling_constant;

  METHODS
      METHOD clear;
      T.fixed := FALSE;
      P.fixed := FALSE;
      V.fixed := FALSE;
      H.fixed := FALSE;
      G.fixed := FALSE;
      END clear;
      METHOD specify;
      T.fixed := TRUE;
      P.fixed := TRUE;
      V.fixed := TRUE;
      H.fixed := TRUE;
      G.fixed := TRUE;
      END specify;
      METHOD reset;
      RUN clear;
      RUN specify;
      END reset;
      METHOD scale;
      T.nominal := T;
      P.nominal := P;
      V.nominal := V;
      H.nominal := sqrt(sqr(H));
      G.nominal := sqrt(sqr(G));
      T.lower_bound := 1.0e-8{K};
      P.lower_bound := 1.0e-8{atm};
      V.lower_bound := 1.0e-8{liter/g_mole};
      H.lower_bound := H - scale*H.nominal;
      G.lower_bound := G - scale*G.nominal;
      T.upper_bound := T + scale*T.nominal;
```

```
      P.upper_bound := P + scale*P.nominal;
      V.upper_bound := V + scale*V.nominal;
      H.upper_bound := H + scale*H.nominal;
      G.upper_bound := G + scale*G.nominal;
      END scale;

END thermodynamic_properties;

MODEL pure_component(
      P WILL_BE pressure;
      T WILL_BE temperature;
      data WILL_BE td_component_constants;
      phase IS_A symbol_constant;
      correlation WILL_BE symbol_constant;
) REFINES thermodynamic_properties();
(* no body *)
END pure_component;


MODEL partial_component;

      V     IS_A molar_volume;
      H     IS_A molar_energy;
      G     IS_A molar_energy;
      scale                  IS_A scaling_constant;

  METHODS
      METHOD clear;
      V.fixed := FALSE;
      H.fixed := FALSE;
      G.fixed := FALSE;
      END clear;
      METHOD specify;
      V.fixed := TRUE;
      H.fixed := TRUE;
      G.fixed := TRUE;
      END specify;
      METHOD reset;
      RUN clear;
      RUN specify;
      END reset;
      METHOD scale;
      V.nominal := V;
      H.nominal := sqrt(sqr(H));
      G.nominal := sqrt(sqr(G));
      V.lower_bound := 1.0e-8{liter/g_mole};
      H.lower_bound := H - scale*H.nominal;
      G.lower_bound := G - scale*G.nominal;
      V.upper_bound := V + scale*V.nominal;
      H.upper_bound := H + scale*H.nominal;
```

```
      G.upper_bound := G + scale*G.nominal;
      END scale;

END partial_component;

(* mixture gone away *)

MODEL homogeneous_mixture(
      P WILL_BE pressure;
      T WILL_BE temperature;
      options WILL_BE single_phase_options;
)REFINES thermodynamic_properties();

      y[options.ds.components]   IS_A mole_fraction;
      mixture_y_def: SUM[y[i] | i IN options.ds.components] = 1.0;
      phi[options.phase]  IS_A constant;
      phi[options.phase] :== 1;

  METHODS
      METHOD clear;
      y[options.ds.components].fixed := FALSE;
      END clear;
      METHOD specify;
      y[options.ds.components-[options.ds.reference]].fixed := TRUE;
      END specify;
      METHOD reset;
      RUN clear;
      RUN specify;
      END reset;
      METHOD scale;
      FOR i IN option.ds.components DO
            y[i].nominal := y[i];
      END;
      END scale;

END homogeneous_mixture;

MODEL two_phase_mixture(
      P WILL_BE pressure;
      T WILL_BE temperature;
      light WILL_BE homogeneous_mixture;
      heavy WILL_BE homogeneous_mixture;
) WHERE (
      P, heavy.P, light.P WILL_BE_THE_SAME;
      T, heavy.T, light.T WILL_BE_THE_SAME;
      light.options.phase != heavy.options.phase;
      light.options.ds.reference == heavy.options.ds.reference;
      (* constrain the reference component to be a mobile species.*)
      INTERSECTION[light.options.ds.components,
                  heavy.options.ds.components,
```

```
                  [light.options.ds.reference]
      ] == [light.options.ds.reference];
) REFINES thermodynamic_properties;
      (* this MODEL could have species which are not able to
       * exist in one of the phases.
       *)
      components “all the components found in at least one phase”
      IS_A set OF symbol_constant;
      components :== [light.options.ds.components,
                   heavy.options.ds.components];

      y[components] “the mole fraction of each species in mixture”
      IS_A fraction;
      mixture_y_def: SUM[y[i] | i IN components] = 1.0;

      mobile “all the components moving between phases”
      IS_A set OF symbol_constant;
      mobile :==
      INTERSECTION[light.options.ds.components,
                  heavy.options.ds.components];

      number_stuck “the number of nondistributing components”
      IS_A integer_constant;
      number_stuck :== CARD[components - mobile];

      phases “the names of the phases” IS_A set OF symbol_constant;
      phases :== [light.options.phase,heavy.options.phase];
      phi[light.options.phase,heavy.options.phase]
      “fraction of total moles in each phase” IS_A fraction;

      reference IS_A symbol_constant;
      reference :== heavy.options.phase;


      SELECT (number_stuck)
      CASE 0: (* this is the standard case *)
            alpha[components]IS_A partition_coefficient;
            ave_alpha   IS_A partition_coefficient;
            FOR i IN components CREATE
            two_phase_mixture_alpha[i]:
                  ave_alpha * light.y[i] = alpha[i] * heavy.y[i];
            two_phase_mixture_phi[i]:
                  y[i] = phi[light.options.phase] * light.y[i] +
                        phi[heavy.options.phase] * heavy.y[i];
            END;
      OTHERWISE:
      (* this cases needs to be rewritten to allow for species which
       * are stuck in one phase or the other.
       *)
      END; (* select number_stuck *)
```

```
METHODS
METHOD clear;
      RUN light.clear;
      RUN heavy.clear;
      V.fixed := FALSE;
      H.fixed := FALSE;
      G.fixed := FALSE;
      y[components].fixed := FALSE;
      alpha[components].fixed := FALSE;
      ave_alpha.fixed := FALSE;
      phi[phases].fixed := FALSE;
      END clear;
METHOD specify;
      (* this method buggy if number_stuck > 0 components *)
      RUN light.specify;
      RUN heavy.specify;
      V.fixed := TRUE;
      H.fixed := TRUE;
      G.fixed := TRUE;
      alpha[components].fixed := TRUE;
      y[light.options.ds.other].fixed := TRUE;
      light.y[components].fixed := FALSE;
      phi[light.options.phase].fixed := TRUE;
END specify;
METHOD scale;
      FOR i IN components DO
                  y[i].nominal := y[i];
            alpha[i].nominal := alpha[i];
            alpha[i].lower_bound := 1e-8;
            alpha[i].upper_bound :=
                  alpha[i] + scale * alpha[i].nominal;
            END;
      RUN light.scale;
      RUN heavy.scale;
END scale;
END two_phase_mixture;

(* alpha_mixture gone away *)

MODEL td_homogeneous_mixture(
      P WILL_BE pressure;
      T WILL_BE temperature;
      options WILL_BE single_phase_options;
) REFINES homogeneous_mixture;

      ds ALIASES options.ds;
      FOR i IN ds.components CREATE
            pure[i] IS_A
            pure_component(P, T, ds.data[i], options.phase,
```

```
                        options.component_thermo_correlation);
      END;
      partial[ds.components] IS_A partial_component;

td_homogeneous_mixture_V:
      V * (ds.data[ds.reference].Pc /
      (1{GAS_C}*ds.data[ds.reference].Tc)) =
      SUM[ y[i] * partial[i].V | i IN ds.components] *
            (ds.data[ds.reference].Pc /
             (1{GAS_C}*ds.data[ds.reference].Tc));
td_homogeneous_mixture_H:
      H / (1{GAS_C} * ds.data[ds.reference].Tc) =
            SUM[ y[i] * partial[i].H | i IN ds.components]/
                  (1{GAS_C} * ds.data[ds.reference].Tc);
td_homogeneous_mixture_G:
      G / (1{GAS_C} * ds.data[ds.reference].Tc) =
            SUM[ y[i] * partial[i].G | i IN ds.components] /
                  (1{GAS_C} * ds.data[ds.reference].Tc);

      y[ds.components].lower_bound := 1.0e-12;

METHODS
METHOD clear;
      RUN pure[ds.components].clear;
      RUN partial[ds.components].clear;
      y[ds.components].fixed := FALSE;
      V.fixed := FALSE;
      H.fixed := FALSE;
      G.fixed := FALSE;
END clear;
METHOD specify;
      RUN pure[ds.components].specify;
      RUN partial[ds.components].specify;
      y[ds.components].fixed := TRUE;
      y[ds.reference].fixed := FALSE;
      END specify;
      METHOD scale;
      RUN pure[ds.components].scale;
      RUN partial[ds.components].scale;
      FOR i IN ds.components DO
            y[i].nominal := y[i];
      END;
      T.nominal := T;
      P.nominal := P;
      V.nominal := V;
      H.nominal := sqrt(sqr(H));
      G.nominal := sqrt(sqr(G));
      T.lower_bound := 1.0e-8{K};
      P.lower_bound := 1.0e-8{atm};
      V.lower_bound := 1.0e-8{liter/g_mole};
```

```
      H.lower_bound := H - scale*H.nominal;
      G.lower_bound := G - scale*G.nominal;
      T.upper_bound := T + scale*T.nominal;
      P.upper_bound := P + scale*P.nominal;
      V.upper_bound := V + scale*V.nominal;
      H.upper_bound := H + scale*H.nominal;
      G.upper_bound := G + scale*G.nominal;
END scale;

END td_homogeneous_mixture;

MODEL td_two_phase_mixture(
(* was td_alpha_mixture *)
      P WILL_BE pressure;
      T WILL_BE temperature;
      light WILL_BE td_homogeneous_mixture;
      heavy WILL_BE td_homogeneous_mixture;
      equilibrated WILL_BE boolean;
) WHERE (
      P, heavy.P, light.P WILL_BE_THE_SAME;
      T, heavy.T, light.T WILL_BE_THE_SAME;
      light.options.phase != heavy.options.phase;
      light.options.ds.reference == heavy.options.ds.reference;
      (* constrain the reference component to be a mobile species.*)
      INTERSECTION[light.options.ds.components,
                  heavy.options.ds.components,
                  [light.options.ds.reference]
      ] == [light.options.ds.reference];
      light.options.ds.components == heavy.options.ds.components;
) REFINES two_phase_mixture;

      data ALIASES heavy.options.ds.data;
      ds ALIASES heavy.options.ds;

td_two_phase_mixture_V:
      V * (data[ds.reference].Pc /
            ((1{GAS_C})*data[ds.reference].Tc)) =
      (phi[light.options.phase] * light.V +
       phi[heavy.options.phase] * heavy.V) *
      (data[ds.reference].Pc / ((1{GAS_C})*data[ds.reference].Tc));

td_two_phase_mixture_H:
      H/((1{GAS_C})*data[ds.reference].Tc) =
      (phi[light.options.phase] * light.H +
       phi[heavy.options.phase] * heavy.H) /
      ((1{GAS_C})*data[ds.reference].Tc);

td_two_phase_mixture_G:
      G/((1{GAS_C})*data[ds.reference].Tc) =
      (phi[light.options.phase] * light.G +
```

```
       phi[heavy.options.phase] * heavy.G) /
      ((1{GAS_C})*data[ds.reference].Tc);

      FOR i IN ds.components CREATE
      equil_condition[i]:
      (light.partial[i].G - heavy.partial[i].G ) / (1E5{J/mole}) = 0;
      (* oops, can't scale with G0 because hydrogen G0 = 0.0 *)
      END;

NOTES
'default' equilibrated {
The equilibrium condition can be relaxed, effectively making this
model devolve to td_heterogeneous mixture.
};
END NOTES;

      WHEN (equilibrated)
      CASE TRUE:
            USE equil_condition;
      END;

      V := 30{liter/mol};
      equilibrated := TRUE;


  METHODS
      METHOD clear;
      RUN two_phase_mixture::clear;
      V.fixed := FALSE;
      H.fixed := FALSE;
      G.fixed := FALSE;
      END clear;

      METHOD reset_to_massbal;
          RUN clear;
          equilibrated := FALSE;
          P.fixed:= TRUE;
          T.fixed := TRUE;
          alpha[ds.components].fixed:= TRUE;
          phi[light.options.phase].fixed:= TRUE;
      END reset_to_massbal;

      METHOD reset_to_fullthermo;
          RUN clear;
          equilibrated := TRUE;
          P.fixed:=TRUE;
          ave_alpha.fixed  := TRUE;
          phi[light.options.phase].fixed:= TRUE;
      END reset_to_fullthermo;
```

```
      METHOD scale;
      RUN two_phase_mixture::scale;
      T.nominal := T;
      P.nominal := P;
      V.nominal := V;
      H.nominal := sqrt(sqr(H));
      G.nominal := sqrt(sqr(G));
      T.lower_bound := 1.0e-8{K};
      P.lower_bound := 1.0e-8{atm};
      V.lower_bound := 1.0e-8{liter/g_mole};
      H.lower_bound := H - scale*H.nominal;
      G.lower_bound := G - scale*G.nominal;
      T.upper_bound := T + scale*T.nominal;
      P.upper_bound := P + scale*P.nominal;
      V.upper_bound := V + scale*V.nominal;
      H.upper_bound := H + scale*H.nominal;
      G.upper_bound := G + scale*G.nominal;
      END scale;

END td_two_phase_mixture;

(* td_equilibrium_mixture merged with td_two_phase *)

MODEL vapor_component(
      P WILL_BE pressure;
      T WILL_BE temperature;
      data WILL_BE td_component_constants;
      correlation WILL_BE symbol_constant;
) REFINES pure_component(phase :== ‘vapor’;);
(* no body *)
END vapor_component;

MODEL liquid_component(
      P WILL_BE pressure;
      T WILL_BE temperature;
      data WILL_BE td_component_constants;
      correlation WILL_BE symbol_constant;
) REFINES pure_component(phase :== ‘liquid’;);

      VP IS_A pressure;
END liquid_component;

MODEL Pitzer_vapor_component(
      P WILL_BE pressure;
      T WILL_BE temperature;
      data WILL_BE td_component_constants;
      correlation WILL_BE symbol_constant WITH_VALUE ‘Pitzer’;
) REFINES vapor_component;

Pitzer_component_V:
```

```
            P*V/(1{GAS_C})/data.Tc = T/data.Tc + (P/data.Pc)*
      (0.083 - 0.422*(data.Tc/T)^1.6 + data.omega*
            (0.139 - 0.172*(data.Tc/T)^4.2));

Pitzer_component_H:
      H/(1{GAS_C})/data.Tc = data.H0/(1{GAS_C})/data.Tc +
      data.cpvapa*(T - data.T0)/(1{GAS_C})/data.Tc +
      data.cpvapb*(T^2 - data.T0^2)/2/(1{GAS_C})/data.Tc +
      data.cpvapc*(T^3 - data.T0^3)/3/(1{GAS_C})/data.Tc +
      data.cpvapd*(T^4 - data.T0^4)/4/(1{GAS_C})/data.Tc +
      (P/data.Pc)*(0.083 - 1.097*(data.Tc/T)^1.6 + data.omega*
            (0.139 - 0.894*(data.Tc/T)^4.2));

Pitzer_component_G:
            G/(1{GAS_C})/data.Tc = data.G0/(1{GAS_C})/data.Tc -
            (data.H0 - data.G0)*(T/data.T0 - 1)/(1{GAS_C})/data.Tc -
            data.cpvapa*(T*ln(T/data.T0) - T + data.T0)
                  /1{GAS_C}/data.Tc -
            data.cpvapb*(T^2 - 2*T*data.T0 + data.T0^2)/
            (2*1{GAS_C}*data.Tc) -
            data.cpvapc*(T^3/2 - 3*T*data.T0^2/2 + data.T0^3)/
            (3*1{GAS_C}*data.Tc) -
            data.cpvapd*(T^4/3 - 4*T*data.T0^3/3 + data.T0^4)/4
                  /(1{GAS_C})/data.Tc +
            T*ln(P/data.P0)/data.Tc +
            (P/data.Pc)*
            (0.083 - 0.422*(data.Tc/T)^1.6 + data.omega*
            (0.139 - 0.172*(data.Tc/T)^4.2));

      T.lower_bound := 1.0{K};
      P.lower_bound := 0.1{atm};
      T.upper_bound := 1000{K};
      P.upper_bound := 20{atm};
      V := 24{liter/mole};

METHODS
METHOD specify;
      T.fixed := TRUE;
      P.fixed := TRUE;
END specify;
END Pitzer_vapor_component;

MODEL Rackett_liquid_component(
      P WILL_BE pressure;
      T WILL_BE temperature;
      data WILL_BE td_component_constants;
      correlation WILL_BE symbol_constant WITH_VALUE ‘Rackett’;
      vp_correlation WILL_BE integer_constant;
) REFINES liquid_component;
(*Rackett_component_VP:*)
```

```
SELECT (vp_correlation)
CASE 1:
      ln(VP/data.Pc)*T/data.Tc*abs(data.Tc - T) =
      (data.vpa * abs(1.0 - T/data.Tc) +
      data.vpb * abs(1.0 - T/data.Tc)^1.5 +
      data.vpc * abs(1.0 - T/data.Tc)^3.0 +
      data.vpd * abs(1.0 - T/data.Tc)^6.0) * (data.Tc - T);
END;

Rackett_component_V: V/data.Vliq =
      data.Zc^(abs(1.0 - T/data.Tc)^(2/7))/
      data.Zc^(abs(1.0 - data.Tliq/data.Tc)^(2/7));

Rackett_component_H:
      H / (1{GAS_C} * data.Tc) = data.H0/(1{GAS_C})/data.Tc +
      (T - data.T0)  * (data.cpvapa/(1{GAS_C})/data.Tc)   +
      (T^2 - data.T0^2)  * (data.cpvapb/2/(1{GAS_C})/data.Tc) +
      (T^3 - data.T0^3)  * (data.cpvapc/3/(1{GAS_C})/data.Tc) +
      (T^4 - data.T0^4)  * (data.cpvapd/4/(1{GAS_C})/data.Tc) +
      (VP/data.Pc)*
      (0.083 - 1.097*(data.Tc/T)^1.6 + data.omega*
      (0.139 - 0.894*(data.Tc/T)^4.2)) -
      (data.Hv/(1{GAS_C})/data.Tc)*
      abs((data.Tc-T)/(data.Tc-data.Tb))^0.38 +
      (P - VP)*(data.Vliq/(1{GAS_C})/data.Tc)*
      (data.Zc^(abs(1.0 - T/data.Tc)^(2/7))/
      data.Zc^(abs(1.0 - data.Tliq/data.Tc)^(2/7)))*(1.0 -
      (-2/7)*(T/data.Tc)*(abs(1-T/data.Tc)^(-5/7))*ln(data.Zc));

Rackett_component_G: G/(1{GAS_C}*data.Tc) =
      data.G0/(1{GAS_C})/data.Tc -
      (T/data.T0 - 1) * ((data.H0 - data.G0)/(1{GAS_C})/data.Tc) -
      (T*ln(T/data.T0) - T + data.T0) *
      (data.cpvapa/(1{GAS_C})/data.Tc) -
      (T^2 - T*(2*data.T0) + data.T0^2) *
      (data.cpvapb/2/(1{GAS_C})/data.Tc) -
      (T^3/2 - T*(3/2*data.T0^2) + data.T0^3) *
      (data.cpvapc/3/(1{GAS_C})/data.Tc) -
      (T^4/3 - T*(4/3*data.T0^3) + data.T0^4) *
      (data.cpvapd/4/(1{GAS_C})/data.Tc) +
      T*ln(VP/data.P0)/data.Tc +
      (VP/data.Pc) *
      (0.083 - 0.422*(data.Tc/T)^1.6 + data.omega *
      (0.139 - 0.172*(data.Tc/T)^4.2)) +
      (P - VP)*(data.Vliq/(1{GAS_C})/data.Tc) *
      (data.Zc^(abs(1.0 - T/data.Tc)^(2/7))/
      data.Zc^(abs(1.0 - data.Tliq/data.Tc)^(2/7)));

VP.lower_bound := 1.0e-12{Pa};
V := 0.1{liter/mole};
```

```
      T.lower_bound := 1.0{K};
      P.lower_bound := 0.1{atm};
      T.upper_bound := 1000{K};
      P.upper_bound := 20{atm};

 METHODS

      METHOD clear;
      RUN liquid_component::clear;
      VP.fixed := FALSE;
      END clear;
      METHOD specify;
      T.fixed := TRUE;
      P.fixed := TRUE;
      END specify;
      METHOD scale;
      RUN liquid_component::scale;
      VP.nominal := VP;
      VP.lower_bound := 1.0e-8{atm};
      VP.upper_bound := VP + scale*VP.nominal;
      END scale;
END Rackett_liquid_component;

MODEL Pitzer_partials(
      P WILL_BE pressure;
      T WILL_BE temperature;
      V WILL_BE molar_volume;
      H WILL_BE molar_energy;
      G WILL_BE molar_energy;
      scale WILL_BE scaling_constant;
      options WILL_BE single_phase_options;
      y[options.ds.components] WILL_BE mole_fraction;
      pure[options.ds.components] WILL_BE Pitzer_vapor_component;
      partial[options.ds.components] WILL_BE partial_component;
);

      ds ALIASES options.ds;

      FOR i IN ds.components CREATE
      Pitzer_mixture_V[i]:
             (partial[i].V - pure[i].V)*(ds.data[i].Pc/
            (1{GAS_C}*ds.data[i].Tc)) =
            -1.0*(0.083 - 0.422*(ds.data[i].Tc/T)^1.6 +
            ds.data[i].omega*
            (0.139 - 0.172*(ds.data[i].Tc/T)^4.2))*(1.0 - y[i])^2 +
            0.50*(1.0 - y[i])*
            SUM[y[j]*((1.0 + (ds.data[j].Vc/ds.data[i].Vc)^(1/3))^3/
            (1.0 + ds.data[j].Zc/ds.data[i].Zc))*
            (0.083 - 0.422*(sqrt(ds.data[i].Tc*ds.data[j].Tc) /T)^1.6
+
```

```
            0.5*(ds.data[i].omega + ds.data[j].omega)*
            (0.139 - 0.172*(sqrt(ds.data[i].Tc*ds.data[j].Tc) / T)^4.2))
             | j IN ds.components - [i]];
              Pitzer_mixture_H[i]:
            (partial[i].H - pure[i].H)/((1{GAS_C})*ds.data[i].Tc) =
            -(P/ds.data[i].Pc)*
            (0.083 - 1.097*(ds.data[i].Tc/T)^1.6 + ds.data[i].omega*
            (0.139 - 0.894*(ds.data[i].Tc/T)^4.2))*(1.0 - y[i])^2 +
            (1.0 - y[i])*(P/(ds.data[i].Pc*2))*
            SUM[y[j]*((1.0 + (ds.data[j].Vc/ds.data[i].Vc)^(1/3))^3/
            (1.0 + ds.data[j].Zc/ds.data[i].Zc))*
            (0.083 - 1.097*(sqrt(ds.data[i].Tc*ds.data[j].Tc) /T)^1.6 +
            0.5*(ds.data[i].omega + ds.data[j].omega)*
            (0.139 - 0.894*(sqrt(ds.data[i].Tc*ds.data[j].Tc) / T)^4.2))
              | j IN ds.components - [i]];
              Pitzer_mixture_G[i]:
            (partial[i].G - pure[i].G - (1{GAS_C})*T*ln(y[i])) /
            (1{GAS_C}*ds.data[i].Tc) =
            -(P/ds.data[i].Pc)*
            (0.083 - 0.422*(ds.data[i].Tc/T)^1.6 + ds.data[i].omega*
            (0.139 - 0.172*(ds.data[i].Tc/T)^4.2))*(1.0 - y[i])^2 +
            (1.0 - y[i])*(P/(ds.data[i].Pc*2))*
            SUM[y[j]*((1.0 + (ds.data[j].Vc/ds.data[i].Vc)^(1/3))^3/
            (1.0 + ds.data[j].Zc/ds.data[i].Zc))*
            (0.083 - 0.422*(sqrt(ds.data[i].Tc*ds.data[j].Tc) /T)^1.6 +
            0.5*(ds.data[i].omega + ds.data[j].omega)*
            (0.139 - 0.172*(sqrt(ds.data[i].Tc*ds.data[j].Tc) / T)^4.2))
            | j IN ds.components - [i]];
      END;
      T.lower_bound := 1.0{K};
      P.lower_bound := 1.0{Pa};
      V := 24{liter/mole};
      partial[ds.components].V := 24{liter/mole};
      pure[ds.components].V := 24{liter/mole};

METHODS

      METHOD specify;
      partial[ds.components].V.fixed := FALSE;
            partial[ds.components].H.fixed := FALSE;
            partial[ds.components].G.fixed := FALSE;
      END specify;

END Pitzer_partials;
```

```
(*UNIFAC_partials is a model which helps to select mixing rules at the
* liquid mixture level, hiding unneccessary code from users.
*)
MODEL UNIFAC_partials(
      P WILL_BE pressure;
      T WILL_BE temperature;
      V WILL_BE molar_volume;
      H WILL_BE molar_energy;
      G WILL_BE molar_energy;
      scale WILL_BE scaling_constant;
      options WILL_BE single_phase_options;
      y[options.ds.components] WILL_BE mole_fraction;
      pure[options.ds.components] WILL_BE Rackett_liquid_component;
      partial[options.ds.components] WILL_BE partial_component;
);

      ds ALIASES options.ds;
      subgroups   IS_A set OF symbol_constant;
      groups            IS_A set OF integer_constant;
      comps[subgroups]IS_A set OF symbol_constant;
      rv[ds.components]IS_A UNIFAC_size;
      qs[ds.components]IS_A UNIFAC_size;
      Jv[ds.components]IS_A factor;
      Ls[ds.components]IS_A factor;
      theta[subgroups]IS_A factor;
      eta[subgroups]IS_A factor;
      uc                 IS_A UNIFAC_constants;

      subgroups :== UNION[ds.data[i].subgroups | i IN ds.components];
      groups :== UNION[ds.data[i].groups | i IN ds.components];
      FOR k IN subgroups CREATE
            comps[k] :==
            [i IN ds.components | k IN ds.data[i].subgroups];
      END;
      FOR k IN subgroups CREATE
            UNIFAC_mixture_theta[k]:
            theta[k] = uc.Q[k]*SUM[ds.data[i].nu[k]*y[i]
                                    |i IN comps[k]];

            UNIFAC_mixture_eta[k]:
            eta[k] =
            SUM[theta[m] | m IN subgroups*uc.sub[uc.group[k]]] +
            SUM[SUM[theta[m]*exp(-uc.a[g][uc.group[k]]/T)
            | m IN subgroups*uc.sub[g]]
            | g IN groups - [uc.group[k]]];
      END;

      FOR i IN ds.components CREATE
            rv[i] :== SUM[0, ds.data[i].nu[k]*uc.R[k]
                        | k IN ds.data[i].subgroups];
```

```
      qs[i] :== SUM[0, ds.data[i].nu[k]*uc.Q[k]
                  | k IN ds.data[i].subgroups];
END;
FOR i IN ds.components CREATE
      UNIFAC_mixture_rv[i]:
      rv[i] = Jv[i]*SUM[rv[j]*y[j] | j IN ds.components];

      UNIFAC_mixture_qs[i]:
      qs[i] = Ls[i]*SUM[qs[j]*y[j] | j IN ds.components];
      partial[i].V = pure[i].V;

      UNIFAC_mixture_H_excess[i]:
      (partial[i].H - pure[i].H)/(1{GAS_C})/ds.data[i].Tc =
      SUM[0,theta[k]*
      SUM[0,SUM[0,theta[n]*
      ((uc.a[g][uc.group[k]] -
      uc.a[uc.group[n]][uc.group[k]])/ds.data[i].Tc)*
      exp(-(uc.a[g][uc.group[k]] +
      uc.a[uc.group[n]][uc.group[k]])/T)*
      SUM[0,ds.data[i].nu[m]*uc.Q[m]
      | m IN ds.data[i].subgroups*uc.sub[g]]
      | g IN ds.data[i].groups - [uc.group[n]]]
      | n IN subgroups]/eta[k]/eta[k]
      | k IN subgroups] -
      SUM[0,(ds.data[i].nu[k]*uc.Q[k]/(
      SUM[0,ds.data[i].nu[m]*uc.Q[m]
      | m IN ds.data[i].subgroups*uc.sub[uc.group[k]]] +
      SUM[0,SUM[0,ds.data[i].nu[m]*uc.Q[m] *
                  exp(-uc.a[g][uc.group[k]]/T)
      | m IN ds.data[i].subgroups*uc.sub[g]]
      | g IN ds.data[i].groups - [uc.group[k]]]]))*
      SUM[0,SUM[0,theta[n]*
      ((uc.a[g][uc.group[k]] -
      uc.a[uc.group[n]][uc.group[k]])/ds.data[i].Tc)*
      exp(-(uc.a[g][uc.group[k]] +
      uc.a[uc.group[n]][uc.group[k]])/T)*
      SUM[0,ds.data[i].nu[m]*uc.Q[m]
      | m IN ds.data[i].subgroups*uc.sub[g]]
      | g IN ds.data[i].groups - [uc.group[n]]]
      | n IN subgroups]/eta[k]
      | k IN ds.data[i].subgroups];

      UNIFAC_mixture_G_excess[i]:
      (partial[i].G - pure[i].G - (1{GAS_C})*T*ln(y[i])) /
      (1{GAS_C}*ds.data[i].Tc) =
      (1.0 - Jv[i] + ln(Jv[i]) -
      5.0*qs[i]*(1.0 - Jv[i]/Ls[i] + ln(Jv[i]/Ls[i])) +
      qs[i]*(1 - ln(Ls[i])))*T/ds.data[i].Tc -
      SUM[0,theta[k]*(
      SUM[0,ds.data[i].nu[m]*uc.Q[m]
```

```
            | m IN ds.data[i].subgroups*uc.sub[uc.group[k]]] +
            SUM[0,SUM[0,ds.data[i].nu[m] * uc.Q[m] *
                        exp(-uc.a[g][uc.group[k]]/T)
            | m IN ds.data[i].subgroups*uc.sub[g]]
            | g IN ds.data[i].groups - [uc.group[k]]])/eta[k]
            | k IN subgroups]*T/ds.data[i].Tc +
            SUM[0,ds.data[i].nu[k]*uc.Q[k]*ln((
            SUM[0,ds.data[i].nu[m]*uc.Q[m]
            | m IN ds.data[i].subgroups*uc.sub[uc.group[k]]] +
            SUM[0,SUM[0, ds.data[i].nu[m] * uc.Q[m] *
                        exp(-uc.a[g][uc.group[k]]/T)
            | m IN ds.data[i].subgroups*uc.sub[g]]
            | g IN ds.data[i].groups - [uc.group[k]]])/eta[k])
            | k IN ds.data[i].subgroups]*T/ds.data[i].Tc;
      END;

      T.lower_bound := 1.0{K};
      P.lower_bound := 1.0{Pa};
      Jv[ds.components].lower_bound := 1.0e-8;
      Ls[ds.components].lower_bound := 1.0e-8;
      theta[subgroups].lower_bound := 0.0;
      eta[subgroups].lower_bound := 0.0;
      V := 0.1{liter/mole};
      partial[ds.components].V := 0.1{liter/mole};
      pure[ds.components].V := 0.1{liter/mole};

      FOR i IN ds.components CREATE
            rv[i] :== SUM[0,ds.data[i].nu[k]*uc.R[k]
                        | k IN ds.data[i].subgroups];
            qs[i] :== SUM[0,ds.data[i].nu[k]*uc.Q[k]
                        | k IN ds.data[i].subgroups];
      END;

METHODS
METHOD clear;
      Jv[ds.components].fixed := FALSE;
      Ls[ds.components].fixed := FALSE;
      theta[subgroups].fixed := FALSE;
      eta[subgroups].fixed := FALSE;
END clear;
METHOD specify;
      partial[ds.components].V.fixed := FALSE;
      partial[ds.components].H.fixed := FALSE;
      partial[ds.components].G.fixed := FALSE;

END specify;
METHOD scale;
      FOR i IN ds.components DO
            Ls[i].nominal := Ls[i];
            Jv[i].nominal := Jv[i];
```

```
            Ls[i].upper_bound := Ls[i]+ scale*Ls[i].nominal;
            Jv[i].upper_bound := Jv[i] + scale*Jv[i].nominal;
      END;
      FOR j IN subgroups DO
            theta[j].nominal := theta[j];
            eta[j].nominal := eta[j];
            theta[j].upper_bound := theta[j] +
                                    scale*theta[j].nominal;
            eta[j].upper_bound := eta[j] + scale*eta[j].nominal;
      END;
END scale;
END UNIFAC_partials;

MODEL Wilson_partials(
      P WILL_BE pressure;
      T WILL_BE temperature;
      V WILL_BE molar_volume;
      H WILL_BE molar_energy;
      G WILL_BE molar_energy;
      scale WILL_BE scaling_constant;
      options WILL_BE single_phase_options;
      y[options.ds.components] WILL_BE mole_fraction;
      pure[options.ds.components] WILL_BE Rackett_liquid_component;
      partial[options.ds.components] WILL_BE partial_component;
);

      ds ALIASES options.ds;
      lambda[ds.components][ds.components]IS_A factor;

      FOR i IN ds.components CREATE
            FOR j IN ds.components CREATE
                  lambda[i][j] = (pure[j].V / pure[i].V) *
                  exp(-pure[i].data.del_ip[pure[j].data.formula] /
                  (1{GAS_C} * T));
            END;
      END;

      FOR i IN ds.components CREATE
            partial[i].V = pure[i].V;

      Wilson_mixture_G_excess[i]:
            partial[i].G - pure[i].G - (1{GAS_C})*T*ln(y[i]) =
            (1{GAS_C})*T*(-ln( SUM[y[j]*lambda[i][j]
                                    | j IN ds.components])
             + 1 -
            SUM[(y[k] * lambda[k][i]) / SUM[ y[j] * lambda[k][j]
            | j IN ds.components] | k IN ds.components]);

      Wilson_mixture_H_excess[i]:
            partial[i].H - pure[i].H =
```

```
            (SUM[ y[j] * lambda[i][j] *
             pure[i].data.del_ip[pure[j].data.formula]
            | j IN ds.components]) /
            (SUM[y[s]*lambda[i][s]
            | s IN ds.components]) -
            (SUM[y[k]*lambda[k][i] | k IN ds.components]*
            SUM[SUM[y[l]*lambda[m][l]*
             pure[m].data.del_ip[pure[l].data.formula]
            | l IN ds.components]
            | m IN ds.components]) / (
            SUM[SUM[y[n]*lambda[o][n]
            | n IN ds.components]
            | o IN ds.components])^2 +
            (SUM[y[p]*lambda[p][i]*
             pure[p].data.del_ip[pure[i].data.formula]
             | p IN ds.components]) /
             (SUM[SUM[y[q]*lambda[r][q]
             | q IN ds.components]
             | r IN ds.components]);
      END;

      V := 0.1{liter/mole};
      partial[ds.components].V := 0.1{liter/mole};
      pure[ds.components].V := 0.1{liter/mole};
METHODS
METHOD clear;
      lambda[ds.components][ds.components].fixed := FALSE;
END clear;
METHOD specify;
      partial[ds.components].V.fixed := FALSE;
      partial[ds.components].G.fixed := FALSE;
      partial[ds.components].H.fixed := FALSE;
END specify;
METHOD scale;
      FOR i IN ds.components DO
            FOR j IN ds.components DO
            lambda[i][j].nominal :=
                  lambda[i][j];
            lambda[i][j].lower_bound := lambda[i][j] -
                  scale*lambda[i][j].nominal;
            lambda[i][j].upper_bound := lambda[i][j] +
                  scale*lambda[i][j].nominal;
            END;
      END;
END scale;
END Wilson_partials;

MODEL vapor_mixture(
      P WILL_BE pressure;
      T WILL_BE temperature;
```

```
      options WILL_BE vapor_phase_options;
) REFINES td_homogeneous_mixture;

      SELECT (options.mixture_thermo_correlation)
      CASE ‘Pitzer’:
            FOR i IN ds.components CREATE
                  pure[i] IS_REFINED_TO
                  Pitzer_vapor_component(P, T, ds.data[i],
                  options.component_thermo_correlation);
            END;
            Pitzer_mixing_rule IS_A
            Pitzer_partials(P, T, V, H, G, scale, options,
                        y, pure, partial);
      END;

METHODS
      METHOD specify;
            RUN td_homogeneous_mixture::specify;
            IF options.mixture_thermo_correlation == ‘Pitzer’ THEN
                  RUN Pitzer_mixing_rule.specify;
            END;
      END specify;

END vapor_mixture;

MODEL liquid_mixture(
      P WILL_BE pressure;
      T WILL_BE temperature;
      options WILL_BE liquid_phase_options;
) REFINES td_homogeneous_mixture;

      SELECT (options.mixture_thermo_correlation)
      CASE ‘UNIFAC’:
            FOR i IN ds.components CREATE
                  pure[i] IS_REFINED_TO
                  Rackett_liquid_component(P, T, ds.data[i],
                  options.component_thermo_correlation,
                  ds.data[i].vp_correlation);
            END;
            UNIFAC_mixing_rule IS_A
            UNIFAC_partials(P, T, V, H, G, scale,
                        options, y, pure, partial);

      CASE ‘Wilson’:
            FOR i IN ds.components CREATE
                  pure[i] IS_REFINED_TO
                  Rackett_liquid_component(P, T, ds.data[i],
                  options.component_thermo_correlation,
                  ds.data[i].vp_correlation);
            END;
```

```
            Wilson_mixing_rule IS_A
            Wilson_partials(P, T, V, H, G, scale,
                        options, y, pure, partial);
      END; (* select correlation *)

METHODS
      METHOD clear;
            RUN td_homogeneous_mixture::clear;
            IF options.mixture_thermo_correlation == ‘UNIFAC’ THEN
                  RUN UNIFAC_mixing_rule.clear;
            END;
            IF options.mixture_thermo_correlation == ‘Wilson’ THEN
                  RUN Wilson_mixing_rule.clear;
            END;
      END clear;

      METHOD specify;
            RUN td_homogeneous_mixture::specify;
            IF (options.mixture_thermo_correlation == ‘UNIFAC’) OR
            (options.mixture_thermo_correlation = ‘Wilson’) THEN
                  partial[ds.components].V.fixed := FALSE;
                  partial[ds.components].H.fixed := FALSE;
                  partial[ds.components].G.fixed := FALSE;
            END;
      END specify;
      METHOD scale;
            RUN td_homogeneous_mixture::scale;
            IF options.mixture_thermo_correlation == ‘UNIFAC’ THEN
                  RUN UNIFAC_mixing_rule.scale;
            END;
            IF options.mixture_thermo_correlation == ‘Wilson’ THEN
                  RUN Wilson_mixing_rule.scale;
            END;
      END scale;
END liquid_mixture;

MODEL td_VLE_mixture(
      P WILL_BE pressure;
      T WILL_BE temperature;
      light WILL_BE vapor_mixture;
      heavy WILL_BE liquid_mixture;
      equilibrated WILL_BE boolean;
) WHERE (
      P, heavy.P, light.P WILL_BE_THE_SAME;
      T, heavy.T, light.T WILL_BE_THE_SAME;
      light.options.phase != heavy.options.phase;
      light.options.ds.reference == heavy.options.ds.reference;
      (* constrain the reference component to be a mobile species.*)
      INTERSECTION[light.options.ds.components,
                  heavy.options.ds.components,
```

```
                  [light.options.ds.reference]
      ] == [light.options.ds.reference];
      light.options.ds.components == heavy.options.ds.components;
      light.options.phase == 'vapor';
      heavy.options.phase == 'liquid';
) REFINES td_two_phase_mixture;
(* no body *)

METHODS
    METHOD specify;
      P.fixed:= TRUE;
      T.fixed := TRUE;
      V.fixed := FALSE;
      H.fixed := FALSE;
      G.fixed:= FALSE;
    END specify;

END td_VLE_mixture;

MODEL murphree_equilibrium_mixture(
      P WILL_BE pressure;
      T WILL_BE temperature;
      light WILL_BE vapor_mixture;
      heavy WILL_BE liquid_mixture;
      equilibrated WILL_BE boolean;
) WHERE (
      P, heavy.P, light.P WILL_BE_THE_SAME;
      T, heavy.T, light.T WILL_BE_THE_SAME;
      light.options.phase != heavy.options.phase;
      light.options.ds.reference == heavy.options.ds.reference;
      (* constrain the reference component to be a mobile species.*)
      INTERSECTION[light.options.ds.components,
                  heavy.options.ds.components,
                  [light.options.ds.reference]
      ] == [light.options.ds.reference];
      light.options.ds.components == heavy.options.ds.components;
      light.options.phase == 'vapor';
      heavy.options.phase == 'liquid';
) REFINES td_VLE_mixture;

booleqnlabel:equilibrated == TRUE; (* logical constraint *)

      alpha[ds.components],
      ave_alpha IS_REFINED_TO relative_volatility;

      vap_eq      IS_A vapor_mixture(light.P,light.T,light.options);

      equil_alpha[ds.components]IS_A relative_volatility;
      ref_y[ds.components]IS_A fraction;
      murph_eff         IS_A factor;
```

```
murphee_equil_ref_y_def:
      SUM[ref_y[ds.components]] = 1;

      FOR i IN ds.components CREATE
            murphee_equil_vap_eq_y_def[i]:
            vap_eq.y[i] = equil_alpha[i] * heavy.y[i];
            (vap_eq.partial[i].G-heavy.partial[i].G)/1E+5{J/mole}=0;
      END;

      FOR i IN ds.other CREATE
            murphee_equil_murph_eff_def[i]:
            murph_eff*(vap_eq.y[i] - ref_y[i]) = light.y[i] - ref_y[i];
      END;

METHODS
      METHOD clear;
            RUN td_VLE_mixture::clear;
            RUN vap_eq.clear;
            equil_alpha[ds.components].fixed := FALSE;
            ref_y[ds.components].fixed := FALSE;
            murph_eff.fixed := FALSE;
      END clear;
      METHOD specify;
            RUN td_two_phase_mixture::specify;
            RUN vap_eq.specify;
            vap_eq.y[ds.components].fixed := FALSE;
            equil_alpha[ds.components].fixed := FALSE;
            ref_y[ds.other].fixed := TRUE;
            murph_eff.fixed := TRUE;
            T.fixed := FALSE;
      END specify;

END murphree_equilibrium_mixture;
```

A.2.6 STREAM

Appendix A Code 9(* stream.lib *)

```
(* ******************+********************************** *)
(* ******      homogeneous_molar_stream ******************** *)
(* ******************+********************************** *)

MODEL homogeneous_molar_stream(
      state WILL_BE homogeneous_mixture;
);

      components ALIASES state.options.ds.components;
      Ftot,
      f[components]IS_A molar_rate;
      scale             IS_A scaling_constant;
      Ftot_scale  IS_A real;

      FOR i IN components CREATE
          f_def[i]: f[i]/Ftot_scale = Ftot*state.y[i]/Ftot_scale;
      END;

      Ftot_scale := 1000 {mole/s};
(* horrible scaling method - global var, ack!
Ken Tyner says we don't need this any more,anyway.*)

  METHODS

    METHOD clear;
      RUN state.clear;
      Ftot.fixed              := FALSE;
      f[components].fixed:= FALSE;
      f_def[components].included:= TRUE;
    END clear;

    METHOD seqmod;
      RUN state.specify;
      state.y[components].fixed:= FALSE;
    END seqmod;

    METHOD specify;
      RUN seqmod;
      f[components].fixed:= TRUE;
    END specify;

    METHOD reset;
      RUN clear;
      RUN specify;
    END reset;

    METHOD scale;
```

```
      RUN state.scale;
      FOR i IN components DO
          f[i].nominal := f[i] + 0.1{mol/s};
          f[i].upper_bound := f[i] + scale*f[i].nominal;
      END;
      Ftot.nominal := Ftot + 0.1{mol/s};
      Ftot.upper_bound := Ftot + scale*Ftot.nominal;
      Ftot_scale := Ftot.nominal;
    END scale;
END homogeneous_molar_stream;

(* ******************+********************************** *)
(* ****** homogeneous_thermodynamic_stream ***************** *)
(* ******************+********************************** *)

MODEL homogeneous_td_stream(
      state WILL_BE homogeneous_mixture;
) REFINES homogeneous_molar_stream;

NOTES
‘usage’ SELF {
td_stream here does NOT denote that the state is a td_homo or td_hetero
mixture. It merely indicates that total H is computed so that certain
thermodynamic principles may be applied using this type of stream.
state.H may be an arbitrary number if the state does not compute
it from some correlation.
}
END NOTES;

      H                        ALIASES state.H;
      Htot                     IS_A energy_rate;
      Htot_scale         IS_A real;

      Htot_def: Htot/Htot_scale = H*Ftot/Htot_scale;

      Htot_scale := 1{MW};

  METHODS

    METHOD clear;
      RUN state.clear;
      Ftot.fixed        := FALSE;
      f[components].fixed:= FALSE;
      Htot.fixed        := FALSE;
      H.fixed                 := FALSE;
      f_def[components].included:= TRUE;
      Htot_def.included:= TRUE;
    END clear;
    METHOD seqmod;
      RUN state.specify;
```

```
      state.y[components].fixed:= FALSE;
      H.fixed                 := TRUE;
    END seqmod;


    METHOD scale;
      RUN state.scale;
      FOR i IN components DO
          f[i].nominal := f[i] + 0.1{mol/s};
          f[i].upper_bound := f[i] + scale*f[i].nominal;
      END;
      Ftot.nominal := Ftot + 0.1{mol/s};
      Ftot.upper_bound := Ftot + scale*Ftot.nominal;
      Htot.nominal := abs(Htot) + 1{kW};
      Htot.lower_bound := Htot - scale*Htot.nominal;
      Htot.upper_bound := Htot + scale*Htot.nominal;
      Ftot_scale := Ftot.nominal;
      Htot_scale := Htot.nominal;

    END scale;

END homogeneous_td_stream;

(* ***********************************+********************* *)
(* ********************      two_phase_molar_stream  ******* *)
(* ***********************************+********************* *)

MODEL two_phase_molar_stream(
      state WILL_BE two_phase_mixture;
);

      components ALIASES state.components;
      Ftot,
      f[components]IS_A molar_rate;
      scale             IS_A scaling_constant;
      Ftot_scale  IS_A real;


      FOR i IN components CREATE
          f_def[i]: f[i]/Ftot_scale = Ftot*state.y[i]/Ftot_scale;
      END;

      Ftot_scale := 1000 {mole/s};
(* horrible scaling method - global var, ack! *)

  METHODS

    METHOD clear;
      RUN state.clear;
      Ftot.fixed              := FALSE;
```

```
      f[components].fixed:= FALSE;
      f_def[components].included:= TRUE;
    END clear;

    METHOD seqmod;
      RUN state.specify;
      state.y[components].fixed:= FALSE;
    END seqmod;

    METHOD specify;
      RUN seqmod;
      f[components].fixed:= TRUE;
    END specify;

    METHOD reset;
      RUN clear;
      RUN specify;
    END reset;

    METHOD scale;
      RUN state.scale;
      FOR i IN components DO
          f[i].nominal := f[i] + 0.1{mol/s};
          f[i].upper_bound := f[i] + scale*f[i].nominal;
      END;
      Ftot.nominal := Ftot + 0.1{mol/s};
      Ftot.upper_bound := Ftot + scale*Ftot.nominal;
      Ftot_scale := Ftot.nominal;
    END scale;
END two_phase_molar_stream;

(* *********************************+********************** *)
(* ********************     thermodynamic_stream    ********* *)
(* *********************************+********************** *)

MODEL two_phase_td_stream(
      state WILL_BE two_phase_mixture;
) REFINES two_phase_molar_stream;

NOTES ‘usage’ SELF {
td_stream here does NOT denote that the state is a td_homo or td_hetero
mixture. It merely indicates that total H is computed so that certain
thermodynamic principles may be applied using this type of stream.
state.H may be an arbitrary number if the state does not compute
it from some correlation.
}
END NOTES;

      H                        ALIASES state.H;
      Htot                     IS_A energy_rate;
```

```
      Htot_scale        IS_A real;

      Htot_def: Htot/Htot_scale = H*Ftot/Htot_scale;

      Htot_scale := 1{MW};

  METHODS

    METHOD clear;
      RUN state.clear;
      Ftot.fixed        := FALSE;
      f[components].fixed:= FALSE;
      Htot.fixed        := FALSE;
      H.fixed                 := FALSE;
      f_def[components].included:= TRUE;
      Htot_def.included:= TRUE;
    END clear;
    METHOD seqmod;
      RUN state.specify;
      state.y[components].fixed:= FALSE;
      H.fixed                 := TRUE;
    END seqmod;


    METHOD scale;
      RUN state.scale;
      FOR i IN components DO
          f[i].nominal := f[i] + 0.1{mol/s};
          f[i].upper_bound := f[i] + scale*f[i].nominal;
      END;
      Ftot.nominal := Ftot + 0.1{mol/s};
      Ftot.upper_bound := Ftot + scale*Ftot.nominal;
      Htot.nominal := abs(Htot) + 1{kW};
      Htot.lower_bound := Htot - scale*Htot.nominal;
      Htot.upper_bound := Htot + scale*Htot.nominal;
      Ftot_scale := Ftot.nominal;
      Htot_scale := Htot.nominal;

    END scale;

END two_phase_td_stream;

(* ********************+********************************** *)
(* ******          vapor_stream         ******************** *)
(* ********************+********************************** *)

MODEL vapor_stream (
      state WILL_BE vapor_mixture;
) REFINES homogeneous_td_stream;
```

```
  METHODS

    METHOD seqmod;
      RUN state.specify;
      state.y[components].fixed:= FALSE;
    END seqmod;

END vapor_stream;

(* *********************+************************************ *)
(* ******         liquid_stream         ********************* *)
(* *********************+************************************ *)

MODEL liquid_stream(
      state WILL_BE liquid_mixture;
) REFINES homogeneous_td_stream;


  METHODS

    METHOD seqmod;
      RUN state.specify;
      state.y[components].fixed:= FALSE;
    END seqmod;

END liquid_stream;

(* *********************+************************************ *)
(* ******      multiphase_stream         ********************* *)
(* *********************+************************************ *)

MODEL multiphase_stream(
      state WILL_BE td_two_phase_mixture;
) REFINES two_phase_td_stream;

      phases ALIASES state.phases;
      phase_flow[phases]IS_A molar_rate;

      FOR k IN phases CREATE
          phase_flow_def[k]: phase_flow[k]/Ftot_scale =
            state.phi[k]*Ftot/Ftot_scale;
      END;

  METHODS

    METHOD clear;
      RUN state.clear;
      Ftot.fixed              := FALSE;
      f[components].fixed:= FALSE;
      Htot.fixed              := FALSE;
```

```
      phase_flow[phases].fixed:= FALSE;
      Htot_def.included := TRUE;
      f_def[components].included:= TRUE;
      phase_flow_def[phases].included:= TRUE;
    END clear;

    METHOD scale;
      RUN state.scale;
      FOR i IN components DO
          f[i].nominal := f[i] + 0.1{mol/s};
          f[i].upper_bound := f[i] + scale*f[i].nominal;
      END;
      Ftot.nominal := Ftot + 0.1{mol/s};
      Ftot.upper_bound := Ftot + scale*Ftot.nominal;
      Htot.nominal := abs(Htot) + 1{kW};
      Htot.lower_bound := Htot - scale*Htot.nominal;
      Htot.upper_bound := Htot + scale*Htot.nominal;

      FOR j IN phases DO
          phase_flow[j].nominal := phase_flow[j] + 0.1{mol/s};
          phase_flow[j].upper_bound :=
            phase_flow[j] + scale*phase_flow[j].nominal;
      END;
      Ftot_scale := Ftot.nominal;
      Htot_scale := Htot.nominal;
    END scale;

    METHOD seqmod;
      RUN state.specify;
      state.y[components].fixed:= FALSE;
    END seqmod;

END multiphase_stream;

MODEL vapor_liquid_stream(
      state WILL_BE td_VLE_mixture;
) REFINES multiphase_stream;
(* no body *)
END vapor_liquid_stream;
```

A.2.7 FLASH

This library was developed by Jennifer Perry in about three hours, record time for creating any large set of models in ASCEND. Ms. Perry was slightly familiar with a library of similar functionality in ASCEND III.

Appendix A Code 10
```
(* flash.lib *)
MODEL flash_base;
      Qin IS_A energy_rate;
NOTES
'usage' SELF {
Never create one of these directly.
This is just a common ancestor type.
}
END NOTES;
END flash_base;

MODEL vapor_liquid_flash(
      feed WILL_BE liquid_stream;
      vapout WILL_BE vapor_stream;
      liqout WILL_BE liquid_stream;
      flash_state WILL_BE td_VLE_mixture;
) WHERE (
      flash_state.heavy, liqout.state WILL_BE_THE_SAME;
      flash_state.light, vapout.state WILL_BE_THE_SAME;
      vapout.state.T, feed.state.T WILL_NOT_BE_THE_SAME;
      vapout.state.P, feed.state.P WILL_NOT_BE_THE_SAME;
      liqout.state.T, feed.state.T WILL_NOT_BE_THE_SAME;
      liqout.state.P, feed.state.P WILL_NOT_BE_THE_SAME;
      liqout.state.options.ds.components ==
      vapout.state.options.ds.components;
) REFINES flash_base;

      components ALIASES flash_state.components;
      H_scale IS_A real;
      F_scale IS_A real;


      FOR i IN components - [feed.state.options.ds.reference] CREATE
            flash_state.y[i] = feed.state.y[i];
      END;
      (vapout.Ftot - flash_state.phi['vapor'] * feed.Ftot)/F_scale =0;
      (liqout.Ftot - flash_state.phi['liquid'] * feed.Ftot)/F_scale
=0;


      EBal: (feed.Htot + Qin - (vapout.Htot + liqout.Htot))/H_scale
=0;
```

```
      H_scale := 1e+8{J/s};
      F_scale := 10{mole/s};

METHODS

METHOD clear;
      RUN feed.clear;
      RUN vapout.clear;
      RUN liqout.clear;
      RUN flash_state.clear;
      Qin.fixed := FALSE;
END clear;

METHOD seqmod;
      RUN flash_state.specify;
      flash_state.y[components].fixed := FALSE;
END seqmod;

METHOD reset_to_massbal;
      flash_state.equilibrated := FALSE;
      RUN reset;
END reset_to_massbal;

METHOD reset_to_fullthermo;
      flash_state.equilibrated := TRUE;
      RUN reset;
END reset_to_fullthermo;

METHOD specify;
      RUN seqmod;
      RUN feed.specify;
END specify;

METHOD reset;
      RUN clear;
      RUN specify;
END reset;

END vapor_liquid_flash;

MODEL tray(
      liqin WILL_BE liquid_stream;
      vapin WILL_BE vapor_stream;
      liqout WILL_BE liquid_stream;
      vapout WILL_BE vapor_stream;
      tray_state WILL_BE td_VLE_mixture;
) WHERE (
      tray_state.heavy, liqout.state WILL_BE_THE_SAME;
      tray_state.light, vapout.state WILL_BE_THE_SAME;
      vapout.state.T, vapin.state.T, liqin.state.T
```

```
            WILL_NOT_BE_THE_SAME;
      vapout.state.P, vapin.state.P, liqin.state.P
            WILL_NOT_BE_THE_SAME;
      liqin.state.options.ds.components ==
      liqout.state.options.ds.components;
      vapin.state.options.ds.components ==
      liqout.state.options.ds.components;
) REFINES flash_base;

      cmo_ratio IS_A factor;

      cmo: (cmo_ratio * liqin.Ftot - liqout.Ftot)/F_scale = 0;

      components ALIASES tray_state.components;
      H_scale IS_A real;
      F_scale IS_A real;
      scale IS_A scaling_constant;

      FOR i IN components - [vapin.state.options.ds.reference] CREATE
      (
            tray_state.y[i] * (liqin.Ftot + vapin.Ftot) -
            (vapin.state.y[i] * vapin.Ftot +
                  liqin.state.y[i] * liqin.Ftot)
      ) / F_scale = 0;
      END;

      (vapout.Ftot - tray_state.phi[‘vapor’] * (vapin.Ftot
+liqin.Ftot))
      / F_scale = 0;
      (liqout.Ftot - tray_state.phi[‘liquid’] *
            (vapin.Ftot + liqin.Ftot))
      / F_scale = 0;


      EBal: ((vapin.Htot+liqin.Htot) + Qin -
            (vapout.Htot + liqout.Htot))/ H_scale = 0;

      H_scale := 1e+8{J/s};
      F_scale := 10{mole/s};

      METHODS

METHOD clear;
      RUN vapin.clear;
      RUN liqin.clear;
      RUN vapout.clear;
      RUN liqout.clear;
      RUN tray_state.clear;
      cmo_ratio.fixed:= FALSE;
      Qin.fixed := FALSE;
```

```
END clear;

METHOD seqmod;
      RUN tray_state.specify;
      tray_state.y[components].fixed := FALSE;
      tray_state.phi['vapor'].fixed := FALSE;
      cmo_ratio.fixed:= TRUE;
END seqmod;

METHOD reset_to_massbal;
      tray_state.equilibrated := FALSE;
      RUN reset;
END reset_to_massbal;

METHOD reset_to_fullthermo;
      tray_state.equilibrated := TRUE;
      RUN reset;
END reset_to_fullthermo;

METHOD reset_to_adiabatic;
      RUN reset_to_fullthermo;
      cmo_ratio.fixed := FALSE;
      Qin.fixed := TRUE;
      Qin := 0{W};
END reset_to_adiabatic;

METHOD specify;
      RUN seqmod;
      RUN vapin.specify;
      RUN liqin.specify;
END specify;

METHOD reset;
      RUN clear;
      RUN specify;
END reset;

METHOD scale;
      RUN tray_state.scale;
      RUN liqin.scale;
      RUN vapin.scale;
      RUN liqout.scale;
      RUN vapout.scale;
      cmo_ratio.nominal := cmo_ratio;
      cmo_ratio.lower_bound := 0.0;
      cmo_ratio.upper_bound := cmo_ratio + scale*cmo_ratio.nominal;
END scale;

END tray;
```

```
MODEL feed_tray(
      feed WILL_BE vapor_liquid_stream;
      liqin WILL_BE liquid_stream;
      vapin WILL_BE vapor_stream;
      liqout WILL_BE liquid_stream;
      vapout WILL_BE vapor_stream;
      feed_tray_state WILL_BE td_VLE_mixture;
) WHERE (
      feed_tray_state.heavy, liqout.state WILL_BE_THE_SAME;
      feed_tray_state.light, vapout.state WILL_BE_THE_SAME;
      vapout.state.T, feed.state.T WILL_NOT_BE_THE_SAME;
      vapout.state.P, feed.state.P WILL_NOT_BE_THE_SAME;
      liqin.state.options.ds.components == feed.components;
      vapin.state.options.ds.components == feed.components;
      liqout.state.options.ds.components == feed.components;
      vapout.state.options.ds.components == feed.components;
) REFINES flash_base;

      q IS_A factor;

      q_eqn: (liqout.Ftot - (liqin.Ftot + q * feed.Ftot)) / F_scale
=0;
      q := 1.0;


      components ALIASES feed_tray_state.components;
      H_scale IS_A real;
      F_scale IS_A real;
      scale IS_A scaling_constant;

      FOR i IN components - [vapin.state.options.ds.reference] CREATE
      ( (feed_tray_state.y[i] *
            (feed.Ftot + liqin.Ftot + vapin.Ftot)) -
        (vapin.state.y[i]*vapin.Ftot + liqin.state.y[i]*liqin.Ftot +
            feed.state.y[i]*feed.Ftot)
      ) / F_scale = 0;
      END;
      (vapout.Ftot -
      feed_tray_state.phi['vapor'] *
(vapin.Ftot+liqin.Ftot+feed.Ftot))
      / F_scale = 0;
      (liqout.Ftot -
      feed_tray_state.phi['liquid'] *
(vapin.Ftot+liqin.Ftot+feed.Ftot))
      / F_scale = 0;


      EBal: ((vapin.Htot + liqin.Htot + feed.Htot) + Qin -
            (vapout.Htot + liqout.Htot)) / H_scale = 0;
      H_scale := 1e+8{J/s};
```

```
      F_scale := 10{mole/s};

METHODS

METHOD clear;
      RUN feed.clear;
       RUN vapin.clear;
      RUN liqin.clear;
      RUN vapout.clear;
      RUN liqout.clear;
      RUN feed_tray_state.clear;
      q.fixed:= FALSE;
      Qin.fixed := FALSE;
END clear;

METHOD seqmod;
      RUN feed_tray_state.specify;
      feed_tray_state.y[components].fixed := FALSE;
      feed_tray_state.phi['vapor'].fixed := FALSE;
      q.fixed:= TRUE;
END seqmod;

METHOD reset_to_massbal;
      feed_tray_state.equilibrated := FALSE;
      RUN reset;
END reset_to_massbal;

METHOD reset_to_fullthermo;
      feed_tray_state.equilibrated := TRUE;
      RUN reset;
END reset_to_fullthermo;

METHOD reset_to_adiabatic;
      RUN reset_to_fullthermo;
      q.fixed := FALSE;
      Qin.fixed := TRUE;
      Qin := 0{W};
END reset_to_adiabatic;

METHOD specify;
      RUN seqmod;
      RUN feed.specify;
      RUN vapin.specify;
      RUN liqin.specify;
END specify;

METHOD reset;
      RUN clear;
      RUN specify;
END reset;
```

```
METHOD scale;
      RUN feed_tray_state.scale;
      RUN feed.scale;
      RUN liqin.scale;
      RUN vapin.scale;
      RUN liqout.scale;
      RUN vapout.scale;
      q.nominal := abs(q);
      q.lower_bound := q - scale*q.nominal;
      q.upper_bound := q + scale*q.nominal;
END scale;
END feed_tray;


MODEL simple_condenser(
      vapin WILL_BE vapor_stream;
      liqout WILL_BE liquid_stream;
      distillate WILL_BE liquid_stream;
) WHERE (
      liqout.state, distillate.state WILL_BE_THE_SAME;
) REFINES flash_base;

      reflux_ratio      IS_A factor;

      reflux_def: (liqout.Ftot - reflux_ratio*distillate.Ftot)
      /F_scale=0;

      condenser_state ALIASES liqout.state;
      components ALIASES condenser_state.options.ds.components;
      H_scale IS_A real;
      F_scale IS_A real;
      scale IS_A scaling_constant;

      FOR i IN components - [vapin.state.options.ds.reference] CREATE
            condenser_state.y[i] = vapin.state.y[i];
      END;

      (vapin.Ftot - distillate.Ftot - liqout.Ftot) / F_scale = 0;


      EBal: (vapin.Htot + Qin - (liqout.Htot + distillate.Htot))
      /H_scale =0;
      H_scale := 1e+8{J/s};
      F_scale := 10{mole/s};

METHODS

METHOD clear;
      RUN vapin.clear;
```

```
      RUN distillate.clear;
      RUN liqout.clear;
      RUN condenser_state.clear;
      Qin.fixed := FALSE;
      reflux_ratio.fixed := FALSE;
END clear;

METHOD seqmod;
      RUN condenser_state.specify;
      condenser_state.y[components].fixed := FALSE;
      reflux_ratio.fixed := TRUE;
END seqmod;

METHOD specify;
      RUN seqmod;
      RUN vapin.specify;
END specify;

METHOD reset;
      RUN clear;
      RUN specify;
END reset;

METHOD scale;
      RUN condenser_state.scale;
      RUN vapin.scale;
      RUN liqout.scale;
      RUN distillate.scale;
END scale;

END simple_condenser;


MODEL simple_reboiler(
      liqin WILL_BE liquid_stream;
      vapout WILL_BE vapor_stream;
      bottoms WILL_BE liquid_stream;
      reboiler_state WILL_BE td_VLE_mixture;
) WHERE (
      reboiler_state.light, vapout.state WILL_BE_THE_SAME;
      reboiler_state.heavy, bottoms.state WILL_BE_THE_SAME;
        vapout.state.T, liqin.state.T WILL_NOT_BE_THE_SAME;
        vapout.state.P, liqin.state.P WILL_NOT_BE_THE_SAME;
        bottoms.state.T, liqin.state.T WILL_NOT_BE_THE_SAME;
        bottoms.state.P, liqin.state.P WILL_NOT_BE_THE_SAME;
        bottoms.state.options.ds.components ==
        vapout.state.options.ds.components;
) REFINES flash_base;

NOTES ‘ascii-picture’ SELF {
```

```
   |         ^
   v liqin   | vapout
 /-------------\
 |             |
 |~~~~~~~~~~~~~|
 \__OOO________+--> bottoms
     | |
Qin ^ .
}
END NOTES;

      reboil_ratio      IS_A factor;

      reboil_def: (vapout.Ftot - reboil_ratio * bottoms.Ftot )
      / F_scale = 0;

      components ALIASES reboiler_state.components;
      H_scale IS_A real;
      F_scale IS_A real;


      FOR i IN components - [liqin.state.options.ds.reference] CREATE
            reboiler_state.y[i] = liqin.state.y[i];
      END;

      (vapout.Ftot - reboiler_state.phi[‘vapor’] * liqin.Ftot)
      /F_scale = 0;
      (bottoms.Ftot - reboiler_state.phi[‘liquid’] * liqin.Ftot)
      /F_scale = 0;

      EBal: (liqin.Htot + Qin - (vapout.Htot + bottoms.Htot))
      /H_scale = 0;
      H_scale := 1e+8{J/s};
      F_scale := 10{mole/s};

METHODS

METHOD clear;
      RUN liqin.clear;
      RUN bottoms.clear;
      RUN vapout.clear;
      RUN reboiler_state.clear;
      Qin.fixed := FALSE;
      reboil_ratio.fixed := FALSE;
END clear;

METHOD seqmod;
      RUN reboiler_state.specify;
      reboiler_state.y[components].fixed := FALSE;
      reboiler_state.phi[‘vapor’].fixed := FALSE;
```

```
      reboil_ratio.fixed := TRUE;
END seqmod;

METHOD reset_to_massbal;
      reboiler_state.equilibrated := FALSE;
      RUN reset;
END reset_to_massbal;

METHOD reset_to_fullthermo;
      reboiler_state.equilibrated := TRUE;
      RUN reset;
END reset_to_fullthermo;

METHOD specify;
      RUN seqmod;
      RUN liqin.specify;
END specify;

METHOD reset;
      RUN clear;
      RUN specify;
END reset;

METHOD scale;
      RUN reboiler_state.scale;
      RUN liqin.scale;
      RUN vapout.scale;
      RUN bottoms.scale;
END scale;

END simple_reboiler;


MODEL test_vapor_liquid_flash;

(* This is written in the top down style of a novice modeler.
 * Expert models put the statements in reverse order, adding
 * new statements needed _above_ the old ones as they discover
 * from the parameter lists what else is needed to create
 * each object.
 * Of course order in ASCEND doesn't matter in terms of having
 * a correct MODEL, but it does affect compiler performance.
 * Defining arguments before passing them helps the compiler.
 *)
      Flash IS_A vapor_liquid_flash(Feed, Vapout, Liqout,
flash_state);

      Feed IS_A liquid_stream(feed_state);
      Vapout IS_A vapor_stream(vapor_state);
      Liqout IS_A liquid_stream(liquid_state);
```

```
      flash_state IS_A
      td_VLE_mixture(P, T,  vapor_state, liquid_state, equilibrated);

      P      IS_A pressure;
      T      IS_A temperature;
      equilibrated IS_A boolean;
      feed_state IS_A liquid_mixture(feed_P, feed_T, liquid_options);
      liquid_state IS_A liquid_mixture(P, T, liquid_options);
      vapor_state IS_A vapor_mixture(P, T, vapor_options);

      feed_PIS_A pressure;
      feed_T IS_A temperature;
      vapor_options IS_A vapor_phase_options(ds,’Pitzer’,’Pitzer’);
      liquid_options IS_A liquid_phase_options(ds,’Rackett’, ’UNI-
FAC’);

      ds    IS_A
      td_component_data_set([‘n_pentane’,’n_hexane’,’n_heptane’],
                              ’n_heptane’);

METHODS

METHOD clear;
      RUN Flash.clear;
END clear;

METHOD reset;
      RUN clear;
      RUN Flash.reset;
END reset;

METHOD values;
      flash_state.alpha[‘n_pentane’] := 5;
      flash_state.alpha[‘n_hexane’] := 2;
      flash_state.alpha[‘n_heptane’] := 1;
      T := 298 {K};
      P := 1{atm};
      Feed.f[‘n_pentane’] := 3{mole/s};
      Feed.f[‘n_hexane’] := 3{mole/s};
      Feed.f[‘n_heptane’] := 3{mole/s};
END values;

END test_vapor_liquid_flash;


MODEL test_hard_vapor_liquid_flash;
      (* This MODEL is bottom-up, as an expert would write *)
```

```
      ds    IS_A
      td_component_data_set([‘acetone’,’chloroform’,’benzene’],
                              ’benzene’);

      vapor_options IS_A vapor_phase_options(ds,’Pitzer’,’Pitzer’);
      liquid_options IS_A liquid_phase_options(ds,’Rackett’, ’UNI-
FAC’);

      feed_PIS_A pressure;
      feed_T IS_A temperature;
      P      IS_A pressure;
      T      IS_A temperature;

      equilibrated IS_A boolean;
      feed_state IS_A liquid_mixture(feed_P, feed_T, liquid_options);
      liquid_state IS_A liquid_mixture(P, T, liquid_options);
      vapor_state IS_A vapor_mixture(P, T, vapor_options);


      Feed IS_A liquid_stream(feed_state);
      Vapout IS_A vapor_stream(vapor_state);
      Liqout IS_A liquid_stream(liquid_state);
      flash_state IS_A
      td_VLE_mixture(P, T,  vapor_state, liquid_state, equilibrated);

      Flash IS_A vapor_liquid_flash(Feed, Vapout, Liqout,
flash_state);

METHODS

METHOD clear;
      RUN Flash.clear;
END clear;

METHOD reset;
      RUN clear;
      RUN Flash.reset;
END reset;

METHOD values;
      flash_state.alpha[‘acetone’] := 3;
      flash_state.alpha[‘chloroform’] := 2;
      flash_state.alpha[‘benzene’] := 1;
      T := 298 {K};
      P := 1{atm};
      Feed.f[‘acetone’,’chloroform’,’benzene’] := 3{mole/s};
END values;

END test_hard_vapor_liquid_flash;
```

```
MODEL test_tray;

      ds    IS_A
      td_component_data_set([‘n_pentane’,’n_hexane’,’n_heptane’],
                             ’n_heptane’);

      vapor_options IS_A vapor_phase_options(ds,’Pitzer’,’Pitzer’);
      liquid_options IS_A liquid_phase_options(ds,’Rackett’, ’UNI-
FAC’);

      P     IS_A pressure;
      T     IS_A temperature;
      vapin_PIS_A pressure;
      vapin_T IS_A temperature;
      liqin_P IS_A pressure;
      liqin_T IS_A temperature;

      vapin_state IS_A vapor_mixture(vapin_P, vapin_T,
vapor_options);
      liqin_state IS_A liquid_mixture(liqin_P, liqin_T,
liquid_options);
      liquid_state IS_A liquid_mixture(P, T, liquid_options);
      vapor_state IS_A vapor_mixture(P, T, vapor_options);

      Liqin IS_A liquid_stream(liqin_state);
      Vapin IS_A vapor_stream(vapin_state);
      Vapout IS_A vapor_stream(vapor_state);
      Liqout IS_A liquid_stream(liquid_state);

      equilibrated IS_A boolean;
      tray_state IS_A
      td_VLE_mixture(P, T,  vapor_state, liquid_state, equilibrated);

      Tray IS_A tray(Liqin, Vapin, Liqout, Vapout, tray_state);

METHODS

METHOD clear;
      RUN Tray.clear;
END clear;

METHOD reset;
      RUN Tray.reset;
END reset;

METHOD values;
      tray_state.alpha[‘n_pentane’] := 3;
      tray_state.alpha[‘n_hexane’] := 2;
      tray_state.alpha[‘n_heptane’] := 1;
```

```
      T := 298 {K};
      P := 1{atm};
      Liqin.f[‘n_pentane’] := 3{mole/s};
      Liqin.f[‘n_hexane’] := 3{mole/s};
      Liqin.f[‘n_heptane’] := 3{mole/s};
END values;

END test_tray;

MODEL test_feed_tray;
      ds    IS_A
      td_component_data_set([‘n_pentane’,’n_hexane’,’n_heptane’],
                              ’n_heptane’);

      vapor_options IS_A vapor_phase_options(ds,’Pitzer’,’Pitzer’);
      liquid_options IS_A liquid_phase_options(ds,’Rackett’, ’UNI-
FAC’);

      P     IS_A pressure;
      T     IS_A temperature;
      feedin_P IS_A pressure;
      feedin_T IS_A temperature;
      vapin_PIS_A pressure;
      vapin_T IS_A temperature;
      liqin_P IS_A pressure;
      liqin_T IS_A temperature;

      equilibrated IS_A boolean;
      feedin_state IS_A td_VLE_mixture(
            feedin_P, feedin_T, feed_vapor_state, feed_liquid_state,
            equilibrated);

      feed_vapor_state
      IS_A vapor_mixture(feedin_P, feedin_T, vapor_options);

      feed_liquid_state
      IS_A liquid_mixture(feedin_P, feedin_T, liquid_options);

      vapin_state IS_A vapor_mixture(vapin_P, vapin_T,
vapor_options);
      liqin_state IS_A liquid_mixture(liqin_P, liqin_T,
liquid_options);
      liquid_state IS_A liquid_mixture(P, T, liquid_options);
      vapor_state IS_A vapor_mixture(P, T, vapor_options);

      Feedin IS_A vapor_liquid_stream(feedin_state);
      Liqin IS_A liquid_stream(liqin_state);
      Vapin IS_A vapor_stream(vapin_state);
      Vapout IS_A vapor_stream(vapor_state);
      Liqout IS_A liquid_stream(liquid_state);
```

```
      feed_tray_state IS_A
      td_VLE_mixture(P, T,  vapor_state, liquid_state, equilibrated);

      Feed_tray IS_A
      feed_tray(Feedin, Liqin, Vapin, Liqout, Vapout, feed_tray_state);
METHODS

METHOD clear;
      RUN Feed_tray.clear;
END clear;

METHOD reset;
      RUN Feed_tray.reset;
END reset;

METHOD values;
      feed_tray_state.alpha[‘n_pentane’] := 3;
      feed_tray_state.alpha[‘n_hexane’] := 2;
      feed_tray_state.alpha[‘n_heptane’] := 1;
      T := 298 {K};
      P := 1{atm};
      Liqin.f[‘n_pentane’] := 3{mole/s};
      Liqin.f[‘n_hexane’] := 3{mole/s};
      Liqin.f[‘n_heptane’] := 3{mole/s};
END values;

END test_feed_tray;

MODEL test_condenser;
      ds    IS_A
      td_component_data_set([‘n_pentane’,’n_hexane’,’n_heptane’],
                              ’n_heptane’);

      vapor_options IS_A vapor_phase_options(ds,’Pitzer’,’Pitzer’);
      liquid_options IS_A liquid_phase_options(ds,’Rackett’, ’UNIFAC’);

      P      IS_A pressure;
      T      IS_A temperature;
      vapin_P      IS_A pressure;
      vapin_T      IS_A temperature;
      liquid_state IS_A liquid_mixture(P, T, liquid_options);
      vapin_state IS_A vapor_mixture(vapin_P, vapin_T, vapor_options);

      Vapin IS_A vapor_stream(vapin_state);
      Distillate IS_A liquid_stream(liquid_state);
      Liqout IS_A liquid_stream(liquid_state);
```

```
      Condenser IS_A
      simple_condenser(Vapin, Liqout, Distillate);

METHODS

METHOD clear;
      RUN Condenser.clear;
END clear;

METHOD reset;
      RUN Condenser.reset;
END reset;

METHOD values;
      T := 298 {K};
      P := 1{atm};
      Vapin.f[‘n_pentane’] := 3{mole/s};
      Vapin.f[‘n_hexane’] := 3{mole/s};
      Vapin.f[‘n_heptane’] := 3{mole/s};
END values;

END test_condenser;


MODEL test_reboiler;

      ds    IS_A
      td_component_data_set([‘n_pentane’,’n_hexane’,’n_heptane’],
                              ’n_heptane’);

      vapor_options IS_A vapor_phase_options(ds,’Pitzer’,’Pitzer’);
      liquid_options IS_A liquid_phase_options(ds,’Rackett’, ’UNI-
FAC’);

      P      IS_A pressure;
      T      IS_A temperature;
      liqin_PIS_A pressure;
      liqin_T IS_A temperature;

      equilibrated IS_A boolean;
      liqin_state IS_A liquid_mixture(liqin_P, liqin_T,
liquid_options);
      liquid_state IS_A liquid_mixture(P, T, liquid_options);
      vapor_state IS_A vapor_mixture(P, T, vapor_options);

      Vapout IS_A vapor_stream(vapor_state);
      Bottoms IS_A liquid_stream(liquid_state);
      Liqin IS_A liquid_stream(liqin_state);
      reboiler_state IS_A
      td_VLE_mixture(P, T,  vapor_state, liquid_state, equilibrated);
```

```
      Reboiler IS_A simple_reboiler(Liqin, Vapout,
                              Bottoms, reboiler_state);

METHODS

METHOD clear;
      RUN Reboiler.clear;
END clear;

METHOD reset;
      RUN clear;
      RUN Reboiler.reset;
END reset;

METHOD values;
      reboiler_state.alpha[‘n_pentane’] := 3;
      reboiler_state.alpha[‘n_hexane’] := 2;
      reboiler_state.alpha[‘n_heptane’] := 1;
      T := 298 {K};
      P := 1{atm};
      Liqin.f[‘n_pentane’] := 3{mole/s};
      Liqin.f[‘n_hexane’] := 3{mole/s};
      Liqin.f[‘n_heptane’] := 3{mole/s};
END values;

END test_reboiler;
```

A.2.8 COLUMN

We developed and tested this library in about 10 hours, given a debugged flash.lib.

Appendix A Code 11
```
(*distillation column library *)
MODEL tray_stack(
      n_trays WILL_BE integer_constant;
      vapout WILL_BE vapor_stream;
      liqin WILL_BE liquid_stream;
      vapin WILL_BE vapor_stream;
      liqout WILL_BE liquid_stream;
      reduce WILL_BE fraction;
      equilibrated WILL_BE boolean;
) WHERE (
      n_trays > 1;
      vapout, liqin, vapin, liqout WILL_NOT_BE_THE_SAME;
      vapout.state.options, vapin.state.options WILL_BE_THE_SAME;
      liqout.state.options, liqin.state.options WILL_BE_THE_SAME;
);

NOTES 'ascii-picture'
SELF {
vapout ^ |
       | v liqin
      +---+
      |_1_|
      |___| n_trays (at least 2)
      |___|
      |_n_|
 vapin ^ |
       | v liqout
}
'usage'
equilbrated {
When TRUE, enforces equilbrium between phases on all the trays.
When FALSE, uses constant alpha between phases on all the trays.
}
reduce {
Homotopy parameter used by reduce_Q to move fixed errors in the
energy balances (Qin per tray) toward 0.
}
END NOTES;

      liquid_options ALIASES liqout.state.options;
      vapor_options ALIASES vapout.state.options;
      internal_tray_P[2..n_trays-1] IS_A pressure;
      internal_tray_T[2..n_trays-1] IS_A temperature;

(*
```

```
* The ALIASES/IS_A break if the liqout,vapout are on the same tray.
*)
      tray_P[tsP] ALIASES
            (vapout.state.P, internal_tray_P[2..n_trays-1],
                  liqout.state.P)
            WHERE tsP IS_A set OF integer_constant
            WITH_VALUE (1..n_trays);
      tray_T[tsT] ALIASES
            (vapout.state.T, internal_tray_T[2..n_trays-1],
                  liqout.state.T)
            WHERE tsT IS_A set OF integer_constant
            WITH_VALUE (1..n_trays);

      FOR i IN [1..n_trays-1] CREATE
            internal_liquid_state[i] IS_A liquid_mixture(
                  tray_P[i],
                  tray_T[i],
                  liquid_options
            );
        END;
      FOR i IN [2..n_trays] CREATE
            internal_vapor_state[i] IS_A vapor_mixture(
                  tray_P[i],
                  tray_T[i],
                  vapor_options
            );
        END;

      tray_liquid_state[tsL] ALIASES
            (internal_liquid_state[1..n_trays-1], liqout.state)
            WHERE tsL IS_A set OF integer_constant
            WITH_VALUE (1..n_trays);
      tray_vapor_state[tsV] ALIASES
            (vapout.state,internal_vapor_state[2..n_trays])
            WHERE tsV IS_A set OF integer_constant
            WITH_VALUE (1..n_trays);

      FOR i IN [1..n_trays] CREATE
            tray_state[i] IS_A td_VLE_mixture(
                  tray_P[i],
                  tray_T[i],
                  tray_vapor_state[i],
                  tray_liquid_state[i],
                  equilibrated
            );
      END;
      (* now like P,T, need to set up
       * tray_liqin[1..n_trays+1], tray_vapin[0..n_trays],
         * input streams.
       *)
```

```
      FOR i IN [2..n_trays] CREATE
            internal_vapout[i] IS_A vapor_stream(
                  internal_vapor_state[i]
            );
      END;
      tray_vapin[tsIV] ALIASES
            (vapout, internal_vapout[2..n_trays], vapin)
            WHERE tsIV IS_A set OF integer_constant
            WITH_VALUE (0..n_trays);

      FOR i IN [1..n_trays-1] CREATE
            internal_liqout[i] IS_A liquid_stream(
                  internal_liquid_state[i]
            );
      END;
      tray_liqin[tsIL] ALIASES
            (liqin, internal_liqout[1..n_trays-1], liqout)
            WHERE tsIL IS_A set OF integer_constant
            WITH_VALUE (1..n_trays+1);

      Tray_set IS_A set OF integer_constant;
(* TRUE but not necessary: Tray_set, tsL, tsV, tsP, tsT ARE_THE_SAME;
*)
      Tray_set :== tsL;

      FOR i IN Tray_set CREATE
            Stack[i] IS_A tray(
                  tray_liqin[i],
                  tray_vapin[i],
                  tray_liqin[i+1],
                  tray_vapin[i-1],
                  tray_state[i]
            );
      END;

METHODS

METHOD clear;
      RUN Stack[Tray_set].clear;
END clear;

METHOD specify;
      RUN seqmod;
      RUN liqin.specify;
      RUN vapin.specify;
END specify;

METHOD seqmod;
      RUN Stack[Tray_set].seqmod;
END seqmod;
```

```
METHOD reset;
      RUN clear;
      RUN specify;
END reset;

METHOD scale;
      RUN Stack[Tray_set].scale;
END scale;

METHOD reset_to_massbal;
      equilibrated := FALSE;
      RUN reset;
END reset_to_massbal;

METHOD reset_to_fullthermo;
      equilibrated := TRUE;
      RUN reset;
END reset_to_fullthermo;

METHOD reset_to_adiabatic;
      RUN reset_to_fullthermo;
      Stack[Tray_set].cmo_ratio.fixed := FALSE;
      Stack[Tray_set].Qin.fixed := TRUE;
      Stack[Tray_set].Qin := 0{W};
END reset_to_adiabatic;

METHOD reduce_Q;
      FOR j IN [Tray_set] DO
            Stack[j].Qin := reduce * Stack[j].Qin;
      END;
END reduce_Q;

END tray_stack;


MODEL test_tray_stack;
      n_trays_belowIS_A integer_constant;
      n_trays_below :== 2;
      equilibratedIS_A  boolean;
      reduce            IS_A  fraction;

NOTES
‘usage’ equilibrated {
This variable and feed.equilibrated may or may not be the same.
Do not assume they are when writing methods.
}
reduce {
This is a fraction from 0 to 1. The method reduce_Q uses it
to move the Qin on each stack and feed tray toward 0.
```

```
Essentially this is a physical homotopy parameter.
}
END NOTES;

      ds    IS_A td_component_data_set(
            [‘n_pentane’,’n_hexane’,’n_heptane’],
            ‘n_heptane’
      );
      vapor_options IS_A vapor_phase_options(ds,’Pitzer’,’Pitzer’);
      liquid_options IS_A liquid_phase_options(ds,’Rackett’,’UNI-
FAC’);

      vapout_P, liqin_P, vapin_P, liqout_PIS_A pressure;

      vapout_T, liqin_T, vapin_T, liqout_TIS_A temperature;

      vapout_state IS_A vapor_mixture(vapout_P,vapout_T,
vapor_options);
      liqin_state IS_A liquid_mixture(liqin_P, liqin_T,
liquid_options);
      vapin_state IS_A vapor_mixture(vapin_P, vapin_T,
vapor_options);
      liqout_state IS_A liquid_mixture(liqout_P, liqout_T,
                                    liquid_options);

      Vapout      IS_A vapor_stream(vapout_state);
      Liqin       IS_A liquid_stream(liqin_state);
      Vapin       IS_A vapor_stream(vapin_state);
      Liqout      IS_A liquid_stream(liqout_state);

      Section IS_A tray_stack(
            n_trays_below,
            Vapout,
            Liqin,
            Vapin,
            Liqout,
            reduce,
            equilibrated
      );

METHODS

METHOD reset;
      RUN clear;
      RUN specify;
END reset;

METHOD clear;
      RUN Section.clear;
END clear;
```

```
METHOD specify;
      RUN Section.specify;
END specify;

METHOD reset_to_adiabatic;
      RUN Section.reset_to_adiabatic;
END reset_to_adiabatic;

METHOD scale;
      RUN section.scale;
END scale;
END test_tray_stack;


MODEL simple_column(
      distillate  WILL_BE liquid_stream;
      n_trays_aboveWILL_BE integer_constant;
      feed              WILL_BE vapor_liquid_stream;
      n_trays_belowWILL_BE integer_constant;
      bottoms           WILL_BE liquid_stream;
      equilibratedWILL_BE boolean;
      reduce            WILL_BE fraction;
) WHERE (
      n_trays_above > 1;
      n_trays_below > 1;
      distillate, bottoms, feed WILL_NOT_BE_THE_SAME;
      bottoms.state.options, distillate.state.options,
      feed.state.heavy.options WILL_BE_THE_SAME;
);

      liquid_options ALIASES feed.state.heavy.options;
      vapor_options ALIASES feed.state.light.options;

NOTES
'ascii-picture' SELF {
                 ___   total condenser
              /   \________> distillate liquid
              |___|
(condensor_vapin)^ |
                 | |
                 | v(rectifier_liqin)
                +---+
                |___|
                |___| n_trays_above (at least 2)
                |___|
                |___|
(rectifier_vapin)^ |
                 | |
                 | v(feed_tray_liqin)
```

```
                +---+
 V/L feed ----->|   |
              +___+
(feed_tray_vapin)^ |
                 | |
                 | v(stripper_liqin)
                +---+
                |___|
                |___| n_trays_below (at least 2)
                |___|
                |___|
 (stripper_vapin)^ |
                 | |
                 | v(reboiler_liqin)
                 ___
                  |   |________> bottoms liquid
                  \___/ non-adiabatic flash
}
'usage' equilibrated {
This variable and feed.equilibrated may or may not be the same.
Do not assume they are when writing methods.
}
reduce {
This is a fraction from 0 to 1. The method reduce_Q uses it
to move the Qin on each stack and feed tray toward 0.
Essentially this is a physical homotopy parameter.
}
'changing' SELF {
This MODEL is very large, therefore it is very carefully
structured following the ascii-picture.
It should be relatively easy to modify this MODEL
just by carefully following the patterns we use. The pattern is
that all streams are given names corresponding to their input
roles and then all the subordinate P, T, state parts are named
by suffixing the corresponding stream names.  When a part is
in common with the output streams, we create an internal name
following the regular pattern to keep the symmetries clear.
Note that the symmetries are preserved even in this aliasing.
}
END NOTES;

(*
 * The creation steps:
 *    alias the options from the feed stream. (done above).
 *    inter-section P,T, states (single phase)
 *    inter-section streams (single phase)
 *    reboiler and feed_tray states (two phase)
 *    column sections
 * At the end of the steps, everything has been wired up
 * simply by passing common parts to both places they belong.
```

```
 * For clarity, no arrays are used.
 *
 * If you are nuts enough to reimplement this MODEL using
 * arrays, have a look at tray_stack for techniques to avoid
 * sending the compiler to hell.
 *
 * The thermodynamic options of the feed propagate to the whole
 * column without using ARE_ALIKE or ARE_THE_SAME.
 *)
(* thermodynamic P,T and state definitions for inter-section streams
*)

      feed_tray_P IS_A pressure;
      feed_tray_T IS_A temperature;

      condenser_vapin_PIS_A pressure;
      rectifier_liqin_PALIASES distillate.state.P;
      rectifier_vapin_PALIASES feed_tray_P;
      feed_tray_liqin_PIS_A pressure;
      feed_tray_vapin_PIS_A pressure;
      stripper_liqin_PALIASES feed_tray_P;
      stripper_vapin_PALIASES bottoms.state.P;
      reboiler_liqin_PIS_A pressure;

      condenser_vapin_TIS_A temperature;
      rectifier_liqin_TALIASES distillate.state.T;
      rectifier_vapin_TALIASES feed_tray_T;
      feed_tray_liqin_TIS_A temperature;
      feed_tray_vapin_TIS_A temperature;
      stripper_liqin_TALIASES feed_tray_T;
      stripper_vapin_TALIASES bottoms.state.T;
      reboiler_liqin_TIS_A temperature;

      condenser_vapin_state IS_A vapor_mixture(
            condenser_vapin_P,
            condenser_vapin_T,
            vapor_options
      );
      rectifier_liqin_state ALIASES distillate.state;
      rectifier_vapin_state IS_A vapor_mixture(
            rectifier_vapin_P,
            rectifier_vapin_T,
            vapor_options
      );
      feed_tray_liqin_state IS_A liquid_mixture(
            feed_tray_liqin_P,
            feed_tray_liqin_T,
            liquid_options
      );
      feed_tray_vapin_state IS_A vapor_mixture(
```

```
            feed_tray_vapin_P,
            feed_tray_vapin_T,
            vapor_options
      );
      stripper_liqin_state IS_A liquid_mixture(
            stripper_liqin_P,
            stripper_liqin_T,
            liquid_options
      );
      stripper_vapin_state IS_A vapor_mixture(
            stripper_vapin_P,
            stripper_vapin_T,
            vapor_options
      );
      reboiler_liqin_state IS_A liquid_mixture(
            reboiler_liqin_P,
            reboiler_liqin_T,
            liquid_options
      );

(* thermodynamic state definitions for reboiler, feed_tray *)
      feed_tray_stateIS_A td_VLE_mixture(
            feed_tray_P,
            feed_tray_T,
            rectifier_vapin_state,
            stripper_liqin_state,
            equilibrated
      );

      reboiler_state IS_A td_VLE_mixture(
            stripper_vapin_P,
            stripper_vapin_T,
            stripper_vapin_state,
            bottoms.state,
            equilibrated
      );

(* inter-section streams *)
      condenser_vapinIS_A vapor_stream(
            condenser_vapin_state
      );
      rectifier_liqinIS_A liquid_stream(
            rectifier_liqin_state
      );
      rectifier_vapinIS_A vapor_stream(
            rectifier_vapin_state
      );
      feed_tray_liqinIS_A liquid_stream(
            feed_tray_liqin_state
      );
```

```
      feed_tray_vapinIS_A vapor_stream(
            feed_tray_vapin_state
      );
      stripper_liqinIS_A liquid_stream(
            stripper_liqin_state
      );
      stripper_vapinIS_A vapor_stream(
            stripper_vapin_state
      );
      reboiler_liqinIS_A liquid_stream(
            reboiler_liqin_state
      );

(* column sections *)
      condenser IS_A simple_condenser(
            condenser_vapin,
            rectifier_liqin,
            distillate
      );
      rectifying_section IS_A tray_stack(
            n_trays_above,
            condenser_vapin,
            rectifier_liqin,
            rectifier_vapin,
            feed_tray_liqin,
            reduce,
            equilibrated
      );
      feed_tray IS_A feed_tray(
            feed,
            feed_tray_liqin,
            feed_tray_vapin,
            stripper_liqin,
            rectifier_vapin,
            feed_tray_state
      );
      stripping_section IS_A tray_stack(
            n_trays_below,
            feed_tray_vapin,
            stripper_liqin,
            stripper_vapin,
            reboiler_liqin,
            reduce,
            equilibrated
      );
      reboiler IS_A simple_reboiler(
            reboiler_liqin,
            stripper_vapin,
            bottoms,
            reboiler_state
```

```
      );

(* tray by tray profiles commonly desired *)
      N_trays IS_A integer_constant;
      N_trays :== n_trays_above + 1 + n_trays_below + 1;
      VLE_set IS_A set OF integer_constant;
      VLE_set :== [1 .. N_trays];
      Feed_loc IS_A integer_constant;
      Feed_loc :== n_trays_above +1;

      Tray[zTr] ALIASES (
            condenser,
            rectifying_section.Stack[1..n_trays_above],
            feed_tray,
            stripping_section.Stack[1..n_trays_below],
            reboiler)
      WHERE zTr IS_A set OF integer_constant
      WITH_VALUE (0..N_trays);

      Profile_traynum[VLE_set] IS_A integer_constant;
      FOR i IN VLE_set CREATE
            Profile_traynum[i] :== i;
      END;

      Profile_P[zP] ALIASES (
            rectifying_section.tray_P[1..n_trays_above],
            feed_tray_P,
            stripping_section.tray_P[1..n_trays_below],
            bottoms.state.P)
      WHERE zP IS_A set OF integer_constant WITH_VALUE (VLE_set);

      Profile_cmo_ratio[zc] ALIASES (
            rectifying_section.Stack[1..n_trays_above].cmo_ratio,
            stripping_section.Stack[1..n_trays_below].cmo_ratio)
      WHERE zc IS_A set OF integer_constant
      WITH_VALUE (1 .. n_trays_above,
                  Feed_loc+1 .. Feed_loc+n_trays_below);

      Profile_T[zT] ALIASES (
            rectifying_section.tray_T[1..n_trays_above],
            feed_tray_T,
            stripping_section.tray_T[1..n_trays_below],
            bottoms.state.T)
      WHERE zT IS_A set OF integer_constant WITH_VALUE (VLE_set);

      Profile_Qin[zQ] ALIASES (Tray[0..N_trays].Qin)
      WHERE zQ IS_A set OF integer_constant WITH_VALUE (0..N_trays);

      Profile_Lin[zL] ALIASES (
            rectifying_section.tray_liqin[1..n_trays_above].Ftot,
```

```
            feed_tray.liqin.Ftot,
            stripping_section.tray_liqin[1..n_trays_below].Ftot,
            reboiler.liqin.Ftot)
      WHERE zL IS_A set OF integer_constant WITH_VALUE (VLE_set);

      Profile_Vin[zV] ALIASES (
            rectifying_section.tray_vapin[0..n_trays_above].Ftot,
            feed_tray.vapin.Ftot,
            stripping_section.tray_vapin[1..n_trays_below].Ftot)
      WHERE zV IS_A set OF integer_constant WITH_VALUE (0..N_trays-
1);

      components ALIASES feed.state.light.options.ds.components;
      reference ALIASES feed.state.light.options.ds.reference;

      FOR i IN components CREATE

      Profile_x[i][zx[i]] ALIASES (

rectifying_section.Stack[1..n_trays_above].tray_state.heavy.y[i],
      feed_tray_state.heavy.y[i],

stripping_section.Stack[1..n_trays_below].tray_state.heavy.y[i],
      bottoms.state.y[i])
      WHERE zx[i] IS_A set OF integer_constant WITH_VALUE (VLE_set);

      Profile_y[i][zy[i]] ALIASES (

rectifying_section.Stack[1..n_trays_above].tray_state.light.y[i],
      feed_tray_state.light.y[i],

stripping_section.Stack[1..n_trays_below].tray_state.light.y[i],
      reboiler_state.light.y[i])
      WHERE zy[i] IS_A set OF integer_constant WITH_VALUE (VLE_set);

      Profile_kvalues_when_full_thermo[i][zk[i]] ALIASES (
      rectifying_section.Stack[1..n_trays_above].tray_state.alpha[i],
      feed_tray_state.alpha[i],
      stripping_section.Stack[1..n_trays_below].tray_state.alpha[i],
      reboiler_state.alpha[i])
      WHERE zk[i] IS_A set OF integer_constant WITH_VALUE (VLE_set);

      END;

(*
      VLE_set,
      zx[components], zy[components], zk[components],
      zT, zP ARE_THE_SAME; TRUE but not needed *)
(*    zTr, zQ ARE_THE_SAME; TRUE but not needed *)
```

```
      omb_slack[components]IS_A molar_rate;
      FOR i IN components CREATE
      OMB[i]: feed.f[i] = (distillate.f[i] + bottoms.f[i] +
                              omb_slack[i]);
        END;
      omb_slack[components] := 0{mol/s};
      omb_slack[components].lower_bound := -1000{mole/s};

METHODS

METHOD clear;
      RUN Tray[zTr].clear;
      reduce.fixed:= FALSE;
      omb_slack[components].fixed := FALSE;
      OMB[components].included:= TRUE;
END clear;

METHOD overall_mb;
(* is this method still useful? *)
(*    tray[Feed_loc].totfeedflow[components].included := FALSE; *)
      omb_slack[components] := 0.0{mol/s};
      omb_slack[components].fixed := TRUE;
END overall_mb;

METHOD propagate_feed;
      (* propagate feed tray flows and relative volatilities,
       * presumably after solving the feed tray.
       *)
      FOR i IN components DO
            rectifying_section.tray_state[1..n_trays_above].alpha[i]
:=
                  feed_tray_state.alpha[i];
            stripping_section.tray_state[1..n_trays_below].alpha[i]
:=
                  feed_tray_state.alpha[i];
            reboiler_state.alpha[i] := feed_tray_state.alpha[i];
            (* copy feed flow rates to all internal streams *)
            Tray[0..N_trays-1].liqout.f[i] := feed.f[i];
            Tray[1..N_trays].vapout.f[i] := feed.f[i];
            (* This copying should probably factor in the RR and BR *)
      END;
END propagate_feed;

METHOD total_reflux;
(* this method is BROKEN in both this library and Bob's *)
condenser.reflux_ratio.fixed := TRUE;
condenser.reflux_def.included := FALSE;
(*
Tray[0].totprod.Ftot := 0.0{mol/s};
Tray[0].totprod.Ftot.fixed := FALSE;
```

```
*)
feed.f[components].fixed := FALSE;
feed_tray.qeq.included := FALSE;
feed.Ftot := 0.0{mol/s};
feed.Ftot.fixed := TRUE;
feed_tray_state.y[components - [reference]].fixed := TRUE;
feed_tray_state.phi[‘liquid’].fixed := TRUE;
feed_tray_state.phi[‘liquid’] := 0.5;
RUN feed_tray.liqout.specify;
reboiler.reboil_ratio.fixed := TRUE;
reboiler.reboil_def.included := FALSE;
END total_reflux;

METHOD scale;
      RUN condenser.scale;
      RUN rectifying_section.scale;
      RUN feed_tray.scale;
      RUN stripping_section.scale;
      RUN reboiler.scale;
END scale;

METHOD seqmod;
      RUN condenser.seqmod;
      RUN rectifying_section.seqmod;
      RUN feed_tray.seqmod;
      RUN stripping_section.seqmod;
      RUN reboiler.seqmod;
END seqmod;

METHOD specify;
      RUN seqmod;
      RUN feed.specify;
END specify;

METHOD reset;
      RUN clear;
      RUN specify;
END reset;

METHOD reset_to_massbal;
      equilibrated := FALSE;
      feed.state.equilibrated := FALSE;
      RUN reset;
END reset_to_massbal;

METHOD reset_to_fullthermo;
      equilibrated := TRUE;
      feed.state.equilibrated := TRUE;
      RUN reset;
END reset_to_fullthermo;
```

```
METHOD reset_to_adiabatic;
      RUN reset_to_fullthermo;
      (* condenser, reboiler Qin left free *)
      rectifying_section.Stack[1..n_trays_above].cmo_ratio.fixed :=
            FALSE;
      stripping_section.Stack[1..n_trays_below].cmo_ratio.fixed :=
            FALSE;
      Tray[1..N_trays-1].Qin.fixed := TRUE;
      Tray[1..N_trays-1].Qin := 0{W};
      feed_tray.q.fixed := FALSE;
END reset_to_adiabatic;

END simple_column;


MODEL test_simple_column;

      ds    IS_A td_component_data_set(
            [‘n_pentane’,’n_hexane’,’n_heptane’],
            ‘n_heptane’
      );
      vapor_options IS_A vapor_phase_options(ds,’Pitzer’,’Pitzer’);
      liquid_options IS_A liquid_phase_options(ds,’Rackett’,’UNI-
FAC’);

      feed_P IS_A pressure;
      feed_T IS_A temperature;
      feed_vapor_state IS_A vapor_mixture(
            feed_P,
            feed_T,
            vapor_options
      );
      feed_liquid_state IS_A liquid_mixture(
            feed_P,
            feed_T,
            liquid_options
      );
      Equilibrated IS_A boolean;
      feed_state IS_A td_VLE_mixture(
            feed_P,
            feed_T,
            feed_vapor_state,
            feed_liquid_state,
            Equilibrated
      );
      Feed        IS_A vapor_liquid_stream(feed_state);

      distillate_PIS_A pressure;
      distillate_T IS_A temperature;
```

```
	distillate_state IS_A liquid_mixture(
		distillate_P,
		distillate_T,
		liquid_options
	);
	DistillateIS_A liquid_stream(distillate_state);

	bottoms_PIS_A pressure;
	bottoms_T IS_A temperature;
	bottoms_state IS_A liquid_mixture(
		bottoms_P,
		bottoms_T,
		liquid_options
	);
	Bottoms     IS_A liquid_stream(bottoms_state);

	n_trays_above, n_trays_below IS_A integer_constant;
	n_trays_above :== 6;
	n_trays_below :== 5;
	reduce IS_A fraction;
	Column IS_A simple_column(
		Distillate,
		n_trays_above,
		Feed,
		n_trays_below,
		Bottoms,
		Equilibrated,
		reduce
	);

METHODS
METHOD clear;
	RUN feed.clear;
	RUN Column.clear;
	Equilibrated := TRUE;
END clear;

METHOD values;
	Column.feed_tray_state.alpha['n_pentane'] := 3;
	Column.feed_tray_state.alpha['n_hexane'] := 2;
	Column.feed_tray_state.alpha['n_heptane'] := 1;
	Column.condenser.reflux_ratio := 1.3;
	feed_T := 298 {K};
	feed_P := 1{atm};
	Feed.f['n_pentane'] := 3{mole/s};
	Feed.f['n_hexane'] := 3{mole/s};
	Feed.f['n_heptane'] := 3{mole/s};
	(* here we should SOLVE the feed tray *)
	RUN Column.propagate_feed;
END values;
```

```
METHOD scale;
      RUN Column.scale;
END scale;
END test_simple_column;


MODEL demo_column(
      components IS_A set OF symbol_constant;
      reference IS_A symbol_constant;
      n_trays IS_A integer_constant;
      feed_location IS_A integer_constant;
) WHERE (
      reference IN components == TRUE;
      n_trays > 5;
      feed_location > 2;
      feed_location < n_trays - 2;
);

      ds    IS_A td_component_data_set(components,reference);
      vapor_options IS_A vapor_phase_options(ds,’Pitzer’,’Pitzer’);
      liquid_options IS_A liquid_phase_options(ds,’Rackett’,’UNI-
FAC’);

      feed_P IS_A pressure;
      feed_T IS_A temperature;
      feed_vapor_state IS_A vapor_mixture(
            feed_P,
            feed_T,
            vapor_options
      );
      feed_liquid_state IS_A liquid_mixture(
            feed_P,
            feed_T,
            liquid_options
      );
      Equilibrated IS_A boolean;
      feed_state IS_A td_VLE_mixture(
            feed_P,
            feed_T,
            feed_vapor_state,
            feed_liquid_state,
            Equilibrated
      );
      Feed        IS_A vapor_liquid_stream(feed_state);

      distillate_PIS_A pressure;
      distillate_T IS_A temperature;
      distillate_state IS_A liquid_mixture(
            distillate_P,
```

```
            distillate_T,
            liquid_options
      );
      DistillateIS_A liquid_stream(distillate_state);

      bottoms_PIS_A pressure;
      bottoms_T IS_A temperature;
      bottoms_state IS_A liquid_mixture(
            bottoms_P,
            bottoms_T,
            liquid_options
      );
      Bottoms     IS_A liquid_stream(bottoms_state);

      n_trays_above, n_trays_below IS_A integer_constant;
      n_trays_above :== feed_location - 1;
      n_trays_below :== n_trays - feed_location - 1;
      reduce IS_A fraction;
      Column IS_A simple_column(
            Distillate,
            n_trays_above,
            Feed,
            n_trays_below,
            Bottoms,
            Equilibrated,
            reduce
      );

(* component names in order of boiling point. useful for methods. *)
      z_boiling_comp[1..CARD[components]] IS_A symbol;
      z_bc IS_A symbol;
      z_bi IS_A integer;
METHODS
METHOD clear;
      RUN feed.clear;
      RUN Column.clear;
      Equilibrated := TRUE;
END clear;

METHOD values;
      (* The demo user may very well want to rewrite this method
       * for their particular mixture.
       *)
      z_bi := 1;
      (* order the components arbitrarily in a list *)
      FOR i IN components DO
            z_boiling_comp[z_bi] := i;
            z_bi := z_bi + 1;
      END;
(* use a bubble point sort, pun intended, to order the components. *)
```

```
      FOR i IN [1..CARD[components]-1] DO
            FOR j IN [i+1 .. CARD[components]] DO
                  IF ds.data[z_boiling_comp[i]].Tb >
                        ds.data[z_boiling_comp[j]].Tb THEN
                        z_bc := z_boiling_comp[j];
                        z_boiling_comp[j] := z_boiling_comp[i];
                        z_boiling_comp[i] := z_bc;
                  END;
            END;
      END;
      z_bi := 1;
      (* assign integer alpha’s *)
      FOR i IN [1.. CARD[components]] DO
            Column.feed_tray_state.alpha[z_boiling_comp[i]] :=
                  CARD[components]-i+1;
      END;
      Column.condenser.reflux_ratio := 1.3;
      feed_T := 298 {K};
      feed_P := 1{atm};
      Feed.f[components] := 3{mole/s};
      RUN Column.propagate_feed;
END values;

METHOD scale;
      RUN Column.scale;
END scale;
END demo_column;

MODEL mw_demo_column;
      mw_column IS_A demo_column([‘methanol’,’water’],’water’,13,7);
END mw_demo_column;

MODEL abc_demo_column;
      abc_column IS_A
      demo_column([‘benzene’,’chloroform’,’acetone’],’benzene’,13,7);
END abc_demo_column;

MODEL c567_demo_column;
      c567_column IS_A
      demo_column([‘n_pentane’,’n_hexane’,’n_heptane’],
                  ’n_heptane’,13,7);
END c567_demo_column;
```

A.2.9 PLOT

This library is an ASCEND IIIc library that still works in ASCEND IV, as do all the ASCEND IIIc libraries. We are not offering this as an example of good reusable modeling style. It is included because newisom.asc in the models section of this appendix needs it.

Appendix A Code 12

```
(* plot.lib *)
MODEL plt_point;
      x, y IS_A solver_var;
END plt_point;

(****************************************************************)

MODEL plt_curve;
      legend      IS_A symbol_constant;
      npnt        IS_A integer_constant;
      pnt[1..npnt]IS_A plt_point;

      pnt[1..npnt].xARE_ALIKE;
      pnt[1..npnt].yARE_ALIKE;
END plt_curve;

(****************************************************************)

MODEL plt_plot_integer;
      title, XLabel, YLabel IS_A symbol_constant;
      ncurve            IS_A integer_constant;
      curve[1..ncurve]IS_A plt_curve;
END plt_plot_integer;

(****************************************************************)

MODEL plt_plot_symbol;
      title, XLabel, YLabel IS_A symbol_constant;
      curve_set   IS_A set OF symbol_constant;
      curve[curve_set]IS_A plt_curve;
END plt_plot_symbol;
```

A.2.10 BVP

Note that it would be very easy to incorporate a new integration rule, a refinement of `propagation`, into this library.

Appendix A Code 13(* bvp.lib: a library for collocation *)

```
(* ********************************************************** *)
(* ******  bvp_point: the collocation MODEL API  ************* *)
(* ********************************************************** *)
MODEL bvp_point;
NOTES ‘usage’ SELF {
 Generic definition of the differential equation system,
 Dy[i] = dydx[i], i=1..n_eq, D = d/dx.
 Create a refinement of this type that includes a part
 which is your DAE MODEL and
 merge your variables with x,y,dydx as needed.
 * Note that the integration will behave much better if the
 * typical values of your state variables are all about the
 * same size. Integrators can lie if you give them poorly
 * scaled problems. Dimensionless variables are generally best
 * to use if you can easily formulate your MODEL in those terms.
}
END NOTES;

   n_eq “the number of implicit ODE’s” IS_A integer_constant;
   dydx[1..n_eq] “the derivative variables”,
   y[1..n_eq]   “the state variables”,
   x “the independent variable”       IS_A solver_var;


   METHODS
       METHOD clear;
           x.fixed := FALSE;
           y[1..n_eq].fixed := FALSE;
           dydx[1..n_eq].fixed := FALSE;
       END clear;
       METHOD specify;
           (* The state variables are set
            * TRUE, while their derivatives are FALSE.
            *)
           x.fixed := TRUE;
           y[1..n_eq].fixed := TRUE;
           dydx[1..n_eq].fixed := FALSE;
       END specify;
END bvp_point;


(* ****************************************************** *)
(* *********  bvp internal models *********************** *)
```

```
(* ******************************************************** *)

MODEL propagation(
      n_point WILL_BE integer_constant;
      eval[0..n_point] WILL_BE bvp_point;
      n_eq WILL_BE integer_constant;
) WHERE (
      n_eq == eval[0].n_eq;
);
NOTES
'usage' SELF {
This MODEL does nothing by itself, and should never be created.
Only the refinements of this MODEL should be created.
If you need a new collocation formula, create it by refining
propagation and then update the SELECT and IF statements in
the MODEL integration.
}
'usage' n_order {
n_order did nothing.
It was mainly a carrier of information for the user.
Specifically, it gives the order of the error of the collocation.
in terms of h.
this information should just be kept in NOTES instead, like so.
}
END NOTES;

      h IS_A solver_var;
        initial ALIASES  eval[0];
        final ALIASES eval[n_point];
        final.x = initial.x + h;

    METHODS
        METHOD clear;
            RUN eval[0..n_point].clear;
            h.fixed := FALSE;
        END clear;
        METHOD specify;
            RUN eval[0..n_point].specify;
            h.fixed := TRUE;
            final.x.fixed := FALSE;
        END specify;
        METHOD reset;
            RUN clear;
            RUN specify;
        END reset;
END propagation;

MODEL euler(
      n_point WILL_BE integer_constant;
      eval[0..n_point] WILL_BE bvp_point;
```

```
      n_eq WILL_BE integer_constant;
) WHERE (
      n_eq == eval[0].n_eq;
      n_point == 1;
) REFINES propagation();
NOTES
'order of error' SELF {
The error in the euler method is O(h).
}
END NOTES;

        FOR i IN [1..n_eq] CREATE
            eval[1].y[i] = eval[0].y[i] + h*eval[0].dydx[i];
        END;

    METHODS
        METHOD specify;
            RUN eval[0..n_point].specify;
            h.fixed := TRUE;
            final.x.fixed := FALSE;
            eval[1..n_point].y[1..n_eq].fixed := FALSE;
            eval[1..n_point].x.fixed := FALSE;
        END specify;

END euler;

MODEL trapezoid(
      n_point WILL_BE integer_constant;
      eval[0..n_point] WILL_BE bvp_point;
      n_eq WILL_BE integer_constant;
) WHERE (
      n_eq == eval[0].n_eq;
      n_point == 1;
) REFINES propagation();

NOTES
'order of error' SELF {
The error in the trapezoidal rule is O(h^2).
}
END NOTES;

        FOR i IN [1..n_eq] CREATE
            eval[1].y[i] = eval[0].y[i] + 0.5*h*
                (eval[0].dydx[i] + eval[1].dydx[i]);
        END;

    METHODS
        METHOD specify;
            RUN eval[0..n_point].specify;
            h.fixed := TRUE;
```

```
            final.x.fixed := FALSE;
            eval[1..n_point].y[1..n_eq].fixed := FALSE;
            eval[1..n_point].x.fixed := FALSE;
        END specify;

END trapezoid;

MODEL midpoint(
      n_point WILL_BE integer_constant;
      eval[0..n_point] WILL_BE bvp_point;
      n_eq WILL_BE integer_constant;
) WHERE (
      n_eq == eval[0].n_eq;
      n_point == 2;
) REFINES propagation();

NOTES
'order of error' SELF {
The error in the midpoint rule is O(h^2).
}
END NOTES;

        eval[1].x = initial.x + (1.0/2.0)*h;
        FOR i IN [1..n_eq] CREATE
            eval[1].y[i] = eval[0].y[i] + (1.0/2.0)*h*
                eval[0].dydx[i];
            eval[2].y[i] = eval[0].y[i] + h*
                eval[1].dydx[i];
        END;

    METHODS
        METHOD specify;
            RUN eval[0..n_point].specify;
            h.fixed := TRUE;
            final.x.fixed := FALSE;
            eval[1..n_point].y[1..n_eq].fixed := FALSE;
            eval[1..n_point].x.fixed := FALSE;
        END specify;

END midpoint;

MODEL simpsons(
      n_point WILL_BE integer_constant;
      eval[0..n_point] WILL_BE bvp_point;
      n_eq WILL_BE integer_constant;
) WHERE (
      n_eq == eval[0].n_eq;
      n_point == 2;
) REFINES propagation();
```

```
NOTES
'order of error' SELF {
The error in simpson's rule is O(h^4).
}
END NOTES;

        eval[1].x = initial.x + 0.5*h;
        FOR i IN [1..n_eq] CREATE
            eval[1].y[i] = eval[0].y[i] + 0.5*h*
                ((5.0/12.0)*eval[0].dydx[i] +
                (8.0/12.0)*eval[1].dydx[i] -
                (1.0/12.0)*eval[2].dydx[i]);
            eval[2].y[i] = eval[0].y[i] + h*
                ((1.0/6.0)*eval[0].dydx[i] +
                (4.0/6.0)*eval[1].dydx[i] +
                (1.0/6.0)*eval[2].dydx[i]);
        END;

    METHODS
        METHOD specify;
            RUN eval[0..n_point].specify;
            h.fixed := TRUE;
            final.x.fixed := FALSE;
            eval[1..n_point].y[1..n_eq].fixed := FALSE;
            eval[1..n_point].x.fixed := FALSE;
        END specify;

END simpsons;

MODEL runge_kutta_gill(
      n_point WILL_BE integer_constant;
      eval[0..n_point] WILL_BE bvp_point;
      n_eq WILL_BE integer_constant;
) WHERE (
      n_eq == eval[0].n_eq;
      n_point == 4;
) REFINES propagation();

NOTES
'order of error' SELF {
The error in the 4 point Runge-Kutta-Gill method is O(h^4).
}
END NOTES;

        eval[1].x = eval[2].x;
        eval[3].x = eval[4].x;
        eval[1].x = initial.x + 0.5*h;
        FOR i IN [1..n_eq] CREATE
            eval[1].y[i] = eval[0].y[i] + 0.5*h*
                eval[0].dydx[i];
```

```
            eval[2].y[i] = eval[0].y[i] + 0.5*h*
                (0.41421356*eval[0].dydx[i] +
                0.58578644*eval[1].dydx[i]);
            eval[3].y[i] = eval[0].y[i] + h*
                (-0.707106781*eval[1].dydx[i] +
                1.707106781*eval[2].dydx[i]);
            eval[4].y[i] = eval[0].y[i] + h*
                (0.166666667*eval[0].dydx[i] +
                0.097631073*eval[1].dydx[i] +
                0.569035590*eval[2].dydx[i] +
                0.166666667*eval[3].dydx[i]);
        END;

    METHODS
        METHOD specify;
            RUN eval[0..n_point].specify;
            h.fixed := TRUE;
            final.x.fixed := FALSE;
            eval[1..n_point].y[1..n_eq].fixed := FALSE;
            eval[1..n_point].x.fixed := FALSE;
        END specify;

END runge_kutta_gill;

MODEL gauss(
      n_point WILL_BE integer_constant;
      eval[0..n_point] WILL_BE bvp_point;
      n_eq WILL_BE integer_constant;
) WHERE (
      n_eq == eval[0].n_eq;
      n_point == 4;
) REFINES propagation();

NOTES
'order of error' SELF {
The error in 4 point gauss method is O(h^6).
}
END NOTES;

        eval[1].x = initial.x + 0.11270167*h;
        eval[2].x = initial.x + 0.5*h;
        eval[3].x = initial.x + 0.88729833*h;
        FOR i IN [1..n_eq] CREATE
            eval[1].y[i] = initial.y[i] + 0.11270167*h*
                (1.23236*eval[1].dydx[i] -
                0.31922*eval[2].dydx[i] +
                0.08686*eval[3].dydx[i]);
            eval[2].y[i] = initial.y[i] + 0.5*h*
                (0.60053*eval[1].dydx[i] +
                0.44444*eval[2].dydx[i] -
```

```
                0.04497*eval[3].dydx[i]);
            eval[3].y[i] = initial.y[i] + 0.88729833*h*
                (0.30203*eval[1].dydx[i] +
                0.54144*eval[2].dydx[i] +
                0.15653*eval[3].dydx[i]);
            eval[4].y[i] = initial.y[i] + h*
                (0.27778*eval[1].dydx[i] +
                0.44444*eval[2].dydx[i] +
                0.27778*eval[3].dydx[i]);
        END;

    METHODS
        METHOD specify;
            RUN eval[0..n_point].specify;
            h.fixed := TRUE;
            final.x.fixed := FALSE;
            eval[1..n_point].y[1..n_eq].fixed := FALSE;
            eval[1..n_point].x.fixed := FALSE;
        END specify;

END gauss;

MODEL lobatto(
      n_point WILL_BE integer_constant;
      eval[0..n_point] WILL_BE bvp_point;
      n_eq WILL_BE integer_constant;
) WHERE (
      n_eq == eval[0].n_eq;
      n_point == 4;
) REFINES propagation();

NOTES
'order of error' SELF {
The error in the 4 lobatto point method is O(h^8).
}
END NOTES;

        eval[1].x = initial.x + 0.11270167*h;
        eval[2].x = initial.x + 0.5*h;
        eval[3].x = initial.x + 0.88729833*h;
        FOR i IN [1..n_eq] CREATE
            eval[1].y[i] = initial.y[i] + 0.11270167*h*
                (0.428010*eval[0].dydx[i] +
                0.602339*eval[1].dydx[i] -
                0.044301*eval[2].dydx[i] +
                0.029587*eval[3].dydx[i] -
                0.015635*eval[4].dydx[i]);
            eval[2].y[i] = initial.y[i] + 0.5*h*
                (-0.06250*eval[0].dydx[i] +
                0.68122*eval[1].dydx[i] +
```

```
                0.44444*eval[2].dydx[i] -
                0.12566*eval[3].dydx[i] +
                0.06250*eval[4].dydx[i]);
            eval[3].y[i] = initial.y[i] + 0.88729833*h*
                (0.001986*eval[0].dydx[i] +
                0.309303*eval[1].dydx[i] +
                0.506524*eval[2].dydx[i] +
                0.236552*eval[3].dydx[i] -
                0.054365*eval[4].dydx[i]);
            eval[4].y[i] = initial.y[i] + h*
                (0.277778*eval[1].dydx[i] +
                0.444444*eval[2].dydx[i] +
                0.277778*eval[3].dydx[i]);
        END;

    METHODS
        METHOD specify;
            RUN eval[0..n_point].specify;
            h.fixed := TRUE;
            final.x.fixed := FALSE;
            eval[1..n_point].y[1..n_eq].fixed := FALSE;
            eval[1..n_point].x.fixed := FALSE;
        END specify;

END lobatto;

MODEL integration(
      nstep IS_A integer_constant;
      npoint IS_A integer_constant;
      method IS_A symbol_constant;
      grid[0..nstep*npoint] WILL_BE bvp_point;
      stepsize WILL_BE solver_var;
      n_eq IS_A integer_constant;
) WHERE (
      ((method == ‘Euler’) AND (npoint == 1)) OR
      ((method == ‘trapezoidal’) AND (npoint == 1)) OR
      ((method == ‘midpoint’) AND (npoint == 2)) OR
      ((method == ‘Simpsons’) AND (npoint == 2)) OR
      ((method == ‘Runge-Kutta-Gill’) AND (npoint == 4)) OR
      ((method == ‘Gauss’) AND (npoint == 4)) OR
      ((method == ‘Lobatto’) AND (npoint == 4));
);

NOTES ‘usage’ SELF {
This writes the equations for a mesh with nstep collocation
intervals. In each interval there npoint nodes.
The total mesh has (nstep * npoint + 1) nodes.

In order to use this MODEL, create a refinement of bvp_point
that contains the MODEL you want to integrate, then create
```

```
an array of them (grid) and pass it in along with the other options.
See the MODEL bvp_point for details of how to do this.
See the MODELs bvp_test and bvp_dynamics for an example.
}
END NOTES;

      (* cook up the little arrays for propagation out of the
       * big array from the user.
       *)
      FOR i IN [0..nstep-1] CREATE
            step_nodes[i+1][pset[i]] ALIASES
                  (grid[(i*npoint)..((i+1)*npoint)])
            WHERE pset[i] IS_A set OF integer_constant WITH_VALUE
                  (0..npoint);
      END;

      (* cook up the steps for various types of integration rule *)
      SELECT (method)
      CASE 'Euler':
            FOR i IN [1..nstep] CREATE
                  euler1_step[i] IS_A
                  euler(npoint,step_nodes[i],n_eq);
            END;
      CASE 'trapezoidal':
            FOR i IN [1..nstep] CREATE
                  trapezoid1_step[i] IS_A
                  trapezoid(npoint,step_nodes[i],n_eq);
            END;
      CASE 'midpoint':
            FOR i IN [1..nstep] CREATE
                  midpoint2_step[i] IS_A
                  midpoint(npoint,step_nodes[i],n_eq);
            END;
      CASE 'Simpsons':
            FOR i IN [1..nstep] CREATE
                  simpsons2_step[i] IS_A
                  simpsons(npoint,step_nodes[i],n_eq);
            END;
      CASE 'Runge-Kutta-Gill':
            FOR i IN [1..nstep] CREATE
                  rkg4_step[i] IS_A
                  runge_kutta_gill(npoint,step_nodes[i],n_eq);
            END;
      CASE 'Gauss':
            FOR i IN [1..nstep] CREATE
                  gauss4_step[i] IS_A
                  gauss(npoint,step_nodes[i],n_eq);
            END;
      CASE 'Lobatto':
            FOR i IN [1..nstep] CREATE
```

```
            lobatto4_step[i] IS_A
            lobatto(npoint,step_nodes[i],n_eq);
      END;
END;


METHODS
METHOD values;
      IF method == 'Euler' THEN
            euler1_step[1..nstep].h := stepsize;
      END;
      IF method == 'trapezoidal' THEN
            trapezoid1_step[1..nstep].h := stepsize;
      END;
      IF method == 'midpoint' THEN
            midpoint2_step[1..nstep].h := stepsize;
      END;
      IF method == 'Simpsons' THEN
            simpsons2_step[1..nstep].h := stepsize;
      END;
      IF method == 'Runge-Kutta-Gill' THEN
            rkg4_step[1..nstep].h := stepsize;
      END;
      IF method == 'Gauss' THEN
            gauss4_step[1..nstep].h := stepsize;
      END;
      IF method == 'Lobatto' THEN
            lobatto4_step[1..nstep].h := stepsize;
      END;
  END values;
METHOD clear;
      stepsize.fixed := FALSE;
      IF method == 'Euler' THEN
            RUN euler1_step[1..nstep].clear;
      END;
      IF method == 'trapezoidal' THEN
            RUN trapezoid1_step[1..nstep].clear;
      END;
      IF method == 'midpoint' THEN
            RUN midpoint2_step[1..nstep].clear;
      END;
      IF method == 'Simpsons' THEN
            RUN simpsons2_step[1..nstep].clear;
      END;
      IF method == 'Runge-Kutta-Gill' THEN
            RUN rkg4_step[1..nstep].clear;
      END;
      IF method == 'Gauss' THEN
            RUN gauss4_step[1..nstep].clear;
      END;
```

```
        IF method == 'Lobatto' THEN
            RUN lobatto4_step[1..nstep].clear;
        END;
    END clear;
    METHOD specify;
        stepsize.fixed := TRUE;
        IF method == 'Euler' THEN
            FOR i IN [1..nstep] DECREASING DO
                RUN euler1_step[i].specify;
            END;
        END;
        IF method == 'trapezoidal' THEN
            FOR i IN [1..nstep] DECREASING DO
                RUN trapezoid1_step[i].specify;
            END;
        END;
        IF method == 'midpoint' THEN
            FOR i IN [1..nstep] DECREASING DO
                RUN midpoint2_step[i].specify;
            END;
        END;
        IF method == 'Simpsons' THEN
            FOR i IN [1..nstep] DECREASING DO
                RUN simpsons2_step[i].specify;
            END;
        END;
        IF method == 'Runge-Kutta-Gill' THEN
            FOR i IN [1..nstep] DECREASING DO
                RUN rkg4_step[i].specify;
            END;
        END;
        IF method == 'Gauss' THEN
            FOR i IN [1..nstep] DECREASING DO
                RUN gauss4_step[i].specify;
            END;
        END;
        IF method == 'Lobatto' THEN
            FOR i IN [1..nstep] DECREASING DO
                RUN lobatto4_step[i].specify;
            END;
        END;
    END specify;

    METHOD reset;
        RUN clear;
        RUN specify;
    END reset;

END integration;
```

```
(**********************************************************)
(************** demo and test models **********************)
(**********************************************************)

MODEL test_dynamics;
      x IS_A factor;
      t IS_A time;
      xdot IS_A frequency;
      xdot = 2{1/s} * x * exp(-2{1/s}*t) * cos(t*1{rad/s});
METHODS
METHOD clear;
      x.fixed := FALSE;
      xdot.fixed := FALSE;
      t.fixed := FALSE;
END clear;
METHOD specify;
      x.fixed := TRUE;
      xdot.fixed := FALSE;
      t.fixed := TRUE;
END specify;
METHOD values;
      xdot.lower_bound := -1e6{1/s};
END values;
END test_dynamics;

MODEL dynamic_point REFINES bvp_point;

      n_eq :== 1; (* you can have as many as you like *)

      (* replace the following line with an IS_A of your MODEL *)
      your_model IS_A test_dynamics;

      (* connect independent variables. If your MODEL is autonomous,
       * you don't need to, but then you must call
       * bvp_point::clear and bvp_point::specify in your methods
       * to get x set properly.
       *)
      x, your_model.t ARE_THE_SAME;

      (* connect your variables to the integration state variables *)
      y[1], your_model.x ARE_THE_SAME;

      (* connect your variables to the integration derivatives *)
      dydx[1], your_model.xdot ARE_THE_SAME;

METHODS

METHOD clear;

      RUN your_model.clear;
```

```
      (* You don’t need to run bvp_point::clear because
      you’ve wired all the bvp variables into some of
      yours. (Not TRUE if your MODEL is autonomous, though).
            RUN bvp_point::clear;
       *)
END clear;

METHOD specify;

      RUN your_model.specify;
NOTES
‘usage’ SELF{
If you write your_model correctly, you don’t need
to run bvp_point::specify because you’ve wired all
the bvp variables into some of yours.
(Not TRUE if your MODEL is autonomous, though).
RUN bvp_point::specify;
}
END NOTES;

END specify;
END dynamic_point;


MODEL bvp_test;
      stepsize IS_A time;
      npoint, nstep IS_A integer_constant;
      npoint :== 1;
      nstep :== 100;
      nodes[0..nstep*npoint] IS_A dynamic_point;
      initial ALIASES nodes[0];
      final ALIASES nodes[nstep*npoint];
      integral IS_A
      integration(nstep,npoint,’Euler’,nodes,stepsize,initial.n_eq);

      (* we don’t tie the test MODEL to the ASCEND plot MODEL *)

METHODS
METHOD clear;
      stepsize.fixed := FALSE;
      RUN integral.clear;
      RUN nodes[0..nstep*npoint].clear;
END clear;
METHOD specify;
      stepsize.fixed := TRUE;
      RUN integral.specify;
END specify;
METHOD values;
      stepsize := 0.05{s};
```

```
      initial.x := 0{s};
      initial.y[1] := 1;
      RUN nodes[0..nstep*npoint].your_model.values;
      RUN integral.values;
END values;

END bvp_test;
```