THERMOSYS™ 4.1

User Guide

Release Version 4.1.1

October 2013

Acknowledgements

THERMOSYS™ was originally developed through work funded by the Air Conditioning and Refrigeration Center at the University of Illinois at Urbana-Champaign (UIUC). The following past and present graduate students from the Alleyne Research Group at UIUC have had a considerable impact upon the development of THERMOSYS: Bryan Rasmussen, Rajat Shah, Michael Keir, Brian Eldredge, Tom McKinley, Bin Li, Neera Jain, Joseph Fasl, Justin Koeln, and Matthew Williams. THERMOSYS was licensed to CU Aerospace in 2013 via UIUC's Office of Technology Management; our special thanks to Prof. Andrew Alleyne and Jeff Wallace for enabling this technology transfer.

Foreword

THERMOSYS can be downloaded via the link provided in the email sent upon purchase of the software. Additionally, all future releases of THERMOSYS will be announced via the website and via email to registered users.

The developers of THERMOSYS ask that if there is a need to reference THERMOSYS for any publications or presentations, the following citations be used:

CU Aerospace, THERMOSYS version 4, 2013 [http://thermosys.us].

Alleyne, A., et. al., THERMOSYS 4 Toolbox, University of Illinois at Urbana-Champaign, 2012, [http://arg.mechse.illinois.edu/index.php?id=1161%7CTHERMOSYS].

Please report any bugs or concerns by email to the THERMOSYS developers at thermosys@cuaerospace.com, or through the "CONTACT US" link on the THERMOSYS website. CU Aerospace offers limited free support of THERMOSYS as well as enhanced technical support for a fee.

Revision History

THERMOSYS 4 Beta Release ..... *November, 2012*

THERMOSYS 4.0.2 ..... *May, 2013*

THERMOSYS 4.0.3 ..... *May, 2013*

THERMOSYS 4.1 ..... *August, 2013*

THERMOSYS 4.1.1 ..... *October, 2013*

Table of Contents

CHAPTER **Page**

List of Tables

List of Figures

Nomenclature

ACRC ..... Air Conditioning and Refrigeration Center
ARG ..... Alleyne Research Group
EEV ..... Electronic Expansion Valve
GUI ..... Graphical User Interface
NIST ..... National Institute of Standards and Technology
PI ..... Proportional-Integral
TXV ..... Thermostatic Expansion Valve
UIUC ..... University of Illinois at Urbana-Champaign

Chapter 1

Introduction

1.1 THERMOSYS 4.1

The THERMOSYS toolbox for MATLAB/Simulink is a suite of modeling and simulation tools for vapor compression systems. It was developed at the University of Illinois at Urbana-Champaign through sponsorship by the Air Conditioning and Refrigeration Center. THERMOSYS 4.1 is designed to facilitate the inclusion of new and/or custom models to the existing software tool and, in the process, eliminate compatibility issues between future releases of THERMOSYS and different versions of MATLAB. This requires a framework that relies on Simulink, a visual programming package, to handle the simulation of the component models while the models themselves are coded as Level 2 MATLAB S-Functions.

The purpose of this document is to provide new users with the necessary background to navigate the THERMOSYS toolbox. Details of model derivations are not included here but references are provided in subsequent sections. In the remaining sections of this chapter, a brief history of the THERMOSYS toolbox will be described. Then, the specific component and fluid property features of THERMOSYS 4.1 will be discussed along with installation instructions for the toolbox.

1.2 THERMOSYS Toolbox Progression

Over the last decade, THERMOSYS has gone through multiple revisions as shown in Figure 1.1. THERMOSYS 4.1.1 is the latest of these revisions which includes significant improvements over previous versions. In this section a brief history of previous THERMOSYS versions is provided to help the reader understand how THERMOSYS has evolved.

The first THERMOSYS toolbox was developed in 2002 and was called THERMOSYS ACRC. The models were developed in native Simulink. This generally provides advantages with respect to simulation speed. However, in the years after these models were developed, MathWorks released newer versions of MATLAB and Simulink in which older model files were

not compatible. To eliminate compatibility issues between future releases of THERMOSYS and different versions of MATLAB, THERMOSYS Academic was developed around 2005 (later renamed THERMOSYS 2007a) which utilized a framework that relied on Simulink to handle the simulation of the dynamic models while the models themselves were coded as MATLAB functions.

In parallel with the development of THERMOSYS 2007a came the development of THERMOSYS 3.1 which utilized S-functions to create new heat exchanger models containing a switched model framework [1]. These followed the format of THERMOSYS 2007a in which both the MATLAB and Simulink environments were used during a simulation. However, the algebraic models for static components such as the electronic expansion valve and compressor were returned to the native Simulink environment as was done in THERMOSYS ACRC with the objective of trying to improve computation speed of the models. THERMOSYS 3.1 also utilized GUIs for specifying operational and physical parameters whereas this had been done using m-files in THERMOSYS 2007a.

In 2011, a decision was made to discontinue the use of THERMOSYS 2007a and focus on improvements to THERMOSYS 3.1 to make it more efficient and user friendly. This culminated in the release of THERMOSYS 4 which utilizes Level 2 MATLAB S-Functions to code all component models, static and dynamic, in the MATLAB environment. This produces Simulink models which run faster than real-time.

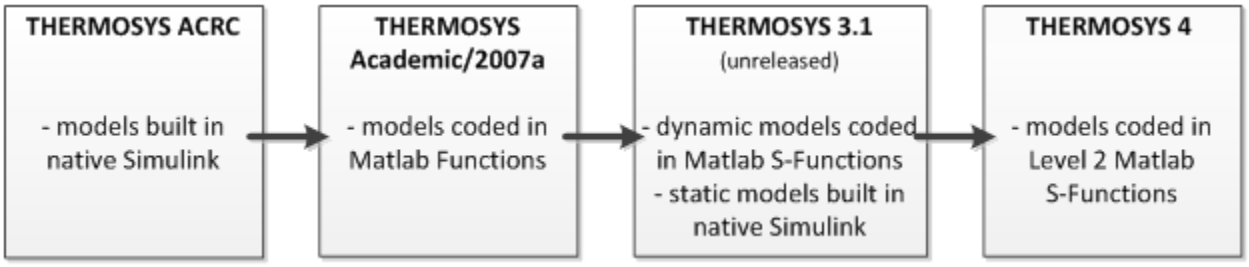

Figure 1.1: THERMOSYS Evolution

1.3 Features of THERMOSYS 4.1

1.3.1 Components

There are a number of different component models that have been developed for THERMOSYS 4.1. These are described in Table 1.1.

Table 1.1 Complete List of Available Models in THERMOSYS 4.1

Model Name	Description	Model Type
Box Model	Linear environment/load model	Dynamic
Compressor	Nonlinear compressor model	Static
Condenser	Nonlinear switched condenser model	Dynamic
EEV (Poly)	Nonlinear Electronic Expansion Valve model (calculations based on polynomial)	Static
EEV (Maps)	Nonlinear Electronic Expansion Valve model (calculations based on empirical map)	Static
Evaporator	Nonlinear evaporator model	Dynamic
Hydro Resistance	Nonlinear hydraulic resistance model (3 versions)	Static
Multipurpose Tank	Nonlinear refrigerant tank model	Dynamic
TXV	Nonlinear Thermostatic Expansion Valve model	Dynamic

1.3.2 Fluid Properties

Refrigerant properties are a necessary part of vapor compression system modeling. As will be described in Chapter 2, individual component model equations are described in THERMOSYS using Level 2 MATLAB S-Functions. Therefore, the fluid properties for a given refrigerant are accessed through a single MATLAB structure. In the THERMOSYS 4.1.1 release, fluid properties for refrigerants R134a and R404A are included. Instructions on developing fluid property tables (in the form of MATLAB structures) for additional refrigerants and/or other fluids are provided in Appendix A.

Although not needed to run a system model in THERMOSYS 4.1, fluid property table Simulink blocks are also accessible through the Simulink Library Browser as shown in Figure 1.2.

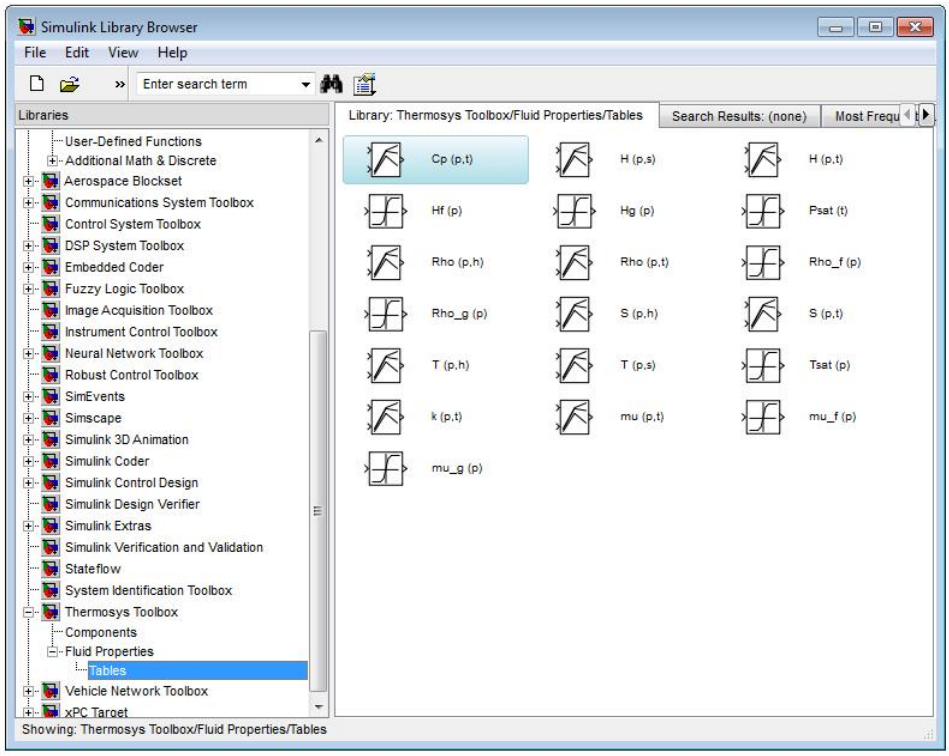

Figure 1.2: Fluid Property Tables in Simulink Library Browser

These were commonly used in previous releases of THERMOSYS in which the component models were entirely Simulink based and have been included here for use in the Simulink environment if desired by the user.

1.4 Organization of User Guide

In the remainder of this document, the reader will be provided with a guided tour through the THERMOSYS toolbox. Chapter 2 describes the model structure of THERMOSYS, and Chapter 3 describes, in detail, how one can build a system model in THERMOSYS using the available component models. Finally, Chapter 4 provides a tutorial on running one of the sample systems provided with the toolbox.

Chapter 2

THERMOSYS Model Structure

This chapter provides an explanation regarding how the mathematics behind THERMOSYS 4.1 was implemented into the MATLAB/Simulink environment. For an in depth explanation about the mathematical theory itself, please visit *http://arg.mechse.illinois.edu/* and view the theses of the students who contributed to the development of THERMOSYS in the publications section of the website. The students' names can be found in the forward of this user guide, and on the THERMOSYS page of the ARG website.

The implementation of THERMOSYS can be subdivided into three main parts: 1) the individual component S-Functions, 2) the individual component blocks and masks in the Simulink environment which house the S-Functions, and 3) additional supporting functions that create fluid properties, store component maps, as well as perform other auxiliary calculations for the system. This chapter will cover the use of each of these in detail.

2.1 Component S-Functions

S-Functions, or system-functions, are a powerful way of combining the advantages of the block based Simulink environment with the advantages of code based MATLAB scripts to model dynamic systems. In its most general form, the S-Function is a MATLAB script that contains a number of embedded functions, unique to each individual S-Function, which execute in a specific sequence in order to simulate the system as if it were modeled in the Simulink block based environment. S-Functions are able to model discrete, continuous, or hybrid components and systems, and THERMOSYS 4.1 utilizes this capability of these functions to model both static and dynamic components of a refrigeration system.

The sequence of steps for flow of information in a Simulink based simulation is displayed below in Figure 2.1.

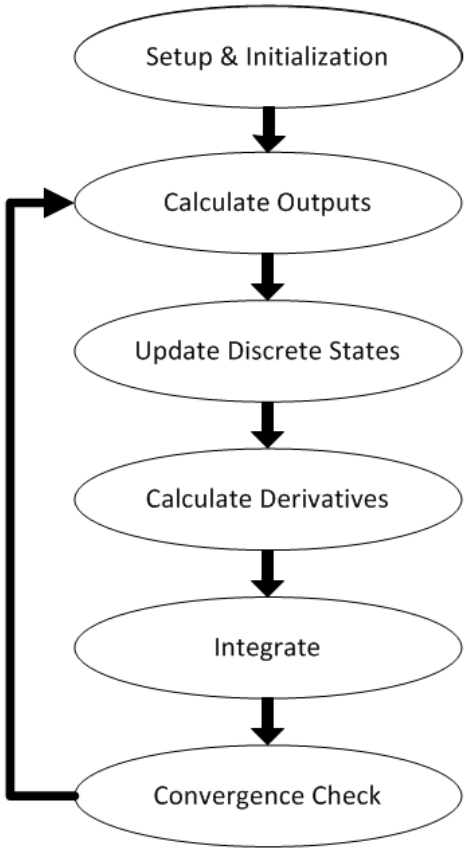

Figure 2.1: Sequence of Steps in a Simulink Model

In order to complete each of the individual steps shown above in Figure 2.1, the S-Function contains sub-functions which are executed by the solver in a particular order. A description of the purpose of each sub-function in the order they are called is listed below.

1. **Setup** – Provides the interface between the S-Function and the block in the Simulink environment. Sets up basic characteristics and allocates memory for objects such as parameters, inputs, outputs, continuous states, as well as initializes the number and order of sub-functions within the S-Function. Only run once per simulation.
2. **Post Propagation Setup** – Sets up the memory allocation for the discrete state vector, as well as the number and properties of each of the discrete states. Only run once per simulation.

3. **Start** – Initializes discrete and continuous states. Only run once at the beginning of the simulation.
4. **Outputs** – Calculates and then passes the output values to the output ports of the Simulink block containing the S-Function. Run first after initialization, and then at every time step of the simulation.
5. **Update** – Calculates discrete state values and assigns these to the discrete state vector. Called and run at each time step of the simulation.
6. **Derivatives** – Calculates the values of the derivatives of each continuous state, and then stores them within the derivative vector. Called and run at each time step of the simulation.

After the derivative function has been called, the Simulink solver will then numerically integrate the derivative vector and check for convergence of the solutions to a prescribed tolerance set in the Simulink interface. If the solutions converge, the simulation will restart the cycle at the output stage for the next time step. If they do not convergence, the simulation with decrease the current time step size and recalculate derivatives and reintegrate until convergence is achieved. It should be noted that S-Functions without continuous states do not contain the *Derivatives* sub-function, and simple static input-output models do not contain the *Update* sub-function as well.

2.2 Component Blocks and Masks

2.2.1 Component Blocks

In order to interface MATLAB S-Functions with the Simulink environment, Level 2 MATLAB S-Function blocks are used which provide the tools for this interfacing. They are contained within the *User-Defined Functions* tab of the Simulink Library Browser.

An example of the exterior of the Level 2 MATLAB S-Function block for the EEV is shown in Figure 2.2 with markings indicating where to connect the input and output signals.

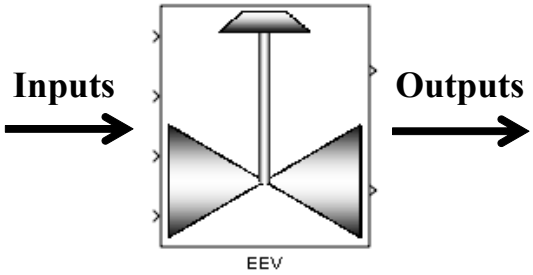

Figure 2.2: S-Function Block Exterior for the EEV

An example of the interior of a Level 2 MATLAB S-Function block is shown below in **Error! Reference source not found.**.

2.2.2 Component Masks

Component masks represent the way in which user supplied parameters and initialization calculations are handled in the Simulink S-Function code structure. Figure 2.3 below shows an example of the user input mask for the THERMOSYS 4.1 condenser model.

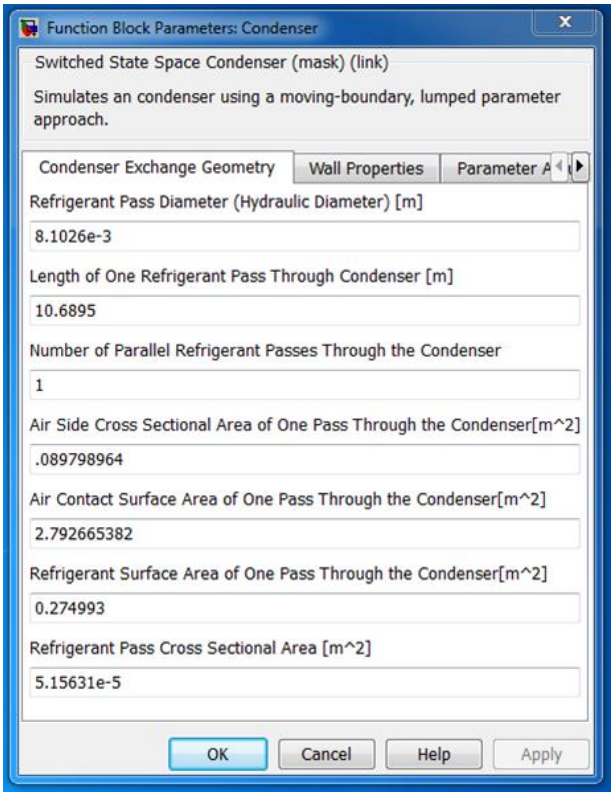

Figure 2.3: Condenser Mask

Users can then edit the preset parameters of the components (such as geometric parameters, operating conditions, etc.). These masks are accessed from the Simulink model environment by double clicking on an S-Function block once the masks are created.

To begin creating a mask in a new S-Function block, the user must right click on the block and select *Create Mask*. (Note: this becomes *Edit Mask* after the first time it is selected). After this a window will appear similar to that in Figure 2.4 below.

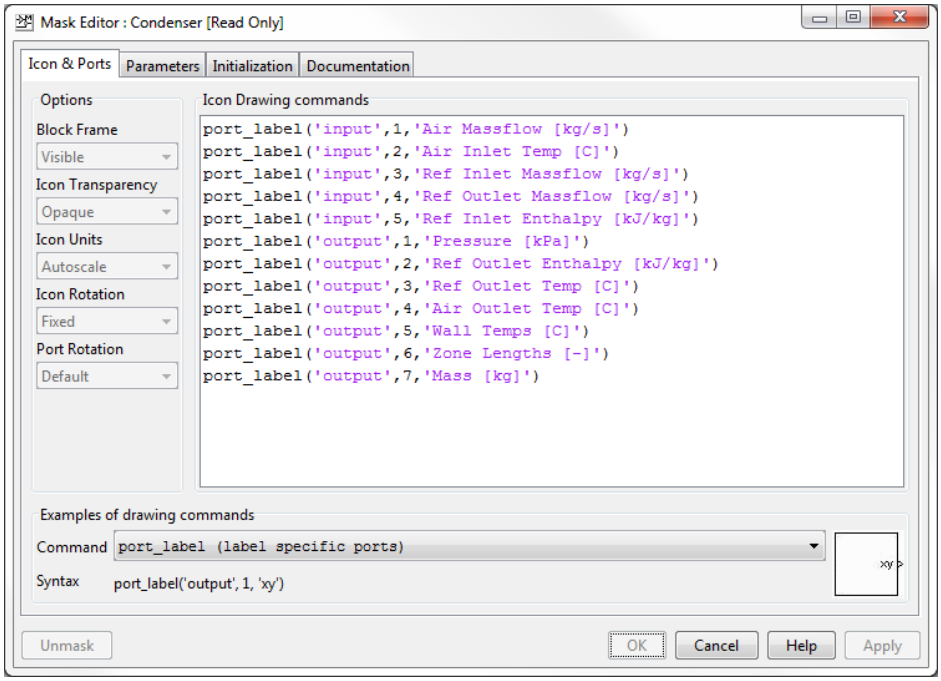

Figure 2.4: Mask Editor Dialog Box

The different tabs at the top of this dialog box correspond to different parts of the mask structure that need to be specified in order to create a mask. Each is briefly explained below.

- **Icon & Ports** – Controls the visual graphics on the exterior of the Level 2 MATLAB S-Function block. Allows for pictures, port by port labels, colors, etc.
- **Parameters** – Creates and sets properties for user inputs such as those shown in Figure 2.3. Allows various types of inputs such as value, checkbox, dropdown box, etc.
- **Initialization** – Customizable initialization script utilizing user inputs that runs when any of the inputs are updated. This script can be used to calculate mask

parameters from user inputs which are then passed into the S-Function through the component block.

- **Documentation** – Allows for input of block description and customizable help files which can be easily accessed from the Simulink environment.

Once the mask is created in the component block, the following steps detail the sequence of actions to set up a simple mask.

1. Define the necessary user inputs and their properties in the parameters tab.
2. Create initialization code to calculate necessary parameters from user inputs in the initialization tab.
3. (Optional) Create help instructions such as component, input-output port, or user input descriptions in the documentation tab.
4. (Optional) Add graphics and port labels to the block in the Icons & Ports tab.

2.3 Support Functions

The final part of the implementation of THERMOSYS into the MATLAB/Simulink environment is the support functions. In the THERMOSYS 4.1 directory the folder labeled *Support Functions* contains a number of different functions, maps, and generation files that are crucial to the implementation of THERMOSYS, but are not specific to any particular component, and some are shared between multiple components. Because of this, these functions and maps are grouped within this folder and called within one or multiple S-Functions. The folders contained within the *Support Functions* folder and their contents are explained below.

- **Fluid Table Generation** – This folder contains the MATLAB scripts that were developed to create property data tables. These tables were populated with data from the NIST program REFPROP 9 [2] and can be interpolated in S-Function codes. Outdated software within REFPROP makes this process incompatible with current MATLAB versions, and an update is in development.
- **Performance Maps** – This folder contains the physical value maps created for compressor and valve components from data taken on the sample system owned

by the Alleyne Research Group at the University of Illinois at Urbana Champaign. More details regarding their creation are given in Appendix A.

- **Property Functions** – This folder contains various physical correlations and other empirically derived functions that calculate properties which are utilized by multiple component S-Functions. Most of these provide a reference to the publication where the correlation was developed at the beginning of the code of each function.
- **Utility Functions** – This folder contains functions which calculate numerical derivatives and perform fast interpolations. These are used many times throughout multiple S-Functions and can be utilized outside of THERMOSYS as well for engineering calculations. Most of these files provide explanations and input descriptions at the beginning of the code of each function.

As improved property correlations, faster interpolation schemes, or new components which require other properties not yet modeled are developed, more functions will be placed into this folder in future versions of THERMOSYS. The user is advised to place any generic property correlation or utility function they create into this folder to keep the file structure organized.

Chapter 3

Building a THERMOSYS System Model

This tutorial covers the creation of a complete system model in THERMOSYS 4.1 using individual component models provided in the toolbox. It is assumed that the toolbox has been installed following the instructions provided in Chapter 1 and that the reader is familiar with the MATLAB/Simulink software package. The model to be created represents a standard air conditioning system, similar to the tutorial in Chapter 4 which describes how to run one of the sample systems provided with THERMOSYS 4.1.1. Specifically, this tutorial recreates *sample model 4*, given in the toolbox.

The steps in creating a model must follow a specific chronological sequence. Each step is described in the sections below. The order of the sections is the same as the required order of these steps. While it may be possible to successfully create a model by deviating from these steps, the steps outline the recommended method to creating a sample system.

3.1 Translating the Physical System into a Model

The first step in developing a THERMOSYS model of a physical system is determining the type and quantity of components needed to accurately represent the system. A basic system can be created from a single compressor, condenser, expansion valve, and evaporator. More complex systems can be created by adding receiver tanks, box models, or multiple evaporators, condensers, etc. The starting point for building a simulation is to envision the THERMOSYS model from the physical system. The outcomes of this process are:

- A list of the types of elements needed to model the system of interest (e.g. evaporators, receivers, condensers, expansion valves, compressors, pipe loss elements, etc.).
- The number of each type of element to be used.

3.2 Build the Simulink Diagram

3.2.3 Add Model Blocks

To add model blocks to a Simulink workspace, simply drag and drop the block from the Simulink library into the workspace. It is recommended that all components be added to the system before any components are connected, in order to present a clear picture of the desired resulting system. To recreate the sample model, a valve, compressor, condenser, and evaporator must be added to the workspace.

3.2.4 Connect Model Blocks

When connecting the components, it is critical to connect to correct input and output ports between each of the components. Using the *goto* and *from* tags in Simulink are the simplest way to connect the components. The *goto* and *from* tag labels in the sample systems demonstrate the proper way to connect each component.

Not all of the outputs from one component need to be connected to other components (e.g. outlet temperature of refrigerant from the evaporator). Additionally, not all inputs can be obtained from other components (e.g. valve opening signal). Typically, refrigerant pressure, enthalpy, and mass flow rate need to be shared between components. Air temperature and mass flow rates, as well as any other unaccounted input ports should be treated as inputs (discussed in section 3.3). The outlet ports which are not used as inputs to other components are instead used to monitor the behavior of the component.

In the THERMOSYS 4.1 update, the previous sixth input of the condenser which represented the input enthalpy derivative has been eliminated from modeling. Additionally, the first order model of the compressor shell has now been directly incorporated into the compressor S-Function. The time constant and initial condition can be user specified in the compressor block mask. If such a response is not desired, a very low value of time constant can be used to simulate an instantaneous effect.

3.2.5 Input Mask Physical Parameters and Operating Conditions

To modify the physical parameters for the components to be modeled, simply double click on the component block. A mask will appear. In the various tabs, both physical and operating parameters of the system can be specified. If the model fails to converge, it might be a result of poor selection of physical parameters and operating conditions. It is recommended that data from an existing experimental system be used to guide the choice of physical parameters and operating conditions.

3.3 Set Model Inputs and Disturbances

For the ports whose values are not set by other components in the system model, input values must be chosen. These values can be as simple as a constant input, or can have a time varying characteristic (as shown by example system 4 in Chapter 4.1.4). Regardless of whether the input signal is static or dynamic, it must be chosen to be representative of the system to be modeled. If the signal deviates significantly from realistic values, it can cause a model to fail to converge.

The standard four component vapor compression system has four inputs: compressor speed, valve aperture, condenser fan speed, and evaporator fan speed. For the THERMOSYS 4.1 models each input should have the following form; the compressor speed input signal in revolutions per minute (RPM), the valve aperture for the polynomial based model in fraction open (values between 0 and 1), the valve aperture for the map based model in percent open (values between 0 and 100), and the heat exchanger fan speeds as the mass flow rate of air (kg/s).

3.4 Run the Simulation

At this point the simulation is ready to be run. Note, in THERMOSYS 4.1, each component is initialized automatically prior to simulation. From the menu bar of the Simulink diagram window, choose *Simulation…Configuration Parameters*. Then select the start and stop time, solver type (fixed or variable step), and the specific solver to be used to integrate the state variables from the pull down menu (ODE23tb is used for the sample systems). Then launch the simulation. Simulation results can be taken from the model through various Simulink blocks (i.e. *scope*, *to workspace*, etc.)

Note, due to the breaking of library links in order to view the specific order and values for the block inputs and outputs, a warning message may be generated when the system is run. This warning can be ignored, and simply informs the user that the blocks are no longer linked to the library.

Chapter 4

Running a Sample System.

4.1 Sample Systems

This tutorial will describe the steps required to simulate a vapor compression cycle by running a sample system in THERMOSYS. In THERMOSYS 4.1.1, we provide 4 different sample systems each using refrigerant R134a. Each sample system demonstrates a different capability of the THERMOSYS toolbox as described below. These sample systems were developed using physical parameters, component maps, and operating conditions based on an experimental vapor compression system owned by the Alleyne Research Group at UIUC.

4.1.1 Example System 1 – Sample_1_TXV_PI_Control.mdl

This sample system consists of a compressor, condenser, thermostatic expansion valve, and an evaporator. A proportional-integral (PI) controller is used to regulate the pressure in the evaporator by varying the compressor speed. This example system demonstrates how the THERMOSYS toolbox can aid in the development of controllers by allowing the user to quickly adjust controller gains and evaluate the performance.

4.1.2 Example System 2 – Sample_2_TXV_Shutdown_Startup.mdl

This sample system consists of a compressor, condenser, thermostatic expansion valve, and an evaporator. This example demonstrates the ability to simulate the large transients that occur during shutdown and startup. From an initial operating condition, the compressor is turned off at 100 seconds and then turned back on at 500 seconds. This example system relies heavily on the mode switching capabilities presented in Reference 1.

4.1.3 Example System 3 – Sample_3_TXV_Box_On_Off_Cycling.mdl

This sample system contains the compressor, condenser, thermostatic expansion valve, evaporator, and the box model which simulates a cooled space with various user-specified heat

loads. This model has the ability to model startup and shutdown, as the temperature inside the box model is regulated using on-off control. The compressor is turned on and off to keep the box temperature within 19-20^{o}C.

4.1.4 Example System 4 – Sample_4_EEV_Shutdown_Startup.mdl

This sample system consists of a compressor, condenser, electronic expansion valve, and an evaporator. During the shutdown and startup processes, where the compressor is turned off and on, the electronic expansion valve is also closed and re-opened.

4.2 Running a Sample System

One of the major improvements from previous versions of THERMOSYS to THERMOSYS 4.1 is the ease of running a simulation. After installing the toolbox, as instructed in Chapter **Error! Reference source not found.**, a user simply opens the desired sample system. The necessary files (FluidProp) will automatically load into the MATLAB workspace. Now the model can be simulated by pressing the *Start Simulation* button. Users are encouraged to add scope blocks to the signals of interest prior to running any of the example systems. Users can change the desired set points or controller gains to develop an intuition for the system and how these models can be used. Users may also wish to double click on the component models and edit the parameters in the component masks in order to simulate different systems or operating conditions. However, users are cautioned that various parameter or operating condition inputs may not be realistic, potentially resulting in the inability for the simulation to converge. These sample systems are intended to provide a new user with a simple introduction to some of the capabilities of THERMOSYS 4.1 and users are encouraged to use these models to gain familiarity with THERMOSYS prior to developing models of new systems or operating conditions.

List of References

[1] McKinley, T. L., Alleyne, A. G., "An advanced nonlinear switched heat exchanger model for vapor compression cycles using the moving-boundary method," International Journal of Refrigeration, vol. 31, no. 7, pp. 1253-1264, 2008.

[2] Reference Fluid Thermodynamic and Transport Properties Database (REFPROP), ver. 9, National Institute of Standards and Technology, [http://www.nist.gov/srd/nist23.cfm].

Appendix A

Performance Mapping Generation

Modeling systems in THERMOSYS requires the generation and use of three maps: valve discharge coefficient, compressor volumetric efficiency, and compressor adiabatic efficiency. Maps are used to go from a specified set of inputs to an output condition. For accurate modeling, maps will need to be re-generated for each new test system. However, sample models can be run with maps for a different physical system.

This appendix discusses criteria for the generation of a data set to be used to create a map, as well as provides sample code to expedite the mapping process. Note that the mapping functions presented in the code are only an example. Additional parameters and higher order terms may be needed to accurately map a given component.

A.1 Data Set for Map Generation

Important factors affecting the map generation are:

- The data set taken from the physical test system used to generate the maps
    - Should try to cover as much of the operating envelope as possible
    - Can specify a desired section of the data using the ‘start’ and ‘stop’ values in the .m files
- The independent variables to be used for the dimensions of the look-up tables such as inlet and outlet pressures and temperature or subcooling, superheat, density, etc.
- The model structure assumed when generating the X matrix (see sample code below). Often just having linear functions of the independent variables is not enough and higher powers and additional cross terms may need to be considered.
- Using a least squares approach, a vector of polynomial coefficients will be generated. Currently, a map is made based on this polynomial, but the polynomial could be coded into the model as well.

- It should be ensured that the bounds of the map cover the range of possible operating conditions
- The choice of grid sizes for the maps is a balance between between accuracy and size of the map. If too fine of a grid is used, the maps may potentially be very large and could significantly slow down the simulation. Default spacing is about 10 values per dimension.
- Save the map generated by the code into the structure that contains all of the maps for that component with a unique name. Then access that name via the drop down menu in the mask of that component in the Simulink model.

A.2 Example Map Generation Code

A.2.1Valve:

```
start = 0;
stop = 100;

% Store the selected range of data in new variables
MDOT = Sensor1(start:stop);
  % MDOT = mass flow rate
PI = Sensor2(start:stop);
  % PI = valve inlet pressure
PO = Sensor3(start:stop);
  % PO = valve outlet pressure
TI = Sensor4(start:stop);
  % TI = valve inlet temperature
TSAT = interp1( FluidProp.TwoPhase.R134a.Psat, FluidProp.TwoPhase.R134a.Tsat, PI);
  % TSAT = saturation temperature at inlet pressure
SUB = TSAT - TI;
SUB = max(SUB,0);
  % SUB = inlet subcooling
EEV = Sensor5(start:stop);
  % EEV = valve command signal
TIME = Sensor6(start:stop);
  % TIME = time values

% Calculate the inlet fluid density, assuming the inlet fluid is saturated liquid
RHO_V = interp2 (FluidProp.TwoPhase.R134a.T, FluidProp.TwoPhase.R134a.P,
FluidProp.TwoPhase.R134a.Rho_pt, TI, PI);
% Calculate the flow coefficient (valve area * discharge coefficient)
CF = MDOT./(sqrt(RHO_V.*(PI-PO)));

% Form input matrix X, assuming a model structure
X = [ones(size(MDOT)) EEV PI SUB.*EEV];
% Find model coefficients using left matrix division (approximate matrix inverse)
a = X\CF;
% Find the flow coefficient values predicted by the model
```

```
CF_model = X*a;
% Find the error (difference between data and model flow coefficient)
error = (CF_model-CF);
% Find the maximum error value, used to evaluate model structure
MaxErr = max(abs(error));
% Find the average error value, used to evaluate model structure
AveErr = mean(error);
% Find the RMS error, used to evaluate model structure
RMS = norm(error)/sqrt(length(MDOT));

% Plot the error as a function of time, used to evaluate model structure
plot(TIME,error);

% Clear the variables used for map generation
clear Pi    % Pi = inlet pressure
clear Po    % Po = outlet pressure
clear eev    % eev = valve command signal
clear Cf_map  % Cf_map = lookup table array

% Define input variable vectors
Pi = 500:50:1000;
eev = 10:5:100;
sub = 0:2:20;

% Generate the lookup table using a for loop for each input, equation should match the
model structure chosen for the X matrix
for c1 = 1:length(eev)
  for c2 = 1:length(Pi)
  for c3 = 1:length(sub)
    Cf_map(c1,c2,c3) = a(1) + a(2).*eev(c1) + a(3).*Pi(c2) ...
      + a(4).*sub(c3)*eev(c1);
  end
  end
end

% Store the input vectors and lookup table array in a structure
ValveProp.Pi = Pi;
ValveProp.u = eev;
ValveProp.sub = sub;
ValveProp.Cf_v = Cf_map;
```

A.2.2 Compressor Adiabatic:

```
start = 0;
stop = 100;

% Store the selected range of data in new variables
RPS = Sensor1(start:stop)./60;
  % RPS = compressor speed in revolutions per second
RPM = Sensor2(start:stop);
  % RPM = compressor speed in revolutions per minute
TKRI = Sensor3(start:stop);
  % TKRI = inlet temperature
TKRO = Sensor4(start:stop);
  % TKRO = outlet temperature
PKI = Sensor5(start:stop);
  % PKI = inlet pressure
PKO = Sensor6(start:stop);
```

```
  % PKO = outlet pressure
TSAT = interp1( FluidProp.TwoPhase.R134a.Psat, FluidProp.TwoPhase.R134a.Tsat, PKI);
  % TSAT = saturated temperature at the inlet
SUP = TKRI - TSAT;
  % SUP = inlet superheat
TSAT2 = interp1( FluidProp.TwoPhase.R134a.Psat, FluidProp.TwoPhase.R134a.Tsat, PKO);
  % TSAT2 = saturated temperature at the outlet
SUP2 = TKRO - TSAT2;
  % SUP2 = outlet superheat
TIME = start:stop;
  % TIME = time values

% Calculate the inlet fluid enthalpy
HIN = interp2 (FluidProp.TwoPhase.R134a.T, FluidProp.TwoPhase.R134a.P,
FluidProp.TwoPhase.R134a.H_pt, TKRI, PKI);
Hg_in = interp1( FluidProp.TwoPhase.R134a.Psat,FluidProp.TwoPhase.R134a.Hg,PKI);
HIN = max(HIN,Hg_in);
% Calculate the outlet fluid enthalpy
HOUT = interp2 (FluidProp.TwoPhase.R134a.T, FluidProp.TwoPhase.R134a.P,
FluidProp.TwoPhase.R134a.H_pt, TKRO, PKO);
Hg_out = interp1 ( FluidProp.TwoPhase.R134a.Psat, FluidProp.TwoPhase.R134a.Hg, PKO);
HOUT = max(HOUT,Hg_out);
% Calculate the inlet fluid entropy
S = interp2 (FluidProp.TwoPhase.R134a.H, FluidProp.TwoPhase.R134a.P,
FluidProp.TwoPhase.R134a.S_ph, HIN, PKI);
% Calculate the isentropic outlet fluid enthalpy
HOUTS = interp2 (FluidProp.TwoPhase.R134a.S, FluidProp.TwoPhase.R134a.P,
FluidProp.TwoPhase.R134a.H_ps, S, PKO);
% Calculate the adiabadic efficiency
ETA_A = (HOUTS - HIN)./(HOUT-HIN);

% Form input matrix X, assuming a model structure
X = [ones(size(PKI)) RPM PKO PKO.*RPM RPM.^2];
% Find model coefficients using left matrix division (approximate matrix inverse)
a = X\ETA_A;
% Find the volumetric efficiency values predicted by the model
ETA_A_model = X*a;
% Find the error (difference between data and model volumetric efficiency)
error = (ETA_A_model-ETA_A);
% Find the maximum error value, used to evaluate model structure
MaxErr = max(abs(error));
% Find the average error value, used to evaluate model structure
AveErr = mean(error);
% Find the RMS error, used to evaluate model structure
rms = norm(error)/sqrt(length(PKI));

% Plot the error as a function of time, used to evaluate model structure
figure;plot(TIME,error)

% Clear the variables used for map generation
clear Pi      % Pi = inlet pressure
clear Po      % Po = outlet pressure
clear rpm      % rpm = compressor speed
clear eta_a_map   % eta_a_map = lookup table array

% Define input variable vectors
Po = 500:100:1200;
Pi = 200:50:500;
rpm = 2000:250:7500;
```

```
% Generate the lookup table using a for loop for each input, equation should match the
model structure chosen for the X matrix
for c1 = 1:length(rpm)
  for c2 = 1:length(Pi)
    for c3 = 1:length(Po)
    eta_a_map(c1,c2,c3) = a(1) + a(2).*rpm(c1)...
    + a(3).*Po(c3)...
    + a(4).*Po(c3)*rpm(c1)+ a(5).*rpm(c1).^2;
    end
  end
end

% Store the input vectors and lookup table array in a structure
CompProp.Pi = Pi;
CompProp.rpm = rpm;
CompProp.Po = Po;
CompProp.eta_a = eta_a_map;
```

A.2.3 Compressor Volumetric:

```
start = 1;
stop = 100;

% Store the selected range of data in new variables
MDOT = Sensor1(start:stop);
  % MDOT = mass flow rate
V = 4.0e-5;
  % V = compressor displacement
RPS = max(Sensor2(start:stop)./60,1);
  % RPS = compressor speed in revolutions per second
RPM = max(Sensor3(start:stop),1);
  % RPM = compressor speed in revolutions per minute
TKRI = Sensor4(start:stop);
  % TKRI = inlet temperature
TKRO = Sensor5(start:stop);
  % TKRO = outlet temperature
PKI = Sensor6(start:stop);
  % PKI = inlet pressure
PKO = Sensor7(start:stop);
  % PKO = outlet pressure
TSAT = interp1( FluidProp.TwoPhase.R134a.Psat, FluidProp.TwoPhase.R134a.Tsat, PKI);
  % TSAT = saturated temperature at the inlet
SUP = TKRI - TSAT;
  % SUP = inlet superheat
TSAT2 = interp1( FluidProp.TwoPhase.R134a.Psat, FluidProp.TwoPhase.R134a.Tsat, PKO);
  % TSAT2 = saturated temperature at the outlet
SUP2 = TKRO -TSAT2;
  % SUP2 = outlet superheat
TIME = start:stop;
  % TIME = time values

% Calculate the inlet fluid density
RHO_K = interp1( FluidProp.TwoPhase.R134a.Psat, FluidProp.TwoPhase.R134a.Rhog, PKI);
% Calculate the volumetric efficiency
ETA_V = MDOT./(V.*RPS.*RHO_K);

% Form input matrix X, assuming a model structure
X = [ones(size(PKI)) RPM PKO PKI PKI.^0.5];
```

```
% Find model coefficients using left matrix division (approximate matrix inverse)
a = X\ETA_V;
% Find the volumetric efficiency values predicted by the model
ETA_V_model = X*a;
% Find the error (difference between data and model volumetric efficiency)
error = (ETA_V_model-ETA_V);
% Find the maximum error value, used to evaluate model structure
MaxErr = max(abs(error));
% Find the average error value, used to evaluate model structure
AveErr = mean(error);
% Find the RMS error, used to evaluate model structure
rms = norm(error)/sqrt(length(PKI));

% Plot the error as a function of time, used to evaluate model structure
plot(TIME,error)

% Clear the variables used for map generation
clear Pi      % Pi = inlet pressure
clear Po      % Po = outlet pressure
clear rpm      % rpm = compressor speed
clear eta_v_map   % eta_v_map = lookup table array

% Define input variable vectors
Po = 500:100:1200;
Pi = 200:50:500;
rpm = 2000:250:7500;

% Generate the lookup table using a for loop for each input, equation should match the
model structure chosen for the X matrix
for c1 = 1:length(rpm)
  for c2 = 1:length(Pi)
    for c3 = 1:length(Po)
    eta_v_map(c1,c2,c3) = a(1) + a(2).*rpm(c1)...
    + a(3).*Po(c3) + a(4).*Pi(c2)...
    + a(5).*Pi(c2)^0.5;
    end
  end
end

% Store the input vectors and lookup table array in a structure
CompProp.Pi = Pi;
CompProp.rpm = rpm;
CompProp.Po = Po;
CompProp.eta_v = eta_v_map;
```