

MassBalanceOpt.jl

September 4, 2025

Contents

Contents	ii
I Introduction	1
1 MassBalanceOpt.jl	2
1.1 Streams	2
1.2 Blocks	2
1.3 Flowsheets	2
1.4 Macros and utility functions	3
II Examples	4
1.5 Mixer, StoicReactor, Separator	5
1.6 Simple Ethylene Plant	14
III API	32
2 MassBalanceOpt.jl	33
2.1 Flowsheets, Streams, and Blocks	33
2.2 Variables and Equations	46
2.3 Printing and Output	50
2.4 Solving Models	53
2.5 Index	53

Part I

Introduction

Chapter 1

MassBalanceOpt.jl

This is a package for formulating mass balance models of chemical processes and solving them with JuMP. The user can build a model from a handful of unit operations ([Mixer](#), [Splitter](#), [Separator](#), [StoicReactor](#), [YieldReactor](#), [MultiYieldReactor](#)) connected by Streams and organized into Flowsheets. The package also provides functions and macros for specifying and working with the model. Familiarity with JuMP is a prerequisite for using this package.

The audience for the package is users of process simulators, particularly those that offer an equation-oriented solution mode, who want to solve optimization models of processes for which energy balances are either unnecessary or so straightforward that they can be formulated directly in JuMP.

JuMP is a general purpose algebraic modeling language. The basic objects in a JuMP model are variables, variable bounds, constraints (referred to here as *equations*), objective functions, and solvers. MassBalanceOpt provides an additional layer of objects specific to process flowsheeting. Those objects are:

1.1 Streams

A [Stream](#) represents a flow of material in a process. A stream contains a total mass flow rate and either a set of mass fractions (if the [StreamBasis](#) = `FRAC`, the default), or component mass flow rates (if the [StreamBasis](#) = `FLOW`). A stream's components are specified as an `OrderedSet` of symbols, usually created with the [@components](#) macro. Streams flow into and out of *blocks*, which are sets of equations that represent mass balance operations. By themselves, streams don't create or contain variables; the blocks do that. Streams can be created with the [@stream](#) or [@streams](#) macros, or by calling the `Stream` constructor directly.

1.2 Blocks

A block is a subtype of [AbstractBlock](#). A block has inlet and outlet streams, and may have other input variables like split fractions or stoichiometric coefficients. A block contains the variables and equations that model a particular unit operation. Blocks can be created with the [@block](#) macro, or by calling the block's constructor. A block is created with some of its variables fixed to default values, so that the block is a self-contained model with the same number of equations and free variables. The inlet stream variables are initially fixed, so the blocks start out in a disconnected state. The function [connect](#) is used to connect a stream that flows out of one block and into another. This is a common practice in an equation-based process simulator.

1.3 Flowsheets

A [Flowsheet](#) is a container for blocks and streams. Every model must have at least one flowsheet. Flowsheets can be created as children of an existing flowsheet, forming a tree structure. A flowsheet has a Julia symbol for

a name. Flowsheet names are embedded in variable and equation names, so flowsheets provide namespaces that allow reuse of block and stream names in different flowsheets. A flowsheet created with the default constructor, `fs = Flowsheet()`, will have the name `:index` and has no parent. A child flowsheet can then be created with `fs_unit1 = Flowsheet(:unit1, fs)`. Variable names in the `:index` flowsheet have no prefix; variables in the `:unit1` flowsheet have the prefix `unit1_` prepended to their names. Thus you can create a stream named `feed` in the `:index` flowsheet and a stream named `feed` in the `:unit1` flowsheet without a variable name collision.

1.4 Macros and utility functions

The package also supplies some macros and utility functions that make it easier to manipulate JuMP models. For example, instead of calling the JuMP function `set_lower_bound` to put a lower bound on a variable, you can use the macro `@set` (or if you want to be more explicit, `@bounds`, which is simply an alias for `@set`), e.g.,

```
@set header_feed_mass < 100_000.0
```

sets the lower bound on variable `header_feed_mass`. You can do several assignments in one `@set` like this:

```
@set begin
    var1 = 1.0
    var2 > 0.0
    1.0 < var3 < 10.0
end
```

See `@set`, `@values`, `@bounds`, `@specs`.

You can print the variables in a block by calling

```
print_vars(block_name)
```

to get a table showing the solution value, bounds, start value, and an indication telling you if the variable is fixed or free. You can use the function

```
print_fixed(model_name)
```

to print only the fixed variables in the model. See [Printing and Output](#).

The package does not supply any functions or macros for creating or manipulating objective functions (except for a single function `eval_obj`).

Part II

Examples

1.5 Mixer, StoicReactor, Separator

This is an example of a simple flowsheet with a Mixer, StoicReactor, and Separator.

Add the package from the Julia REPL in the usual way; type `]` to enter the Pkg REPL mode and run

```
pkg> add MassBalanceOpt
```

To do anything useful you'll also need three other packages:

```
pkg> add JuMP Ipopt OrderedCollections
```

Then import the packages:

```
using MassBalanceOpt, JuMP, Ipopt, OrderedCollections
```

Create a JuMP model and a Flowsheet to hold the blocks and streams:

```
m = Model(Ipopt.Optimizer); fs = Flowsheet();
```

```
Flowsheet(
    name=index    )
```

Suppose we have a feed stream of pure component A that mixes with a recycle stream and is fed to a reactor, where the reaction $A \Rightarrow B + C$ takes place with a specified conversion of component A. The reactor effluent is fed to a distillation column where component C and most of B goes overhead, with unreacted A and a small amount of B going out the bottom and back to the feed header. To create the streams we need to first make some component groups:

```
c_A   = @components A
c_AB  = @components A B
c_BC  = @components B C
c_all = @components A B C
```

```
OrderedCollections.OrderedSet{Symbol} with 3 elements:
 :A
 :B
 :C
```

Now we can create the feed, recycle, and reactor inlet streams:

```
(feed, recycle, rx_in) = @streams begin
    feed, c_A
    recycle, c_AB
    rx_in, c_AB
end
```

```
(Stream(feed, fs=index, basis=FRAC, components=[A]), Stream(recycle, fs=index, basis=FRAC,
↪ components=[A, B]), Stream(rx_in, fs=index, basis=FRAC, components=[A, B]))
```

The feed and recycle streams flow into a Mixer block named feedhdr:

```
feedhdr = @block(feedhdr, Mixer, [feed, recycle], rx_in)
```

```
Mixer(in=[feed, recycle], out=[rx_in])
```

Here's what the model looks like so far:

```
print_model(m)
```

Name	Fix	Value	Lower	Upper	Start
feedhdr_feed_mass	==	1			1
feedhdr_feed_A_massfrac	==	1			1
feedhdr_recycle_mass	==	1			1
feedhdr_recycle_A_massfrac	==	0.5			0.5
feedhdr_recycle_B_massfrac	==	0.5			0.5
feedhdr_rx_in_mass					
feedhdr_rx_in_A_massfrac					
feedhdr_rx_in_B_massfrac					

8 variables

```

feedhdr_total_mass_balance : feedhdr_feed_mass + feedhdr_recycle_mass - feedhdr_rx_in_mass = 0
feedhdr_A_mass_balance : feedhdr_feed_A_massfrac*feedhdr_feed_mass +
↪ feedhdr_recycle_A_massfrac*feedhdr_recycle_mass - feedhdr_rx_in_A_massfrac*feedhdr_rx_in_mass = 0
feedhdr_B_mass_balance : feedhdr_recycle_B_massfrac*feedhdr_recycle_mass -
↪ feedhdr_rx_in_B_massfrac*feedhdr_rx_in_mass = 0
feedhdr_feed_mass = 1
feedhdr_feed_A_massfrac = 1
feedhdr_recycle_mass = 1
feedhdr_recycle_A_massfrac = 0.5
feedhdr_recycle_B_massfrac = 0.5
8 equations

```

The two inlet streams to feedhdr have fixed flow rates and compositions that are set to default values when the block was created. The recycle stream flow rate and composition will eventually become free variables after the rest of the model is built, but right now we need to provide values for the fixed variables:

```

@values begin
  feedhdr_feed_mass      = 10_000.0
  feedhdr_recycle_mass   = 2_000.0
  feedhdr_recycle_A_massfrac = 0.99
  feedhdr_recycle_B_massfrac = 0.01
end
print_fixed(m)

```


Name	Fix	Value	Lower	Upper	Start
----- ---					
feedhdr_feed_mass	==	10000			10000
feedhdr_feed_A_massfrac	==	1			1
feedhdr_recycle_mass	==	2000			2000
feedhdr_recycle_A_massfrac	==	0.99			0.99
feedhdr_recycle_B_massfrac	==	0.01			0.01
5 variables					

Now we can estimate start values for the free variables in block feedhdr:

```
set_start_values(feedhdr)
print_vars(m)
```

Name	Fix	Value	Lower	Upper	Start
----- ---					
feedhdr_feed_mass	==	10000			10000
feedhdr_feed_A_massfrac	==	1			1
feedhdr_recycle_mass	==	2000			2000
feedhdr_recycle_A_massfrac	==	0.99			0.99
feedhdr_recycle_B_massfrac	==	0.01			0.01
feedhdr_rx_in_mass					12000
feedhdr_rx_in_A_massfrac					0.9983333
feedhdr_rx_in_B_massfrac					0.001666667
8 variables					

Now we can create the reactor outlet stream and the reactor block:

```
rx_out = @stream(rx_out, c_all)
rx_stoic = @stoic A => B + C # Reaction stoichiometry
mw = Dict{:A => 30.0, :B => 15.0, :C => 10.0} # Molecular weights
conv = OrderedDict{1 => (c=:A, X=0.8)} # Conversion in reaction 1 (A => B + C)
rx = @block(rx, StoicReactor, rx_in, rx_out, rx_stoic, mw, conv)
print_vars(rx)
```

Name	Fix	Value	Lower	Upper	Start
----- ---					
rx_rx_in_mass	==	1			1
rx_rx_in_A_massfrac	==	0.5			0.5
rx_rx_in_B_massfrac	==	0.5			0.5
rx_rx_out_mass					
rx_rx_out_A_massfrac					
rx_rx_out_B_massfrac					
rx_rx_out_C_massfrac					
rx_rx_in_A_mass					
rx_rx_in_B_mass					
rx_rx_out_A_mass					
rx_rx_out_B_mass					
rx_rx_out_C_mass					
rx_rx_in_A_moles					
rx_rx_in_B_moles					

```

rx_rx_out_A_moles      |           |           |           |
rx_rx_out_B_moles      |           |           |           |
rx_rx_out_C_moles      |           |           |           |
rx_extent_rx_1         |           |           |           |
rx_conv_A_rx_1         ==      0.8|           |           |           | 0.8|
19 variables

```

Notice that the reactor inlet stream mass flow rate and composition are fixed to default values. We need to connect the stream rx_in so that the mass flow rate and composition of stream rx_in in block rx are equal to the mass flow rate and composition of rx_in in block feedhdr. Then we can estimate the start values and solve the model:

```

connect(rx_in)
set_start_values(rx)
@solve
print_vars(m)

```

```

*****
This program contains Ipopt, a library for large-scale nonlinear optimization.
Ipopt is released as open source code under the Eclipse Public License (EPL).
For more information visit https://github.com/coin-or/Ipopt
*****

This is Ipopt version 3.14.19, running with linear solver MUMPS 5.8.1.

Number of nonzeros in equality constraint Jacobian...:    50
Number of nonzeros in inequality constraint Jacobian.:     0
Number of nonzeros in Lagrangian Hessian.....:         7

Total number of variables.....:    21
    variables with only lower bounds:        0
    variables with lower and upper bounds:    0
    variables with only upper bounds:        0
Total number of equality constraints.....:    21
Total number of inequality constraints.....:     0
    inequality constraints with only lower bounds:    0
    inequality constraints with lower and upper bounds: 0
    inequality constraints with only upper bounds:    0

iter   objective    inf_pr  inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
   0   0.0000000e+00  9.09e-13  0.00e+00  -1.0  0.00e+00   -  0.00e+00  0.00e+00   0

Number of Iterations....: 0

               (scaled)                (unscaled)
Objective.....:     0.000000000000000e+00    0.000000000000000e+00
Dual infeasibility.....:     0.000000000000000e+00    0.000000000000000e+00
Constraint violation....:     9.0949470177292824e-13    9.0949470177292824e-13
Variable bound violation:     0.000000000000000e+00    0.000000000000000e+00
Complementarity.....:     0.000000000000000e+00    0.000000000000000e+00
Overall NLP error.....:     9.0949470177292824e-13    9.0949470177292824e-13

```

```

Number of objective function evaluations      = 1
Number of objective gradient evaluations     = 1
Number of equality constraint evaluations     = 1
Number of inequality constraint evaluations   = 0
Number of equality constraint Jacobian evaluations = 1
Number of inequality constraint Jacobian evaluations = 0
Number of Lagrangian Hessian evaluations    = 0
Total seconds in IPOPT                      = 0.044

```

EXIT: Optimal Solution Found.

Name	Fix	Value	Lower	Upper	Start
----- --- ----- ----- ----- -----					
feedhdr_feed_mass	==	10000			10000
feedhdr_feed_A_massfrac	==	1			1
feedhdr_recycle_mass	==	2000			2000
feedhdr_recycle_A_massfrac	==	0.99			0.99
feedhdr_recycle_B_massfrac	==	0.01			0.01
feedhdr_rx_in_mass		12000			12000
feedhdr_rx_in_A_massfrac		0.998333			0.998333
feedhdr_rx_in_B_massfrac		0.00166667			0.00166667
rx_rx_in_mass		12000			12000
rx_rx_in_A_massfrac		0.998333			0.998333
rx_rx_in_B_massfrac		0.00166667			0.00166667
rx_rx_out_mass		10402.67			10402.67
rx_rx_out_A_massfrac		0.2303256			0.2303256
rx_rx_out_B_massfrac		0.4625737			0.4625737
rx_rx_out_C_massfrac		0.3071007			0.3071007
rx_rx_in_A_mass		11980			11980
rx_rx_in_B_mass		20			20
rx_rx_out_A_mass		2396			2396
rx_rx_out_B_mass		4812			4812
rx_rx_out_C_mass		3194.667			3194.667
rx_rx_in_A_moles		399.3333			399.3333
rx_rx_in_B_moles		1.333333			1.333333
rx_rx_out_A_moles		79.86667			79.86667
rx_rx_out_B_moles		320.8			320.8
rx_rx_out_C_moles		319.4667			319.4667
rx_extent_rx_1		319.4667			319.4667
rx_conv_A_rx_1	==	0.8			0.8
27 variables					

The solver converged immediately because the start value estimates were equal to the values at the solution. This won't always be the case. Now we can create the block that models the distillation column:

```

product = @stream(product, c_BC)
col = @block(col, Separator, rx_out, [product, recycle])
print_fixed(col)

```

Name	Fix	Value	Lower	Upper	Start
----- --- ----- ----- ----- -----					
col_rx_out_mass	==	1			1
col_rx_out_A_massfrac	==	0.3333333			0.3333333
col_rx_out_B_massfrac	==	0.3333333			0.3333333
col_rx_out_C_massfrac	==	0.3333333			0.3333333

col_B_recycle_split	==	0.5			0.5
5 variables					

We connect the stream rx_out and set the value of the split fraction of component B into the recycle stream. We don't need to specify split fractions for A or C because the component set specifications force all A into the recycle stream and all C into the product stream.

```
connect(rx_out)
@set col_B_recycle_split = 0.01
set_start_values(col)
@solve
print_vars(col)
```

This is Ipopt version 3.14.19, running with linear solver MUMPS 5.8.1.

```
Number of nonzeros in equality constraint Jacobian...: 99
Number of nonzeros in inequality constraint Jacobian.: 0
Number of nonzeros in Lagrangian Hessian.....: 17

Total number of variables.....: 41
    variables with only lower bounds: 0
    variables with lower and upper bounds: 0
    variables with only upper bounds: 0
Total number of equality constraints.....: 41
Total number of inequality constraints.....: 0
    inequality constraints with only lower bounds: 0
    inequality constraints with lower and upper bounds: 0
    inequality constraints with only upper bounds: 0

iter   objective    inf_pr  inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
   0   0.000000e+00  9.09e-13  0.00e+00  -1.0  0.00e+00   -  0.00e+00  0.00e+00   0
```

Number of Iterations....: 0

	(scaled)	(unscaled)
Objective.....	0.0000000000000000e+00	0.0000000000000000e+00
Dual infeasibility.....	0.0000000000000000e+00	0.0000000000000000e+00
Constraint violation....	9.0949470177292824e-13	9.0949470177292824e-13
Variable bound violation:	0.0000000000000000e+00	0.0000000000000000e+00
Complementarity.....	0.0000000000000000e+00	0.0000000000000000e+00
Overall NLP error.....	9.0949470177292824e-13	9.0949470177292824e-13

Number of objective function evaluations	= 1
Number of objective gradient evaluations	= 1
Number of equality constraint evaluations	= 1
Number of inequality constraint evaluations	= 0
Number of equality constraint Jacobian evaluations	= 1
Number of inequality constraint Jacobian evaluations	= 0
Number of Lagrangian Hessian evaluations	= 0
Total seconds in IPOPT	= 0.000

EXIT: Optimal Solution Found.

Name	Fix	Value	Lower	Upper	Start
------	-----	-------	-------	-------	-------

```

-----|---|-----|-----|-----|
col_rx_out_mass          10402.67|          |          |          |
col_rx_out_A_massfrac    0.2303256|          |          |          |
col_rx_out_B_massfrac    0.4625737|          |          |          |
col_rx_out_C_massfrac    0.3071007|          |          |          |
col_product_mass         7958.547|          |          |          |
col_product_B_massfrac   0.5985867|          |          |          |
col_product_C_massfrac   0.4014133|          |          |          |
col_recycle_mass         2444.12|          |          |          |
col_recycle_A_massfrac    0.9803119|          |          |          |
col_recycle_B_massfrac    0.01968807|          |          |          |
col_rx_out_A_mass        2396|          |          |          |
col_rx_out_B_mass        4812|          |          |          |
col_rx_out_C_mass        3194.667|          |          |          |
col_product_B_mass       4763.88|          |          |          |
col_product_C_mass       3194.667|          |          |          |
col_recycle_A_mass       2396|          |          |          |
col_recycle_B_mass       48.12|          |          |          |
col_A_recycle_split      1|          |          |          |
col_B_product_split      0.99|          |          |          |
col_B_recycle_split      0.01|          |          |          |
col_C_product_split      1|          |          |          |
21 variables

```

We don't really want to fix `col_B_recycle_split` though. A controller will be controlling the mass fraction of B in the recycle stream to a setpoint, so we want to fix the B mass fraction in the recycle stream and free `col_B_recycle_split`. This is called "flipping" the specs on the two variables, and is done like this:

```

@specs col_B_recycle_split ~ col_recycle_B_massfrac
@set col_recycle_B_massfrac = 0.01    # Set the mass fraction of B in the recycle stream
@solve
print_vars(col)

```

This is Ipopt version 3.14.19, running with linear solver MUMPS 5.8.1.

```

Number of nonzeros in equality constraint Jacobian...:    100
Number of nonzeros in inequality constraint Jacobian.:     0
Number of nonzeros in Lagrangian Hessian.....:         17

Total number of variables.....:    41
    variables with only lower bounds:     0
    variables with lower and upper bounds:  0
    variables with only upper bounds:     0
Total number of equality constraints.....:    41
Total number of inequality constraints.....:     0
    inequality constraints with only lower bounds:     0
    inequality constraints with lower and upper bounds:  0
    inequality constraints with only upper bounds:     0

iter   objective    inf_pr  inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
  0   0.0000000e+00  2.37e+01  0.00e+00  -1.0  0.00e+00  -  0.00e+00  0.00e+00  0
  1   0.0000000e+00  2.29e-01  0.00e+00  -1.0  2.39e+01  -  1.00e+00  1.00e+00h  1
  2   0.0000000e+00  9.09e-13  0.00e+00  -3.8  9.48e-05  -  1.00e+00  1.00e+00h  1

```

Number of Iterations.....: 2

	(scaled)	(unscaled)
Objective.....:	0.000000000000000e+00	0.000000000000000e+00
Dual infeasibility.....:	0.000000000000000e+00	0.000000000000000e+00
Constraint violation....:	9.0949470177292824e-13	9.0949470177292824e-13
Variable bound violation:	0.000000000000000e+00	0.000000000000000e+00
Complementarity.....:	0.000000000000000e+00	0.000000000000000e+00
Overall NLP error.....:	9.0949470177292824e-13	9.0949470177292824e-13

Number of objective function evaluations = 3
 Number of objective gradient evaluations = 3
 Number of equality constraint evaluations = 3
 Number of inequality constraint evaluations = 0
 Number of equality constraint Jacobian evaluations = 3
 Number of inequality constraint Jacobian evaluations = 0
 Number of Lagrangian Hessian evaluations = 2
 Total seconds in IPOPT = 0.004

EXIT: Optimal Solution Found.

Name	Fix	Value	Lower	Upper	Start
col_rx_out_mass		10402.67			10402.67
col_rx_out_A_massfrac		0.2303256			0.2303256
col_rx_out_B_massfrac		0.4625737			0.4625737
col_rx_out_C_massfrac		0.3071007			0.3071007
col_product_mass		7982.465			7958.547
col_product_B_massfrac		0.5997894			0.5985867
col_product_C_massfrac		0.4002106			0.4014133
col_recycle_mass		2420.202			2444.12
col_recycle_A_massfrac		0.99			0.9803119
col_recycle_B_massfrac	==	0.01			0.01
col_rx_out_A_mass		2396			2396
col_rx_out_B_mass		4812			4812
col_rx_out_C_mass		3194.667			3194.667
col_product_B_mass		4787.798			4763.88
col_product_C_mass		3194.667			3194.667
col_recycle_A_mass		2396			2396
col_recycle_B_mass		24.20202			48.12
col_A_recycle_split		1			1
col_B_product_split		0.9949705			0.99
col_B_recycle_split		0.005029514			0.01
col_C_product_split		1			1

21 variables

Finally we can connect the recycle stream and solve the complete problem:

```

connect(recycle)
@solve
print_vars(m)

```

This is Ipopt version 3.14.19, running with linear solver MUMPS 5.8.1.

```

Number of nonzeros in equality constraint Jacobian...: 110
Number of nonzeros in inequality constraint Jacobian.: 0
Number of nonzeros in Lagrangian Hessian.....: 19

Total number of variables.....: 44
      variables with only lower bounds: 0
      variables with lower and upper bounds: 0
      variables with only upper bounds: 0
Total number of equality constraints.....: 44
Total number of inequality constraints.....: 0
      inequality constraints with only lower bounds: 0
      inequality constraints with lower and upper bounds: 0
      inequality constraints with only upper bounds: 0

```

```

iter   objective   inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr ls
  0   0.0000000e+00  4.44e+02  0.00e+00  -1.0  0.00e+00  -  0.00e+00  0.00e+00  0
  1   0.0000000e+00  1.11e+00  0.00e+00  -1.0  5.25e+02  -  1.00e+00  1.00e+00h  1
  2   0.0000000e+00  7.15e-05  0.00e+00  -2.5  8.06e-01  -  1.00e+00  1.00e+00h  1
  3   0.0000000e+00  1.82e-12  0.00e+00  -8.6  2.08e-08  -  1.00e+00  1.00e+00h  1

```

Number of Iterations.....: 3

	(scaled)	(unscaled)
Objective.....:	0.0000000000000000e+00	0.0000000000000000e+00
Dual infeasibility.....:	0.0000000000000000e+00	0.0000000000000000e+00
Constraint violation.....:	1.8189894035458565e-12	1.8189894035458565e-12
Variable bound violation:	0.0000000000000000e+00	0.0000000000000000e+00
Complementarity.....:	0.0000000000000000e+00	0.0000000000000000e+00
Overall NLP error.....:	1.8189894035458565e-12	1.8189894035458565e-12

```

Number of objective function evaluations      = 4
Number of objective gradient evaluations      = 4
Number of equality constraint evaluations      = 4
Number of inequality constraint evaluations    = 0
Number of equality constraint Jacobian evaluations = 4
Number of inequality constraint Jacobian evaluations = 0
Number of Lagrangian Hessian evaluations     = 3
Total seconds in IPOPT                       = 0.001

```

EXIT: Optimal Solution Found.

Name	Fix	Value	Lower	Upper	Start
----- --- ----- ----- -----					
feedhdr_feed_mass	==	10000			10000
feedhdr_feed_A_massfrac	==	1			1
feedhdr_recycle_mass		2525.253			2000
feedhdr_recycle_A_massfrac		0.99			0.99
feedhdr_recycle_B_massfrac		0.01			0.01
feedhdr_rx_in_mass		12525.25			12000
feedhdr_rx_in_A_massfrac		0.9979839			0.9983333
feedhdr_rx_in_B_massfrac		0.002016129			0.001666667
rx_rx_in_mass		12525.25			12000
rx_rx_in_A_massfrac		0.9979839			0.9983333
rx_rx_in_B_massfrac		0.002016129			0.001666667
rx_rx_out_mass		10858.59			10402.67
rx_rx_out_A_massfrac		0.2302326			0.2303256

rx_rx_out_B_massfrac		0.4627907			0.4625737
rx_rx_out_C_massfrac		0.3069767			0.3071007
rx_rx_in_A_mass		12500			11980
rx_rx_in_B_mass		25.25253			20
rx_rx_out_A_mass		2500			2396
rx_rx_out_B_mass		5025.253			4812
rx_rx_out_C_mass		3333.333			3194.667
rx_rx_in_A_moles		416.6667			399.3333
rx_rx_in_B_moles		1.683502			1.333333
rx_rx_out_A_moles		83.33333			79.86667
rx_rx_out_B_moles		335.0168			320.8
rx_rx_out_C_moles		333.3333			319.4667
rx_extent_rx_1		333.3333			319.4667
rx_conv_A_rx_1	==	0.8			0.8
col_rx_out_mass		10858.59			10402.67
col_rx_out_A_massfrac		0.2302326			0.2303256
col_rx_out_B_massfrac		0.4627907			0.4625737
col_rx_out_C_massfrac		0.3069767			0.3071007
col_product_mass		8333.333			7958.547
col_product_B_massfrac		0.6			0.5985867
col_product_C_massfrac		0.4			0.4014133
col_recycle_mass		2525.253			2444.12
col_recycle_A_massfrac		0.99			0.9803119
col_recycle_B_massfrac	==	0.01			0.01
col_rx_out_A_mass		2500			2396
col_rx_out_B_mass		5025.253			4812
col_rx_out_C_mass		3333.333			3194.667
col_product_B_mass		5000			4763.88
col_product_C_mass		3333.333			3194.667
col_recycle_A_mass		2500			2396
col_recycle_B_mass		25.25253			48.12
col_A_recycle_split		1			1
col_B_product_split		0.9949749			0.99
col_B_recycle_split		0.005025126			0.01
col_C_product_split		1			1
48 variables					

1.6 Simple Ethylene Plant

This is an example of the use of the MassBalanceOpt package to build a simple model of an ethylene plant and solve it with JuMP. Here is a block diagram of the flowsheet:

Create a JuMP model that uses the Ipopt solver:

```
m = Model(Ipopt.Optimizer)
```

```
A JuMP Model
├ solver: Ipopt
├ objective_sense: FEASIBILITY_SENSE
├ num_variables: 0
├ num_constraints: 0
└ Names registered in the model: none
```

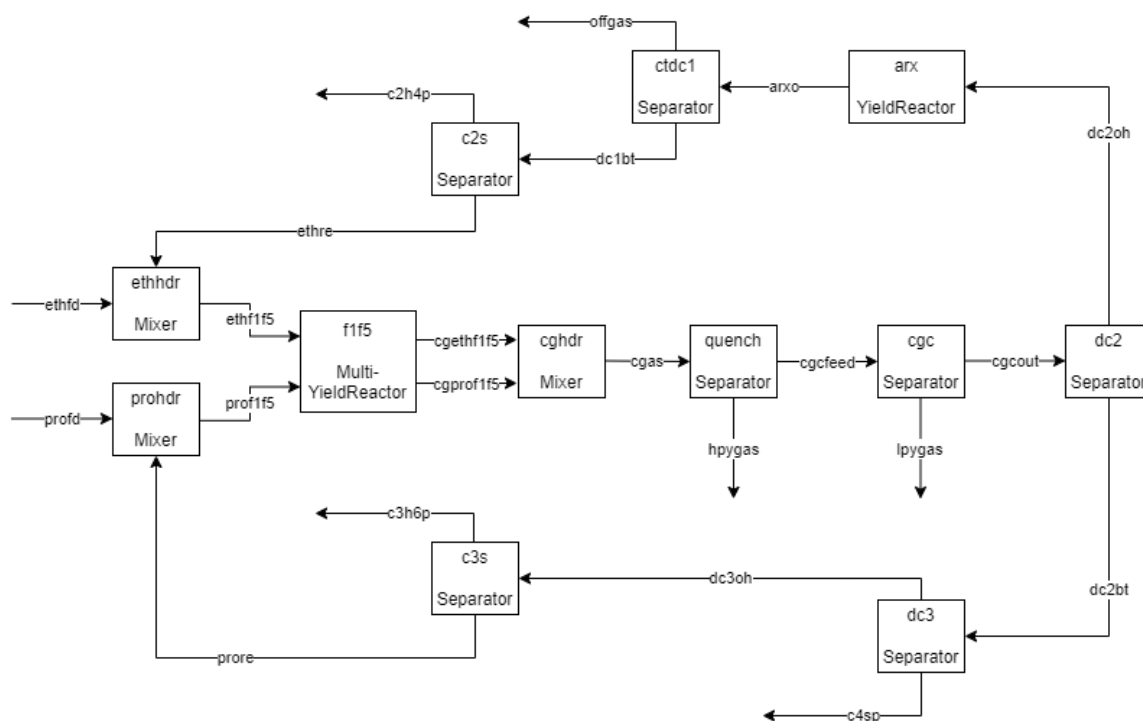



Figure 1.1:

Create a Flowsheet to contain the variables and equations of the model. The default name of a flowsheet is :index. The :index flowsheet does not have a parent flowsheet.

```
fs = Flowsheet()
```

```
Flowsheet(
  name=index )
```

The plant will have two feed streams: ethane and propane, each containing only one component. A component is simply a `Symbol`; it doesn't have any attributes like a chemical formula or molecular weight.

```
comps_eth_feed = @components c2h6
comps_pro_feed = @components c3h8
```

```
OrderedCollections.OrderedSet{Symbol} with 1 element:
:c3h8
```

The feed streams are mixed with ethane and propane recycle streams. The component sets for the recycle streams are:

```
comps_eth_rec = @components c2h4 c2h6
comps_pro_rec = @components c3h6 c3h8 mapd
```

```
OrderedCollections.OrderedSet{Symbol} with 3 elements:
```

```
:c3h6
:c3h8
:mapd
```

The fresh feed streams are mixed with their recycle streams in Mixer blocks. These blocks need streams for the fresh feeds, recycles, and the mixed feed/recycle:

```
(ethfd, ethre, ethf1f5, profd, prore, prof1f5) = @streams begin
  ethfd , comps_eth_feed # Ethane fresh feed
  ethre , comps_eth_rec  # Ethane recycle
  ethf1f5 , comps_eth_rec # Mixed fresh ethane feed and ethane recycle to furnaces F-1
  ↪ through F-5.
  profd , comps_pro_feed # Propane fresh feed
  prore , comps_pro_rec  # Propane recycle
  prof1f5 , comps_pro_rec # Mixed fresh propane feed and propane recycle to furnaces F-1
  ↪ through F-5.
end
```

```
(Stream(ethfd, fs=index, basis=FRAC, components=[c2h6]), Stream(ethre, fs=index, basis=FRAC,
↪ components=[c2h4, c2h6]), Stream(ethf1f5, fs=index, basis=FRAC, components=[c2h4, c2h6]),
↪ Stream(profd, fs=index, basis=FRAC, components=[c3h8]), Stream(prore, fs=index, basis=FRAC,
↪ components=[c3h6, c3h8, mapd]), Stream(prof1f5, fs=index, basis=FRAC, components=[c3h6, c3h8,
↪ mapd]))
```

Create Mixer blocks that represent the ethane and propane feed headers:

```
ethhdr = @block(ethhdr, Mixer, [ethfd, ethre], ethf1f5)
prohdr = @block(prohdr, Mixer, [profd, prore], prof1f5)
```

```
Mixer(in=[profd, prore], out=[prof1f5])
```

Print the equations and variables in the model so far:

```
print_model(m)
```

Name	Fix	Value	Lower	Upper	Start
----- --- ----- ----- -----					
ethhdr_ethfd_mass	==	1			1
ethhdr_ethfd_c2h6_massfrac	==	1			1
ethhdr_ethre_mass	==	1			1
ethhdr_ethre_c2h4_massfrac	==	0.5			0.5
ethhdr_ethre_c2h6_massfrac	==	0.5			0.5
ethhdr_ethf1f5_mass					
ethhdr_ethf1f5_c2h4_massfrac					
ethhdr_ethf1f5_c2h6_massfrac					
prohdr_profd_mass	==	1			1
prohdr_profd_c3h8_massfrac	==	1			1

```

prohdr_prore_mass          ==          1|          |          |          1|
prohdr_prore_c3h6_massfrac ==      0.333333|          |          |      0.333333|
prohdr_prore_c3h8_massfrac ==      0.333333|          |          |      0.333333|
prohdr_prore_mapd_massfrac ==      0.333333|          |          |      0.333333|
prohdr_prof1f5_mass        |          |          |          |
prohdr_prof1f5_c3h6_massfrac |          |          |          |
prohdr_prof1f5_c3h8_massfrac |          |          |          |
prohdr_prof1f5_mapd_massfrac |          |          |          |
18 variables

ethhdr_total_mass_balance : ethhdr_ethfd_mass + ethhdr_ethre_mass - ethhdr_ethf1f5_mass = 0
prohdr_total_mass_balance : prohdr_profd_mass + prohdr_prore_mass - prohdr_prof1f5_mass = 0
ethhdr_c2h4_mass_balance : ethhdr_ethre_c2h4_massfrac*ethhdr_ethre_mass -
↳ ethhdr_ethf1f5_c2h4_massfrac*ethhdr_ethf1f5_mass = 0
ethhdr_c2h6_mass_balance : ethhdr_ethfd_c2h6_massfrac*ethhdr_ethfd_mass +
↳ ethhdr_ethre_c2h6_massfrac*ethhdr_ethre_mass - ethhdr_ethf1f5_c2h6_massfrac*ethhdr_ethf1f5_mass =
↳ 0
prohdr_c3h6_mass_balance : prohdr_prore_c3h6_massfrac*prohdr_prore_mass -
↳ prohdr_prof1f5_c3h6_massfrac*prohdr_prof1f5_mass = 0
prohdr_c3h8_mass_balance : prohdr_profd_c3h8_massfrac*prohdr_profd_mass +
↳ prohdr_prore_c3h8_massfrac*prohdr_prore_mass - prohdr_prof1f5_c3h8_massfrac*prohdr_prof1f5_mass =
↳ 0
prohdr_mapd_mass_balance : prohdr_prore_mapd_massfrac*prohdr_prore_mass -
↳ prohdr_prof1f5_mapd_massfrac*prohdr_prof1f5_mass = 0
ethhdr_ethfd_mass = 1
ethhdr_ethfd_c2h6_massfrac = 1
ethhdr_ethre_mass = 1
ethhdr_ethre_c2h4_massfrac = 0.5
ethhdr_ethre_c2h6_massfrac = 0.5
prohdr_profd_mass = 1
prohdr_profd_c3h8_massfrac = 1
prohdr_prore_mass = 1
prohdr_prore_c3h6_massfrac = 0.3333333333333333
prohdr_prore_c3h8_massfrac = 0.3333333333333333
prohdr_prore_mapd_massfrac = 0.3333333333333333
18 equations

```

Note that the feed stream flow rates and compositions are fixed at default values. The recycle stream flow rates and compositions are also fixed for now; that will be changed after the rest of the model has been built. Set the values of the fixed variables to more realistic values:

```

@values begin
  ethhdr_ethfd_mass = 80_000.0
  prohdr_profd_mass = 34_000.0
  ethhdr_ethre_mass = 39_000.0
  prohdr_prore_mass = 5_000.0

  ethhdr_ethre_c2h4_massfrac = 0.005
  ethhdr_ethre_c2h6_massfrac = 0.995
  prohdr_prore_c3h6_massfrac = 0.005
  prohdr_prore_c3h8_massfrac = 0.96
  prohdr_prore_mapd_massfrac = 0.035
end

```

The function `set_start_values` will calculate initial guesses for the free variables:

```
set_start_values([ethhdr, prohdr])
print_vars(m)
```

Name	Fix	Value	Lower	Upper	Start
----- --- ----- ----- ----- -----					
ethhdr_ethfd_mass	==	80000			80000
ethhdr_ethfd_c2h6_massfrac	==	1			1
ethhdr_ethre_mass	==	39000			39000
ethhdr_ethre_c2h4_massfrac	==	0.005			0.005
ethhdr_ethre_c2h6_massfrac	==	0.995			0.995
ethhdr_ethflf5_mass					119000
ethhdr_ethflf5_c2h4_massfrac					0.001638655
ethhdr_ethflf5_c2h6_massfrac					0.9983613
prohdr_profd_mass	==	34000			34000
prohdr_profd_c3h8_massfrac	==	1			1
prohdr_prore_mass	==	5000			5000
prohdr_prore_c3h6_massfrac	==	0.005			0.005
prohdr_prore_c3h8_massfrac	==	0.96			0.96
prohdr_prore_mapd_massfrac	==	0.035			0.035
prohdr_proflf5_mass					39000
prohdr_proflf5_c3h6_massfrac					0.0006410256
prohdr_proflf5_c3h8_massfrac					0.9948718
prohdr_proflf5_mapd_massfrac					0.004487179
18 variables					

Solve the model:

```
@solve
```

This is Ipopt version 3.14.19, running with linear solver MUMPS 5.8.1.

```
Number of nonzeros in equality constraint Jacobian...: 12
Number of nonzeros in inequality constraint Jacobian.: 0
Number of nonzeros in Lagrangian Hessian.....: 5

Total number of variables.....: 7
    variables with only lower bounds: 0
    variables with lower and upper bounds: 0
    variables with only upper bounds: 0
Total number of equality constraints.....: 7
Total number of inequality constraints.....: 0
    inequality constraints with only lower bounds: 0
    inequality constraints with lower and upper bounds: 0
    inequality constraints with only upper bounds: 0

iter   objective    inf_pr  inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
   0   0.0000000e+00  0.00e+00  0.00e+00  -1.0  0.00e+00   -  0.00e+00  0.00e+00   0

Number of Iterations.....: 0

                                (scaled)                                (unscaled)
Objective.....: 0.0000000000000000e+00  0.0000000000000000e+00
Dual infeasibility.....: 0.0000000000000000e+00  0.0000000000000000e+00
```

```

Constraint violation....: 0.0000000000000000e+00 0.0000000000000000e+00
Variable bound violation: 0.0000000000000000e+00 0.0000000000000000e+00
Complementarity.....: 0.0000000000000000e+00 0.0000000000000000e+00
Overall NLP error.....: 0.0000000000000000e+00 0.0000000000000000e+00

```

```

Number of objective function evaluations      = 1
Number of objective gradient evaluations     = 1
Number of equality constraint evaluations     = 1
Number of inequality constraint evaluations   = 0
Number of equality constraint Jacobian evaluations = 1
Number of inequality constraint Jacobian evaluations = 0
Number of Lagrangian Hessian evaluations    = 0
Total seconds in IPOPT                      = 0.000

```

EXIT: Optimal Solution Found.

The solver converged immediately because `set_start_values` is able to calculate the values of the free variables exactly, in this case.

The mixed feed and recycle streams are fed to the cracking furnaces, which are reactors that thermally crack the feeds into a mixture called "cracked gas." In this simple model the cracked gas will consist of the following components:

```
comps_cg = @components h2 ch4 c2h2 c2h4 c2h6 c3h6 c3h8 mapd c4s pygas
```

```
OrderedCollections.OrderedSet{Symbol} with 10 elements:
```

```

:h2
:ch4
:c2h2
:c2h4
:c2h6
:c3h6
:c3h8
:mapd
:c4s
:pygas

```

The ethane and propane feeds are cracked in separate furnaces. The purpose of the model is to calculate the optimal flow rates of ethane and propane feed, given constraints on furnace capacity. The cracked gas streams leaving the ethane and propane furnaces are:

```

(cgethf1f5, cgprof1f5) = @streams begin
    cgethf1f5 , comps_cg, (basis=FLOW)
    cgprof1f5 , comps_cg, (basis=FLOW)
end

```

```

(Stream(cgethf1f5, fs=index, basis=FLOW, components=[h2, ch4, c2h2, c2h4, c2h6, c3h6, c3h8, mapd,
↪ c4s, pygas]), Stream(cgprof1f5, fs=index, basis=FLOW, components=[h2, ch4, c2h2, c2h4, c2h6,
↪ c3h6, c3h8, mapd, c4s, pygas]))

```

The cracked gas streams a FLOW basis because their mass flow rates may be zero at the solution, which would make the equations based on a mass fraction formulation become singular. The furnaces are modeled with a MultiYieldReactor with two feeds: :eth and :pro.

```
f1f5 = @block(f1f5, MultiYieldReactor, [ethf1f5, prof1f5], [cgethf1f5, cgprof1f5], [:eth, :pro],
↳ :furn)
```

```
MultiYieldReactor(in=[ethf1f5, prof1f5], out=[cgethf1f5, cgprof1f5])
```

The values of the fixed variables in the furnace model are:

```
print_fixed(f1f5)
```

Name	Fix	Value	Lower	Upper	Start
----- ---			-----	-----	-----
f1f5_ethf1f5_mass	==	1			1
f1f5_ethf1f5_c2h4_massfrac	==	0.5			0.5
f1f5_ethf1f5_c2h6_massfrac	==	0.5			0.5
f1f5_prof1f5_mass	==	1			1
f1f5_prof1f5_c3h6_massfrac	==	0.333333			0.333333
f1f5_prof1f5_c3h8_massfrac	==	0.333333			0.333333
f1f5_prof1f5_mapd_massfrac	==	0.333333			0.333333
f1f5_eth_rate	==	1			1
f1f5_pro_rate	==	1			1
f1f5_eth_y_h2_from_c2h4	==	0			0
f1f5_eth_y_ch4_from_c2h4	==	0			0
f1f5_eth_y_c2h2_from_c2h4	==	0			0
f1f5_eth_y_c2h6_from_c2h4	==	0			0
f1f5_eth_y_c3h6_from_c2h4	==	0			0
f1f5_eth_y_c3h8_from_c2h4	==	0			0
f1f5_eth_y_mapd_from_c2h4	==	0			0
f1f5_eth_y_c4s_from_c2h4	==	0			0
f1f5_eth_y_pygas_from_c2h4	==	0			0
f1f5_eth_y_h2_from_c2h6	==	0			0
f1f5_eth_y_ch4_from_c2h6	==	0			0
f1f5_eth_y_c2h2_from_c2h6	==	0			0
f1f5_eth_y_c2h4_from_c2h6	==	0			0
f1f5_eth_y_c3h6_from_c2h6	==	0			0
f1f5_eth_y_c3h8_from_c2h6	==	0			0
f1f5_eth_y_mapd_from_c2h6	==	0			0
f1f5_eth_y_c4s_from_c2h6	==	0			0
f1f5_eth_y_pygas_from_c2h6	==	0			0
f1f5_pro_y_h2_from_c3h6	==	0			0
f1f5_pro_y_ch4_from_c3h6	==	0			0
f1f5_pro_y_c2h2_from_c3h6	==	0			0
f1f5_pro_y_c2h4_from_c3h6	==	0			0
f1f5_pro_y_c2h6_from_c3h6	==	0			0
f1f5_pro_y_c3h8_from_c3h6	==	0			0
f1f5_pro_y_mapd_from_c3h6	==	0			0
f1f5_pro_y_c4s_from_c3h6	==	0			0
f1f5_pro_y_pygas_from_c3h6	==	0			0
f1f5_pro_y_h2_from_c3h8	==	0			0
f1f5_pro_y_ch4_from_c3h8	==	0			0

```

f1f5_pro_y_c2h2_from_c3h8 == 0| | | 0|
f1f5_pro_y_c2h4_from_c3h8 == 0| | | 0|
f1f5_pro_y_c2h6_from_c3h8 == 0| | | 0|
f1f5_pro_y_c3h6_from_c3h8 == 0| | | 0|
f1f5_pro_y_mapd_from_c3h8 == 0| | | 0|
f1f5_pro_y_c4s_from_c3h8 == 0| | | 0|
f1f5_pro_y_pygas_from_c3h8 == 0| | | 0|
f1f5_pro_y_h2_from_mapd == 0| | | 0|
f1f5_pro_y_ch4_from_mapd == 0| | | 0|
f1f5_pro_y_c2h2_from_mapd == 0| | | 0|
f1f5_pro_y_c2h4_from_mapd == 0| | | 0|
f1f5_pro_y_c2h6_from_mapd == 0| | | 0|
f1f5_pro_y_c3h6_from_mapd == 0| | | 0|
f1f5_pro_y_c3h8_from_mapd == 0| | | 0|
f1f5_pro_y_c4s_from_mapd == 0| | | 0|
f1f5_pro_y_pygas_from_mapd == 0| | | 0|
54 variables

```

The values of the fixed inlet stream variables will become free after the streams are connected (see below). The rest of the fixed variables are inputs to the model that need to be set. The single-pass yields in the furnaces are:

```

@values begin
  f1f5_eth_y_h2_from_c2h6 = 0.0411
  f1f5_eth_y_ch4_from_c2h6 = 0.05
  f1f5_eth_y_c2h2_from_c2h6 = 0.003
  f1f5_eth_y_c2h4_from_c2h6 = 0.495
  f1f5_eth_y_c3h6_from_c2h6 = 0.0043
  f1f5_eth_y_c3h8_from_c2h6 = 0.01
  f1f5_eth_y_mapd_from_c2h6 = 0.0002
  f1f5_eth_y_c4s_from_c2h6 = 0.03
  f1f5_eth_y_pygas_from_c2h6 = 0.02

  f1f5_eth_y_h2_from_c2h4 = -0.009
  f1f5_eth_y_ch4_from_c2h4 = 0.129
  f1f5_eth_y_c2h2_from_c2h4 = 0.004
  f1f5_eth_y_c2h6_from_c2h4 = 0.0
  f1f5_eth_y_c3h6_from_c2h4 = 0.003
  f1f5_eth_y_c3h8_from_c2h4 = 0.002
  f1f5_eth_y_mapd_from_c2h4 = 0.0
  f1f5_eth_y_c4s_from_c2h4 = 0.05
  f1f5_eth_y_pygas_from_c2h4 = 0.15

  f1f5_pro_y_h2_from_c3h8 = 0.01
  f1f5_pro_y_ch4_from_c3h8 = 0.21
  f1f5_pro_y_c2h2_from_c3h8 = 0.004
  f1f5_pro_y_c2h4_from_c3h8 = 0.35
  f1f5_pro_y_c2h6_from_c3h8 = 0.04
  f1f5_pro_y_c3h6_from_c3h8 = 0.16
  f1f5_pro_y_mapd_from_c3h8 = 0.004
  f1f5_pro_y_c4s_from_c3h8 = 0.015
  f1f5_pro_y_pygas_from_c3h8 = 0.025

  f1f5_pro_y_h2_from_c3h6 = -0.0035
  f1f5_pro_y_ch4_from_c3h6 = 0.15

```

```

f1f5_pro_y_c2h2_from_c3h6 = 0.005
f1f5_pro_y_c2h4_from_c3h6 = 0.04
f1f5_pro_y_c2h6_from_c3h6 = 0.004
f1f5_pro_y_c3h8_from_c3h6 = 0.0000
f1f5_pro_y_mapd_from_c3h6 = 0.002
f1f5_pro_y_c4s_from_c3h6 = 0.03
f1f5_pro_y_pygas_from_c3h6 = 0.15

f1f5_pro_y_c3h6_from_mapd = 1.0
end

```

The flow rates of ethane and propane feed per furnace are:

```

@values begin
  f1f5_eth_rate = 30_000.0
  f1f5_pro_rate = 40_000.0
end

```

Now that most of the fixed variables have been set, the ethf1f5 and prof1f5 inlet streams can be connected, which will free the inlet stream variables in the f1f5 block and copy the start values from the upstream Mixer blocks into the f1f5 block. The set_start_values function can then calculate the rest of the start values:

```

connect([ethf1f5, prof1f5])
set_start_values(f1f5)
@solve

```

This is Ipopt version 3.14.19, running with linear solver MUMPS 5.8.1.

```

Number of nonzeros in equality constraint Jacobian...:    153
Number of nonzeros in inequality constraint Jacobian.:      0
Number of nonzeros in Lagrangian Hessian.....:         15

Total number of variables.....:    50
   variables with only lower bounds:      0
   variables with lower and upper bounds:  0
   variables with only upper bounds:      0
Total number of equality constraints.....:    50
Total number of inequality constraints.....:      0
   inequality constraints with only lower bounds:      0
   inequality constraints with lower and upper bounds:  0
   inequality constraints with only upper bounds:      0

iter   objective    inf_pr  inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
   0   0.0000000e+00  2.84e-14  0.00e+00  -1.0  0.00e+00   -  0.00e+00  0.00e+00   0

Number of Iterations.....: 0

                                (scaled)                                (unscaled)
Objective.....:     0.0000000000000000e+00    0.0000000000000000e+00
Dual infeasibility.....:  0.0000000000000000e+00    0.0000000000000000e+00
Constraint violation....:  2.8421709430404007e-14    2.8421709430404007e-14
Variable bound violation:  0.0000000000000000e+00    0.0000000000000000e+00
Complementarity.....:  0.0000000000000000e+00    0.0000000000000000e+00

```



```

Overall NLP error.....:  2.8421709430404007e-14    2.8421709430404007e-14

Number of objective function evaluations      = 1
Number of objective gradient evaluations     = 1
Number of equality constraint evaluations     = 1
Number of inequality constraint evaluations   = 0
Number of equality constraint Jacobian evaluations = 1
Number of inequality constraint Jacobian evaluations = 0
Number of Lagrangian Hessian evaluations    = 0
Total seconds in IPOPT                      = 0.000

EXIT: Optimal Solution Found.

```

The cracked gas streams leaving the furnaces are mixed together in the cracked gas header. The same pattern is used to add additional blocks to the model:

1. Define component groups
2. Create inlet and outlet streams
3. Create the block
4. Connect the block's inlet streams to the upstream block
5. Configure variable specifications if necessary
6. Set values of the fixed variables
7. Call `set_start_values`
8. Solve the model and print variables if desired

To keep the output as concise as possible we'll avoid solving the model repeatedly in this example. In practice, a solve is usually done after adding each new block.

```

# Cracked gas header.
cgas = @stream(cgas, comps_cg)
cghdr = @block(cghdr, Mixer, [cgeth1f5, cgprof1f5], cgas)
connect([cgeth1f5, cgprof1f5])
set_start_values(cghdr)

```

The cracked gas flows to the quench section, where heavy pygas is separated out as a product. In this simple model, there's only one pygas component, so we'll pretend that 30% of it is heavy pygas. The outlet stream from the quench section flows to the cracked gas compressor section, where the rest of the pygas is removed.

```

# Quench section.
comps_pg = @components pygas
(hpygas, cgcfeed) = @streams begin
    hpygas , comps_pg
    cgcfeed , comps_cg
end
quench = @block(quench, Separator, cgas, [hpygas, cgcfeed])
@set quench_pygas_hpygas_split = 0.3

```

```

connect(cgas)

# Cracked gas compressor section.
comps_pygas = @components pygas
comps_DC2   = @components h2 ch4 c2h2 c2h4 c2h6 c3h6 c3h8 mapd c4s
(lpygas, cgcout) = @streams begin
    lpygas , comps_pygas
    cgcout , comps_DC2
end
cgc = @block(cgc, Separator, cgcfeed, [lpygas, cgcout])
connect(cgcfeed)
set_start_values([quench, cgc])

```

The cracked gas compressor section outlet stream flows to the front-end deethanizer, which splits it into C2-overhead and C3+ bottoms streams.

```

# Front-end deethanizer.
comps_DC2oh = @components h2 ch4 c2h2 c2h4 c2h6
comps_DC2bt = @components c3h6 c3h8 mapd c4s
(dc2oh, dc2bt) = @streams begin
    dc2oh , comps_DC2oh
    dc2bt , comps_DC2bt
end
dc2 = @block(dc2, Separator, cgcout, [dc2oh, dc2bt])
connect(cgcout)
set_start_values(dc2)

```

The overhead stream from the deethanizer flows to the acetylene reactors, which convert all of the acetylene into ethylene and ethane.

```

# Acetylene reactors.
comps_ARX = @components h2 ch4 c2h4 c2h6
arxo = @stream(arxo, comps_ARX)
arx_stoic = @stoic begin # Reaction stoichiometry
    c2h2 + h2 => c2h4
    c2h2 + 2h2 => c2h6
end

arx_mw = Dict{:c2h2 => 26.03728, # Molecular weights of the reacting components
              :h2   => 2.01588,
              :c2h4 => 28.05316,
              :c2h6 => 30.06904)

arx_conv = OrderedDict{1 => (c=:c2h2, X=0.7)} # C2H2 conversion in reaction 1.
# Don't need to specify C2H2 conversion in reaction
# 2, because
# C2H2 is not present in the outlet stream component
# list.

arx = @block(arx, StoicReactor, dc2oh, arxo, arx_stoic, arx_mw, arx_conv)
connect(dc2oh)
set_start_values(arx)

```

The acetylene reactor outlet stream flows to the cold train/demethanizer. The offgas, containing H₂, CH₄, and a small amount of C₂H₄, is separated out as a product.

```

# Cold train/demethanizer.
comps_0G = @components h2 ch4 c2h4
comps_C2S = @components c2h4 c2h6
(offgas, dc1bt) = @streams begin
    offgas , comps_0G
    dc1bt , comps_C2S
end
ctdc1 = @block(ctdc1, Separator, arxo, [offgas, dc1bt])
connect(arxo)

# The split fraction of C2H4 in the offgas starts out as a fixed variable. The offgas C2H4 mass
↪ fraction is free.
# To calculate good start values, we have to set the value of the split fraction.
@set ctdc1_c2h4_offgas_split = 0.001
set_start_values(ctdc1)

# Now we flip the specs, which frees the offgas C2H4 split fraction and fixes the offgas C2H4 mass
↪ fraction.
# We need to set the offgas C2H4 mass fraction to the desired value.
@specs(ctdc1_c2h4_offgas_split ~ ctdc1_offgas_c2h4_massfrac)
@set ctdc1_offgas_c2h4_massfrac = 0.005

```

The demethanizer bottoms stream flows to the C2 splitter, which splits it into an ethylene product stream and a recycle stream to the ethane header. The recycle stream is mostly ethane with a small amount of ethylene mixed in.

```

# C2 splitter.
c2h4p = @stream(c2h4p, comps_C2S)
c2s = @block(c2s, Separator, dc1bt, [c2h4p, ethre])

# Since c2s_c2h4_c2h4p_split is the first split frac, it's fixed by default. But we prefer to fix
# c2s_c2h4_ethre_split instead, so flip the specs on those two variables.
@specs c2s_c2h4_c2h4p_split ~ c2s_c2h4_ethre_split

# Assign values to the fixed split fracs and set the start values.
@values begin
    c2s_c2h6_c2h4p_split = 0.001
    c2s_c2h4_ethre_split = 0.003
end
connect([dc1bt, ethre])
set_start_values(c2s)

# Flip the specs to fix the mass fractions and free the split fracs.
@specs begin
    c2s_c2h4_ethre_split ~ c2s_ethre_c2h4_massfrac
    c2s_c2h4p_c2h6_massfrac ~ c2s_c2h6_c2h4p_split
end

# Set the values of the fixed mass fractions.
@values begin
    c2s_c2h4p_c2h6_massfrac = 0.0008
    c2s_ethre_c2h4_massfrac = 0.005
end

```

The deethanizer bottoms flows to the depropanizer, which splits it into C3- overhead and C4 bottoms streams.

```

# Depropanizer.
comps_DC3oh  = @components  c3h6 c3h8 mapd
comps_DC3bt  = @components  c4s
(dc3oh, c4sp) = @streams begin
    dc3oh  , comps_DC3oh
    c4sp   , comps_DC3bt
end

# The separation is assumed to be clean, so all the split fracs are 1.
dc3 = @block(dc3, Separator, dc2bt, [dc3oh, c4sp])
connect(dc2bt)
set_start_values(dc3)

```

The depropanizer overhead C3- stream is fed to the C3 splitter, which produces a propylene product stream and a propane recycle stream.

```

# C3 splitter.
comps_C3Soh  = @components  c3h6 c3h8
c3h6p = @stream(c3h6p, comps_C3Soh)
c3s = @block(c3s, Separator, dc3oh, [c3h6p, prore])
connect(dc3oh)

# Fix c3s_c3h8_c3h6p_split and free c3s_c3h8_prore_split, and set the fixed split fractions.
@specs c3s_c3h8_c3h6p_split ~ c3s_c3h8_prore_split
@values begin
    c3s_c3h6_prore_split = 0.003
    c3s_c3h8_c3h6p_split = 0.00017
end

# Calculate the start values.
set_start_values(c3s)

# Flip the split fracs with the mass fractions, and set the mass fractions.
@specs begin
    c3s_prore_c3h6_massfrac ~ c3s_c3h6_prore_split
    c3s_c3h6p_c3h8_massfrac ~ c3s_c3h8_c3h6p_split
end
@set c3s_c3h6p_c3h8_massfrac = 0.0001
@set c3s_prore_c3h6_massfrac = 0.005

# Connect the propane recycle, and solve the model.
connect(prore)
@solve

```

This is Ipopt version 3.14.19, running with linear solver MUMPS 5.8.1.

Number of nonzeros in equality constraint Jacobian...	954
Number of nonzeros in inequality constraint Jacobian..	0
Number of nonzeros in Lagrangian Hessian.....	171
Total number of variables.....	362
variables with only lower bounds:	0
variables with lower and upper bounds:	0
variables with only upper bounds:	0
Total number of equality constraints.....	362

```

Total number of inequality constraints.....:      0
      inequality constraints with only lower bounds:      0
      inequality constraints with lower and upper bounds:      0
      inequality constraints with only upper bounds:      0

iter   objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
  0   0.0000000e+00  3.88e+03  0.00e+00  -1.0  0.00e+00  -  0.00e+00  0.00e+00  0
  1   0.0000000e+00  6.80e+01  0.00e+00  -1.0  1.08e+04  -  1.00e+00  1.00e+00h  1
  2   0.0000000e+00  7.57e-02  0.00e+00  -2.5  7.49e+01  -  1.00e+00  1.00e+00h  1
  3   0.0000000e+00  2.91e-11  0.00e+00  -5.7  4.46e-06  -  1.00e+00  1.00e+00h  1

Number of Iterations.....: 3

                                (scaled)                                (unscaled)
Objective.....:  0.0000000000000000e+00  0.0000000000000000e+00
Dual infeasibility.....:  0.0000000000000000e+00  0.0000000000000000e+00
Constraint violation.....:  2.9103830456733704e-11  2.9103830456733704e-11
Variable bound violation:  0.0000000000000000e+00  0.0000000000000000e+00
Complementarity.....:  0.0000000000000000e+00  0.0000000000000000e+00
Overall NLP error.....:  2.9103830456733704e-11  2.9103830456733704e-11

Number of objective function evaluations      = 4
Number of objective gradient evaluations      = 4
Number of equality constraint evaluations      = 4
Number of inequality constraint evaluations    = 0
Number of equality constraint Jacobian evaluations = 4
Number of inequality constraint Jacobian evaluations = 0
Number of Lagrangian Hessian evaluations     = 3
Total seconds in IPOPT                       = 0.001

EXIT: Optimal Solution Found.

```

Update all the start values in the model.

```
set_start_values(m)
```

Define a set of prices and an objective function:

```

# Prices and objective function.
prices = Dict{
  :ethane_feed => 10.0,
  :propane_feed => 15.0,
  :c2h4_prod   => 40.0,
  :c3h6_prod   => 30.0,
  :c4s_prod    => 28.0,
  :pygas_prod  => 23.0,
  :offgas_prod => 25.0
}

map!(p -> p/100.0, values(prices)) # Convert from cents/lb to $/lb

costs = @expression(m, m[:ethhdr_ethfd_mass] * prices[:ethane_feed] +
                      m[:prohdr_profd_mass] * prices[:propane_feed])
sales = @expression(m, m[:c2s_c2h4p_mass] * prices[:c2h4_prod] +

```

```

        m[:c3s_c3h6p_mass] * prices[:c3h6_prod] +
        m[:dc3_c4sp_mass] * prices[:c4s_prod] +
        m[:quench_hpygas_mass] * prices[:pygas_prod] +
        m[:cgc_lpygas_mass] * prices[:pygas_prod] +
        m[:ctdc1_offgas_mass] * prices[:offgas_prod])
@objective(m, Max, sales - costs);

```

$0.4c2s_c2h4p_mass + 0.3c3s_c3h6p_mass + 0.28dc3_c4sp_mass + 0.23quench_hpygas_mass + 0.23cgc_lpygas_mass +$
 $0.25ctdc1_offgas_mass - 0.1ethhdr_ethfd_mass - 0.15prohdr_profd_mass$

Make the ethane and propane feed flow rates degrees of freedom.

```

@specs begin
    -ethhdr_ethfd_mass
    -prohdr_profd_mass
end
dof = [m[:ethhdr_ethfd_mass], m[:prohdr_profd_mass]]

```

```

2-element Vector{VariableRef}:
 ethhdr_ethfd_mass
 prohdr_profd_mass

```

Add some variable bounds.

```

@bounds begin
    0.0 < ethhdr_ethfd_mass < 2.0e5
    0.0 < prohdr_profd_mass < 2.0e5
    flf5_n_furn < 5.0
    0.0 < flf5_pro_n_furn < 3.0
    1.0 < flf5_eth_n_furn
end
print_bounds(m)

```

Name	Fix	Value	Lower	Upper	Start
ethhdr_ethfd_mass			0	200000	80000
prohdr_profd_mass			0	200000	34000
flf5_eth_n_furn			1		4.182015
flf5_pro_n_furn			0	3	1.08349
flf5_n_furn				5	5.265505
5 variables					

Solve the optimization problem.

```
@solve
```

This is Ipopt version 3.14.19, running with linear solver MUMPS 5.8.1.

```

Number of nonzeros in equality constraint Jacobian...:    958
Number of nonzeros in inequality constraint Jacobian.:     0

```

```

Number of nonzeros in Lagrangian Hessian.....:      171

Total number of variables.....:      364
    variables with only lower bounds:          1
    variables with lower and upper bounds:      3
    variables with only upper bounds:          1
Total number of equality constraints.....:      362
Total number of inequality constraints.....:      0
    inequality constraints with only lower bounds: 0
    inequality constraints with lower and upper bounds: 0
    inequality constraints with only upper bounds: 0

iter   objective    inf_pr   inf_du lg(mu)  ||d|| lg(rg) alpha_du alpha_pr ls
  0   2.7446597e+04  3.16e-01  1.00e+00 -1.0  0.00e+00 -  0.00e+00  0.00e+00  0
  1   2.7633218e+04  3.02e-01  2.00e+01 -1.0  2.64e+04 -  8.41e-01  4.34e-02f  1
  2   2.7629509e+04  3.01e-01  5.10e+03 -1.0  6.27e+03 -  9.57e-01  4.05e-03h  1
  3   2.7628519e+04  3.00e-01  5.12e+03 -1.0  1.13e+05 -  2.67e-01  1.31e-03h 10
  4   2.7628413e+04  3.00e-01  5.13e+03 -1.0  1.57e+05 -  2.38e-01  2.35e-04h 12
  5   2.7628334e+04  3.00e-01  5.17e+03 -1.0  1.69e+05 -  5.51e-01  2.18e-04h 12
  6   2.7628287e+04  3.00e-01  5.17e+03 -1.0  1.87e+05 -  2.79e-01  1.97e-04h 12
  7   2.7628238e+04  3.00e-01  5.18e+03 -1.0  1.86e+05 -  5.91e-01  1.98e-04h 12
  8   2.7628184e+04  3.00e-01  5.19e+03 -1.0  1.82e+05 -  2.73e-01  2.02e-04h 12
  9   2.7628128e+04  3.00e-01  5.21e+03 -1.0  1.81e+05 -  1.00e+00  2.03e-04h 12
iter   objective    inf_pr   inf_du lg(mu)  ||d|| lg(rg) alpha_du alpha_pr ls
 10   2.7628061e+04  3.00e-01  5.21e+03 -1.0  1.75e+05 -  2.39e-01  2.10e-04h 12
 11   2.7627991e+04  3.00e-01  5.23e+03 -1.0  1.73e+05 -  1.00e+00  2.12e-04h 12
 12   2.7627904e+04  3.00e-01  5.23e+03 -1.0  1.65e+05 -  2.55e-01  2.23e-04h 12
 13   2.7440300e+04  6.53e+03  9.91e+03 -1.0  1.62e+05 -  1.00e+00  4.63e-01w  1
 14   2.7378508e+04  6.00e+03  9.12e+03 -1.0  1.18e+04 -  1.00e+00  8.08e-02w  1
 15   2.6615570e+04  7.65e+01  8.34e+02 -1.0  4.44e+03 -  9.94e-01  1.00e+00w  1
 16   2.6615578e+04  2.24e-04  9.86e-05 -1.0  1.18e+00 -  1.00e+00  1.00e+00h  1
 17   2.6615778e+04  8.08e-05  2.35e-05 -3.8  1.42e+01 -  1.00e+00  1.00e+00f  1
 18   2.6615778e+04  9.82e-11  4.46e-11 -5.7  1.54e-02 -  1.00e+00  1.00e+00h  1
 19   2.6615778e+04  5.82e-11  1.46e-11 -8.6  2.63e-04 -  1.00e+00  1.00e+00h  1

Number of Iterations.....: 19

                                (scaled)                                (unscaled)
Objective.....: -2.6615777991641837e+04    2.6615777991641837e+04
Dual infeasibility.....:  1.4551915228366852e-11    1.4551915228366852e-11
Constraint violation....:  5.8207660913467407e-11    5.8207660913467407e-11
Variable bound violation:  4.9999513862530875e-08    4.9999513862530875e-08
Complementarity.....:  2.5059491855751155e-09    2.5059491855751155e-09
Overall NLP error.....:  3.6883370376915734e-10    2.5059491855751155e-09

Number of objective function evaluations      = 148
Number of objective gradient evaluations      = 20
Number of equality constraint evaluations      = 148
Number of inequality constraint evaluations    = 0
Number of equality constraint Jacobian evaluations = 20
Number of inequality constraint Jacobian evaluations = 0
Number of Lagrangian Hessian evaluations     = 19
Total seconds in IPOPT                       = 0.007

EXIT: Optimal Solution Found.

```

Print the values of the degrees of freedom at the solution, the active bounds, and the value of the objective function.

```
print_vars(dof)
print_active(m)
eval_obj(m)
```

```
26615.777991641837
```

Update the start values.

```
set_start_values(m)
```

Lower the ethane feed price to half its previous value and solve the problem again.

```
set_objective_coefficient(m, m[:ethhdr_ethfd_mass], 5.0/100.0)
@solve
```

This is Ipopt version 3.14.19, running with linear solver MUMPS 5.8.1.

```
Number of nonzeros in equality constraint Jacobian...: 958
Number of nonzeros in inequality constraint Jacobian.: 0
Number of nonzeros in Lagrangian Hessian.....: 171
```

```
Total number of variables.....: 364
      variables with only lower bounds: 1
      variables with lower and upper bounds: 3
      variables with only upper bounds: 1
Total number of equality constraints.....: 362
Total number of inequality constraints.....: 0
      inequality constraints with only lower bounds: 0
      inequality constraints with lower and upper bounds: 0
      inequality constraints with only upper bounds: 0
```

iter	objective	inf_pr	inf_du	lg(mu)	d	lg(rg)	alpha_du	alpha_pr	ls
0	3.1743485e+04	1.20e+03	1.00e+00	-1.0	0.00e+00	-	0.00e+00	0.00e+00	0
1	3.2080768e+04	1.14e+03	3.24e+01	-1.0	3.00e+04	-	6.49e-01	5.18e-02f	1
2	3.2085297e+04	1.14e+03	4.93e+02	-1.0	1.25e+04	-	7.02e-02	2.03e-03f	1
3	3.2622401e+04	7.48e+01	3.61e+03	-1.0	1.27e+04	-	8.49e-01	1.00e+00f	1
4	3.8721705e+04	2.61e+03	1.09e+03	-1.0	9.13e+04	-	1.17e-01	9.21e-01f	1
5	4.0289328e+04	1.84e+03	3.22e+02	-1.0	2.72e+04	-	8.48e-01	7.96e-01f	1
6	4.0305004e+04	2.28e+02	3.86e+01	-1.0	2.44e+02	-	9.90e-01	8.85e-01f	1
7	4.0305062e+04	8.29e-03	2.03e-02	-1.0	8.05e-01	-	9.95e-01	1.00e+00h	1
8	4.0305262e+04	2.28e-05	8.71e-05	-3.8	1.40e+00	-	1.00e+00	1.00e+00f	1
9	4.0305262e+04	2.91e-11	1.50e-10	-5.7	1.34e-03	-	1.00e+00	1.00e+00h	1
iter	objective	inf_pr	inf_du	lg(mu)	d	lg(rg)	alpha_du	alpha_pr	ls
10	4.0305262e+04	2.91e-11	1.46e-11	-8.6	2.59e-05	-	1.00e+00	1.00e+00h	1

Number of Iterations.....: 10

	(scaled)	(unscaled)
Objective.....:	-4.0305262454300137e+04	4.0305262454300137e+04


```

Dual infeasibility.....: 1.4551915228366852e-11    1.4551915228366852e-11
Constraint violation....: 2.9103830456733704e-11    2.9103830456733704e-11
Variable bound violation: 4.9999688833679556e-08    4.9999688833679556e-08
Complementarity.....: 2.5059286537557141e-09    2.5059286537557141e-09
Overall NLP error.....: 2.4869221603700835e-10    2.5059286537557141e-09

```

```

Number of objective function evaluations      = 11
Number of objective gradient evaluations      = 11
Number of equality constraint evaluations      = 11
Number of inequality constraint evaluations    = 0
Number of equality constraint Jacobian evaluations = 11
Number of inequality constraint Jacobian evaluations = 0
Number of Lagrangian Hessian evaluations     = 10
Total seconds in IPOPT                       = 0.003

```

EXIT: Optimal Solution Found.

The solver decreases the fresh propane feed flow rate to zero.

```

print_vars(dof)
print_active(m)
eval_obj(m)

```

```
40305.26245430014
```

Part III

API

Chapter 2

MassBalanceOpt.jl

A package for formulating mass balance models of chemical processes and solving them with [JuMP](#).

2.1 Flowsheets, Streams, and Blocks

`MassBalanceOpt.AbstractBlock` – Type.

```
AbstractBlock
```

Abstract supertype for all blocks (Mixer, Splitter, etc.)

See also [Mixer](#), [Splitter](#), [Separator](#), [YieldReactor](#), [MultiYieldReactor](#), [StoicReactor](#), [@Block_fields](#), [@Block_init](#), [@Block_finish](#)

[source](#)

`MassBalanceOpt.Flowsheet` – Type.

```
Flowsheet(name::Symbol=:index, parent=nothing)
```

Create a flowsheet. If name is omitted it defaults to `:index`. If specified, parent must be another Flowsheet; The new flowsheet will be a child of the parent flowsheet. If parent is omitted, the flowsheet has no parent.

A Flowsheet serves as a container for streams, blocks (subtypes of `AbstractBlock`), and other flowsheets. The names of a flowsheet's ancestors are embedded in the variable and equation names of the blocks and streams contained in the flowsheet. For example, if a flowsheet named `:unit1` is a child of the index flowsheet (a flowsheet created with no parent), the variable and equation names in that flowsheet will start with `unit1_`. If the `:unit1` flowsheet has a child named `reactors` the names will start with `unit1_reactors_`. Variables and equations in a flowsheet with no parent have no prefix.

Examples

Create a `:index` flowsheet with no parent:

```
julia> fs = Flowsheet()  
Flowsheet(  
  name=index  )
```

Add a Stream to the `:index` flowsheet:

```
julia> in1 = Stream(:in1, fs, @components(h2,ch4))  
Stream(in1, fs=index, basis=FRAC, components=[h2, ch4])
```

```
julia> fs
Flowsheet(
  name=index,
  streams=[in1]  )
```

Create a flowsheet named `:unit1` whose parent is `:index`:

```
julia> fs_1 = Flowsheet(:unit1, fs)
Flowsheet(
  name=unit1,
  parent=index  )

julia> fs
Flowsheet(
  name=index,
  children=[unit1],
  streams=[in1]  )
```

See also [Stream](#), [AbstractBlock](#)

[source](#)

MassBalanceOpt.@components - Macro.

`@components` symbols...

Create and return a component set made up of one or more unquoted symbols.

Examples

```
julia> comps1 = @components h2 ch4 c2h4
OrderedSet{Symbol} with 3 elements:
 :h2
 :ch4
 :c2h4

julia> comps1 = @components(h2, ch4, c2h4)
OrderedSet{Symbol} with 3 elements:
 :h2
 :ch4
 :c2h4
```

See also [Stream](#)

[source](#)

MassBalanceOpt.Stream - Type.

Representation of a material stream in a mass balance model.

```
Stream(name::Symbol, fs, comps; basis=FRAC)
```

Create a new `Stream` with name and components `comps` in flowsheet `fs`. The keyword argument `basis` can be `FRAC` or `FLOW`, with `FRAC` being the default.

Creating a `Stream` does not create any variables or equations in the model. The stream becomes available for use as an argument to the constructors of blocks, which are subtyped of `AbstractBlock`. Blocks are

responsible for creating stream variables and equations. When a block creates stream variables it uses the basis of each stream to determine what variables to create. If `basis=FRAC` a mass fraction variable is created for every component in the stream; if `basis=FLOW` a component mass flow rate variable is created for every component.

Warning

None of the built-in blocks create equations to sum the mass fractions or the component mass flow rates.

Examples

Create two inlet streams and one outlet stream, then create a Mixer block that mixes the two inlet streams.

```
julia> m = Model(); fs = Flowsheet();

julia> in1 = Stream(:in1, fs, @components(h2, ch4))
Stream(in1, fs=index, basis=FRAC, comps=[h2, ch4])

julia> in2 = Stream(:in2, fs, @components(h2, ch4, c2h4), basis=FLOW)
Stream(in2, fs=index, basis=FLOW, comps=[h2, ch4, c2h4])

julia> out = Stream(:out, fs, @components(h2, ch4, c2h4))
Stream(out, fs=index, basis=FRAC, comps=[h2, ch4, c2h4])

julia> mix1 = Mixer(m, :mix1, fs, [in1, in2], out)
Mixer(in=[in1, in2], out=[out])

julia> print_vars(mix1)

```

Name	Value	Lower	Upper	Start	Spec
-----	-----	-----	-----	-----	-----
mix1_in1_mass	1				F
mix1_in1_h2_massfrac	0.5				F
mix1_in1_ch4_massfrac	0.5				F
mix1_in2_mass	1				F
mix1_in2_h2_mass	0.33333				F
mix1_in2_ch4_mass	0.33333				F
mix1_in2_c2h4_mass	0.33333				F
mix1_out_mass					
mix1_out_h2_massfrac					
mix1_out_ch4_massfrac					
mix1_out_c2h4_massfrac					

```
11 variables
```

See also [@stream](#), [@streams](#), [Flowsheet](#), [@components](#), [StreamBasis](#)

[source](#)

MassBalanceOpt.StreamBasis - Type.

```
@enum StreamBasis
```

FRAC: Generate mass fraction variables (default). FLOW: Generate component mass flow rate variables.

See also [Stream](#)

[source](#)

MassBalanceOpt.is_frac - Function.

```
is_frac(strm: Stream)
```

Return true if the strm basis=FRAC, otherwise return false.

See also [is_flow](#), [StreamBasis](#)

[source](#)

MassBalanceOpt.is_flow - Function.

```
is_flow(strm: Stream)
```

Return true if the strm basis=FLOW, otherwise return false.

See also [is_frac](#), [StreamBasis](#)

[source](#)

MassBalanceOpt.@stream - Macro.

```
@stream(name, comps, ex_basis=: (basis=FRAC))
```

Create a Stream named name with component set comps and StreamBasis basis. @stream assumes that the model is stored in a variable named m and the active flowsheet is stored in a variable named fs in the current scope.

Examples

```
julia> feed = @stream(feed, comps_feed)
julia> prod = @stream(feed, comps_prod, basis=FLOW)
```

See also [Stream](#), [@streams](#), [StreamBasis](#)

[source](#)

MassBalanceOpt.@streams - Macro.

```
@streams(blk)
```

Create a tuple of Streams using the expressions in the begin/end block blk. @streams assumes that the model is stored in a variable named m and the active flowsheet is stored in a variable named fs in the current scope.

Examples

```
julia> (ethfd, profd, cgcout) = @streams(begin
    ethfd, comps_ethfeed
    profd, comps_profeed
    cgcout, comps_cg, (basis=FLOW)
end)
```

See also [Stream](#), [@stream](#), [StreamBasis](#)

[source](#)

MassBalanceOpt.copy_stream - Function.

```
copy_stream(strm::Stream)
```

Copy the values of the strm variables in strm's from block into the values of the strm variables in strm's to block. If either the to or from field of strm is empty, do nothing.

See also [copy_streams](#)

[source](#)

MassBalanceOpt.copy_streams - Function.

```
copy_streams(strms::Vector{Stream})
```

Invoke copy_stream for each element of strms.

See also [copy_stream](#)

[source](#)

MassBalanceOpt.make_stream_vars! - Function.

```
make_stream_vars!(m::Model, strm::Stream, prefix::AbstractString, var_list::Vector{VariableRef})
-> Dict{Symbol, Any}
```

Create stream variables for strm using prefix at the beginning of the variable names. Add the new variables to var_list. Return the stream variables in a dictionary.

See also [Stream](#), [set_stream_var_specs!](#), [make_var!](#)

[source](#)

```
make_stream_vars!(m::Model, strms::Vector{Stream}, prefix::AbstractString,
↳ var_list::Vector{VariableRef})
-> Dict{Symbol, Dict}
```

Create stream variables for all the streams in the array strms and return them in a dictionary.

[source](#)

MassBalanceOpt.set_stream_var_specs! - Function.

```
set_stream_var_specs!(strms::Vector{Stream}, vars::Dict{Symbol, Dict})
```

Fix the variables stored in vars, for each stream in strms, equal to default values.

See also [Stream](#), [make_stream_vars!](#), [fix](#)

[source](#)

MassBalanceOpt.Mixer - Type.

```
Mixer <: AbstractBlock
```

Mix two or more inlet streams into one outlet stream.

Create a Mixer block:

```
Mixer(m::Model, name::Symbol, fs::Flowsheet, inlets::Vector{Stream}, outlet::Stream)
```

or if the current scope contains m and fs bound to a Model and Flowsheet:

```
@block(name, Mixer, inlets, outlet)
```

Examples

```
julia> m = Model(); fs = Flowsheet(); comps = @components A B;
```

```
julia> (in1, in2, out) = @streams begin
    in1, comps
    in2, comps
    out, comps
end;
```

```
julia> mix1 = @block(mix1, Mixer, [in1, in2], out);
```

```
julia> print_vars(mix1)
```

Name	Fix	Value	Lower	Upper	Start
----- ----- ----- ----- -----					
mix1_in1_mass	==	1			1
mix1_in1_A_massfrac	==	0.5			0.5
mix1_in1_B_massfrac	==	0.5			0.5
mix1_in2_mass	==	1			1
mix1_in2_A_massfrac	==	0.5			0.5
mix1_in2_B_massfrac	==	0.5			0.5
mix1_out_mass					
mix1_out_A_massfrac					
mix1_out_B_massfrac					
9 variables					

See also [Stream](#), [Splitter](#), [Separator](#), [YieldReactor](#), [MultiYieldReactor](#), [StoicReactor](#)

[source](#)

MassBalanceOpt.Splitter – Type.

```
Splitter <: AbstractBlock
```

Split one inlet stream into two or more outlet streams.

Create a Splitter block:

```
Splitter(m::Model, name::Symbol, fs::Flowsheet, inlet::Stream, outlets::Vector{Stream})
```

or if the current scope contains m and fs bound to a Model and Flowsheet:

```
@block(name, Splitter, inlet, outlets)
```

Examples

```
julia> m = Model(); fs = Flowsheet(); comps = @components A B;
```

```
julia> (in1, out1, out2) = @streams begin
    in1, comps
    out1, comps
    out2, comps
end;
```



```
julia> spl = @block(spl, Splitter, in1, [out1, out2]);

julia> print_vars(spl)
      Name          Fix      Value      Lower      Upper      Start
-----|-----|-----|-----|-----|
spl_in1_mass        ==          1          |          |          1
spl_in1_A_massfrac  ==         0.5          |          |         0.5
spl_in1_B_massfrac  ==         0.5          |          |         0.5
spl_out1_mass        |          |          |          |          |
spl_out1_A_massfrac  |          |          |          |          |
spl_out1_B_massfrac  |          |          |          |          |
spl_out2_mass        |          |          |          |          |
spl_out2_A_massfrac  |          |          |          |          |
spl_out2_B_massfrac  |          |          |          |          |
spl_out1_split_frac  ==         0.5          |          |         0.5
spl_out2_split_frac  |          |          |          |         0.5
11 variables
```

See also [Stream](#), [Mixer](#), [Separator](#), [YieldReactor](#), [MultiYieldReactor](#), [StoicReactor](#)

[source](#)

MassBalanceOpt.Separator - Type.

Separator <: **AbstractBlock**

Separate the components in the inlet stream into two or more outlet streams.

Create a Separator block:

```
Separator(m::Model, name::Symbol, fs::Flowsheet, inlet::Stream, outlets::Vector{Stream})
```

or if the current scope contains m and fs bound to a Model and Flowsheet:

```
@block(name, Separator, inlet, outlets)
```

Examples

```
julia> m = Model(); fs = Flowsheet(); comps = @components A B;
```

```
julia> (in1, out1, out2) = @streams begin
    in1, comps
    out1, comps
    out2, comps
end;
```

```
julia> sep = @block(sep, Separator, in1, [out1, out2]);
```

```
julia> @set sep_A_out1_split = 0.3
```

```
julia> @set sep_B_out1_split = 0.6
```

```
julia> set_start_values(sep)
```

```
julia> print_vars(sep)
      Name          Fix      Value      Lower      Upper      Start
-----|-----|-----|-----|-----|
```

```

sep_in1_mass      ==      1|      |      |      1|
sep_in1_A_massfrac ==      0.5|      |      |      0.5|
sep_in1_B_massfrac ==      0.5|      |      |      0.5|
sep_out1_mass      |      |      |      0.45|
sep_out1_A_massfrac |      |      |      0.3333333|
sep_out1_B_massfrac |      |      |      0.6666667|
sep_out2_mass      |      |      |      0.55|
sep_out2_A_massfrac |      |      |      0.6363636|
sep_out2_B_massfrac |      |      |      0.3636364|
sep_in1_A_mass      |      |      |      0.5|
sep_in1_B_mass      |      |      |      0.5|
sep_out1_A_mass      |      |      |      0.15|
sep_out1_B_mass      |      |      |      0.3|
sep_out2_A_mass      |      |      |      0.35|
sep_out2_B_mass      |      |      |      0.2|
sep_A_out1_split     ==      0.3|      |      |      0.3|
sep_A_out2_split     |      |      |      0.7|
sep_B_out1_split     ==      0.6|      |      |      0.6|
sep_B_out2_split     |      |      |      0.4|
19 variables

```

See also [Stream](#), [Mixer](#), [Splitter](#), [YieldReactor](#), [MultiYieldReactor](#), [StoicReactor](#)

[source](#)

MassBalanceOpt.@stoic – Macro.

`@stoic(expr)`

Create an array of `OrderedDicts`, each of which encodes the stoichiometric coefficients of a single reaction. The `expr` should be one or more balanced chemical reactions written in the form:

```
aA + bB + ... => cC + dD
```

where `a`, `b`, etc. are real or integer stoichiometric coefficients, and `A`, `B`, etc. are species. The returned array looks like:

```

coef[1] = OrderedDict{:A => -a, :B => -b, :C => c, :D => d} # coefficients for reaction 1
coef[2] = OrderedDict{:A => -a, :B => -b, :C => c, :D => d} # coefficients for reaction 2
...

```

Note that the coefficients in the `OrderedDict` are positive for products and negative for reactants.

Examples

```

julia> coef = @stoic A + B => C
julia> coef
1-element Vector{OrderedCollections.OrderedDict{Symbol, Real}}:
OrderedCollections.OrderedDict{:A => -1, :B => -1, :C => 1}

julia> @stoic begin
    A + B => C
    2A + 3B => 4C + D
end
2-element Vector{OrderedCollections.OrderedDict{Symbol, Real}}:
OrderedCollections.OrderedDict{:A => -1, :B => -1, :C => 1}
OrderedCollections.OrderedDict{:A => -2, :B => -3, :C => 4, :D => 1}

```

See also [StoicReactor](#)

[source](#)

MassBalanceOpt.StoicReactor - Type.

```
StoicReactor <: AbstractBlock
```

Model a reactor in which a set of stoichiometric reactions with specified conversions takes place.

Create a StoicReactor block:

```
StoicReactor(m::Model, name::Symbol, fs::Flowsheet, inlet::Stream, outlet::Stream,
  stoic_coef::Vector{OrderedDict{Symbol, Real}},
  mw::Dict{Symbol, Float64},
  conv::OrderedDict{Int, NamedTuple{(:c, :X), Tuple{Symbol, Float64}}} = OrderedDict{Int,
  ↪ NamedTuple{(:c, :X),
  Tuple{Symbol, Float64}}}())
```

or if the current scope contains m and fs bound to a Model and Flowsheet:

```
@block(name, StoicReactor, inlet, outlet, stoic_coef, mw, conv)
```

Examples

```
julia> m = Model(); fs = Flowsheet(); comps1 = @components A B; comps2 = @components A B C D;

julia> (in1, out1) = @streams begin
    in1, comps1
    out1, comps2
end;

julia> mw = Dict{:A => 30.0, :B => 28.0, :C => 35.0, :D => 30.0};

julia> coef = @stoic A + B => C + D
1-element Vector{OrderedDict{Symbol, Real}}:
OrderedDict{:A => -1, :B => -1, :C => 1, :D => 1}

julia> conv = OrderedDict{1 => (c=:A, X=0.65)}
OrderedDict{Int64, NamedTuple{(:c, :X), Tuple{Symbol, Float64}}} with 1 entry:
 1 => (c = :A, X = 0.65)

julia> r1 = @block(r1, StoicReactor, in1, out1, coef, mw, conv);

julia> print_vars(r1)
```

Name	Fix	Value	Lower	Upper	Start
r1_in1_mass	==	1			1
r1_in1_A_massfrac	==	0.5			0.5
r1_in1_B_massfrac	==	0.5			0.5
r1_out1_mass					
r1_out1_A_massfrac					
r1_out1_B_massfrac					
r1_out1_C_massfrac					
r1_out1_D_massfrac					
r1_in1_A_mass					
r1_in1_B_mass					

r1_out1_A_mass				
r1_out1_B_mass				
r1_out1_C_mass				
r1_out1_D_mass				
r1_in1_A_moles				
r1_in1_B_moles				
r1_out1_A_moles				
r1_out1_B_moles				
r1_out1_C_moles				
r1_out1_D_moles				
r1_extent_rx_1				
r1_conv_A_rx_1	==	0.65		0.65
22 variables				

See also [@stoic](#), [Stream](#), [YieldReactor](#), [MultiYieldReactor](#)

[source](#)

MassBalanceOpt.YieldReactor - Type.

YieldReactor <: **AbstractBlock**

Model a reactor with one inlet and one outlet stream, in which each component in the inlet stream has a specified yield to the components in the outlet stream.

Create a YieldReactor block:

```
YieldReactor(m::Model, name::Symbol, fs::Flowsheet, inlet::Stream, outlet::Stream)
```

julia or if the current scope contains m and fs bound to a Model and Flowsheet:

```
@block(name, YieldReactor, inlet, outlet)
```

Examples

```
julia> m = Model(); fs = Flowsheet(); comps1 = @components A B; comps2 = @components A B C D;
```

```
julia> (in1, out1) = @streams begin
```

```
    in1, comps1
    out1, comps2
```

```
end;
```

```
julia> r1 = @block(r1, YieldReactor, in1, out1);
```

```
julia> @set begin
```

```
    r1_y_C_from_A = 0.3
    r1_y_D_from_A = 0.4
    r1_y_C_from_B = 0.6
    r1_y_D_from_B = 0.1
```

```
end
```

```
julia> set_start_values(r1)
```

```
julia> print_vars(r1)
```

Name	Fix	Value	Lower	Upper	Start
-----	----	-----	-----	-----	-----

r1_in1_mass	==	1				1
r1_in1_A_massfrac	==	0.5				0.5
r1_in1_B_massfrac	==	0.5				0.5
r1_out1_mass						1
r1_out1_A_massfrac						0.15
r1_out1_B_massfrac						0.15
r1_out1_C_massfrac						0.45
r1_out1_D_massfrac						0.25
r1_in1_A_mass						0.5
r1_in1_B_mass						0.5
r1_out1_A_mass						0.15
r1_out1_B_mass						0.15
r1_out1_C_mass						0.45
r1_out1_D_mass						0.25
r1_y_A_from_A						0.3
r1_y_B_from_A	==	0				0
r1_y_C_from_A	==	0.3				0.3
r1_y_D_from_A	==	0.4				0.4
r1_y_A_from_B	==	0				0
r1_y_B_from_B						0.3
r1_y_C_from_B	==	0.6				0.6
r1_y_D_from_B	==	0.1				0.1
22 variables						

See also [Stream](#), [MultiYieldReactor](#), [StoicReactor](#)

[source](#)

MassBalanceOpt.MultiYieldReactor – Type.

MultiYieldReactor <: **AbstractBlock**

Model a reactor with multiple inlet and outlet streams. This is used to represent a set of YieldReactors operating in parallel. The number of inlet streams must equal the number of outlet streams, and the length of feed_names must equal the number of streams.

Create a MultiYieldReactor block:

```
MultiYieldReactor(m::Model, name::Symbol, fs::Flowsheet, inlets::Vector{Stream},
↳ outlets::Vector{Stream},
    feed_names::Vector{Symbol}, reactor_name::Symbol)
```

or if the current scope contains m and fs bound to a Model and Flowsheet:

```
@block(name, MultiYieldReactor, inlets, outlets, feed_names, reactor_name)
```

Examples

```
julia> m = Model(); fs = Flowsheet();

julia> compsA = @components(A); compsB = @components(B); comps_out = @components A B C D;

julia> (feedA, feedB, outA, outB) = @streams begin
    feedA, compsA
    feedB, compsB
    outA, comps_out
```

```

    outB , comps_out
end;

julia> r1 = @block(r1, MultiYieldReactor, [feedA, feedB], [outA, outB], [:feedA, :feedB], :rx);

julia> print_vars(r1)

```

Name	Fix	Value	Lower	Upper	Start
r1_feedA_mass	==	1			1
r1_feedA_A_massfrac	==	1			1
r1_feedB_mass	==	1			1
r1_feedB_B_massfrac	==	1			1
r1_outA_mass					
r1_outA_A_massfrac					
r1_outA_B_massfrac					
r1_outA_C_massfrac					
r1_outA_D_massfrac					
r1_outB_mass					
r1_outB_A_massfrac					
r1_outB_B_massfrac					
r1_outB_C_massfrac					
r1_outB_D_massfrac					
r1_feedA_A_mass					
r1_feedB_B_mass					
r1_outA_A_mass					
r1_outA_B_mass					
r1_outA_C_mass					
r1_outA_D_mass					
r1_outB_A_mass					
r1_outB_B_mass					
r1_outB_C_mass					
r1_outB_D_mass					
r1_total_feed_mass					
r1_feedA_n_rx					
r1_feedA_rate	==	1			1
r1_feedB_n_rx					
r1_feedB_rate	==	1			1
r1_n_rx					
r1_feedA_y_A_from_A					0
r1_feedA_y_B_from_A	==	0			0
r1_feedA_y_C_from_A	==	0			0
r1_feedA_y_D_from_A	==	0			0
r1_feedB_y_A_from_B	==	0			0
r1_feedB_y_B_from_B					0
r1_feedB_y_C_from_B	==	0			0
r1_feedB_y_D_from_B	==	0			0

38 variables

See also [Stream](#), [YieldReactor](#), [StoicReactor](#)

[source](#)

MassBalanceOpt.@block - Macro.

```
@block(name, kind, args...)
```

Create a block named `name` with type `kind` in flowsheet `fs` using arguments `args...`. The value of `kind` must be a subtype of `AbstractBlock`. `@block` assumes that the model is stored in a variable named `m` and the active flowsheet is stored in a variable named `fs` in the current scope.

Examples

```
julia> ctdcl = @block(ctdcl, Separator, arxo, [offgas, dclbt])
```

```
julia> arx = @block(arx, StoicReactor, dc2oh, arxo, arx_stoic, arx_mw, arx_conv)
```

See also [Mixer](#), [Splitter](#), [Separator](#), [YieldReactor](#), [MultiYieldReactor](#), [StoicReactor](#)

[source](#)

MassBalanceOpt.@Block_fields – Macro.

```
@Block_fields
```

Create the fields typically used in a block. They are:

```
name::Symbol
fs::Flowsheet
inlets::Vector{Stream}
outlets::Vector{Stream}
strm_vars::Dict{Symbol, Dict}
var_list::Vector{VariableRef}
eq_list::Vector{ConstraintRef}
```

See also [AbstractBlock](#), [@Block_init](#), [@Block_finish](#)

[source](#)

MassBalanceOpt.@Block_init – Macro.

```
@Block_init
```

Create the stream variables used in a block and set their specs.

See also [AbstractBlock](#), [@Block_fields](#), [@Block_finish](#)

[source](#)

MassBalanceOpt.@Block_finish – Macro.

```
@Block_finish
```

Assign a newly created block referenced by `self` to its container flowsheet, and record the block in the to and from fields of the inlet and outlet streams.

See also [AbstractBlock](#), [@Block_fields](#), [@Block_init](#)

[source](#)

MassBalanceOpt.make_mass_flow_vars! – Function.

```
make_mass_flow_vars!(m::Model,
    prefix::AbstractString,
    inlets::Vector{Stream},
    outlets::Vector{Stream},
    strm_vars::Dict{Symbol, Dict},
    var_list::Vector{VariableRef},
    eq_list::Vector{ConstraintRef})::Dict{Symbol, Dict}
```

Conditionally create component mass flow rate variables for the streams in inlets and outlets. If the stream has a FLOW basis, component mass flow rate variables already exist. If the stream has a FRAC basis, make new component mass flow variables and equations to calculate them. Add equations to calculate the outlet stream total mass flow rates from the component mass flow rates.

[source](#)

`JuMP.set_start_values` – Method.

```
set_start_values(blk::AbstractBlock; copy_inlets::Bool=true)
```

Set the start values of the block variables in `blk`. If `copy_inlets=false`, don't copy the inlet stream variable start values from the upstream block.

[source](#)

2.2 Variables and Equations

`MassBalanceOpt.make_var!` – Function.

```
make_var!(m::Model, name::AbstractString, var_list::Vector{VariableRef})
```

Create a new JuMP variable named `name`, register it in model `m` as `m[:name]`, and add it to `var_list`.

[source](#)

`JuMP.fix` – Method.

```
fix(var::VariableRef, [val::Real])
```

If `val` is passed, fix the variable `var` equal to `val` and set its start value to `val`. If no `val` is passed, fix the variable equal to its value in the most recent solution. If no solution is available, fix the variable equal to its start value. If no start value is available, fix the variable equal to 0.

See also [free](#), [@specs](#)

[source](#)

`MassBalanceOpt.free` – Function.

```
free(var::VariableRef)
```

Delete the fixing constraint for variable `var`.

See also [fix](#), [@specs](#)

[source](#)

`MassBalanceOpt.flip` – Function.

```
flip(var1::VariableRef, [var2::VariableRef])
```

If `var2` is passed, invert the specs of `var1` and `var2`, i.e., if `var1` is fixed and `var2` is free, make `var1` free and `var2` fixed; if `var1` is free and `var2` is fixed, make `var1` fixed and `var2` free. If both vars have the same spec, do nothing.

If `var2` is not passed, invert the spec of `var1`.

Warning

The newly fixed variable will have its value set to:

1. its value in the most recent solution
2. its start value
3. 0

in that order. Use `set_value` or `@set` to change the value of the newly fixed variable.

See also [@set](#), [@specs](#)

[source](#)

`MassBalanceOpt.connect` – Function.

```
connect(var_lhs::VariableRef, var_rhs::VariableRef)
```

Given two variables `var_lhs` and `var_rhs`, at least one of which is fixed, add a new constraint:

```
var_lhs == var_rhs
```

If both variables are fixed, free `var_rhs`. Otherwise free the single fixed variable. If both variables are free, don't add a new constraint and return nothing.

Return the newly created `ConstraintRef`. See also [@set](#)

[source](#)

```
connect(strm::Stream)
```

Connect all the variables in the source and destination blocks of `strm`. If the stream has no source, or no destination, do nothing.

[source](#)

```
connect(m::Model, strms::Vector{Stream})
```

Connect all the streams in the array `strms`.

[source](#)

```
connect(fs::Flowsheet)
```

Connect all the streams in the flowsheet `fs`.

[source](#)

`JuMP.set_value` – Method.

```
set_value(var::VariableRef, val::Real)
```

If `var` is fixed, reset its fixed value to `val`. If `var` is free, set its start value to `val`.

See also [fix](#), [@set](#), [@values](#)

[source](#)

MassBalanceOpt.set_lower – Function.

```
set_lower(var::VariableRef, lower::Real)
```

If var is not fixed, set its lower bound to lower.

See also [set_upper](#), [delete_lower](#)

[source](#)

MassBalanceOpt.set_upper – Function.

```
set_upper(var::VariableRef, upper::Real)
```

If var is not fixed, set its upper bound to upper.

See also [set_lower](#), [delete_upper](#)

[source](#)

MassBalanceOpt.delete_lower – Function.

```
delete_lower(var::VariableRef)
```

If var is not fixed and has a lower bound, delete its lower bound.

See also [set_lower](#), [delete_upper](#)

[source](#)

MassBalanceOpt.delete_upper – Function.

```
delete_upper(var::VariableRef)
```

If var is not fixed and has an upper bound, delete its upper bound.

See also [set_upper](#), [delete_lower](#)

[source](#)

MassBalanceOpt.@set – Macro.

```
@set expr
```

Do the operations described by expr (see examples below). @set assumes that the model is stored in a variable named `m` in the current scope.

Examples

Set the value of the variable `var` to 100.0. If the variable is fixed, set the fixed value to 100.0, otherwise set the start value to 100.0:

```
julia> @set var = 100.0
```

Connect variables `var1` and `var2`. This adds the equation `var1 == var2` to the model. At least one of the variables must be fixed:

```
julia> @set var1 = var2
```

Set the upper bound on `var` to 100.0:

```
julia> @set var <= 100.0
```

Set the upper bound on var to 100.0:

```
julia> @set var <= 100.0
```

Delete the upper bound on var:

```
julia> @set var <= Inf
```

Delete the lower bound on var:

```
julia> @set -Inf <= var
```

Delete the lower and upper bounds on var:

```
julia> @set -Inf <= var <= Inf
```

Flip the specs on var1 and var2. If var1 is fixed and var2 is free, free var1 and fix var2. If var1 is free and var2 is fixed, fix var1 and free var2. If both variables have the same spec, do nothing:

```
julia> @set var1 ~ var2
```

Flip the spec on var. If var is fixed, free var. If var is free, fix var:

```
julia> @set ~var
```

Fix var:

```
julia> @set +var
```

Free var:

```
julia> @set -var
```

Set the lower and upper bounds on var:

```
julia> @set 1.0 < var < 2.0
julia> @set 2.0 > var > 1.0
```

Combine several operations in a begin/end block:

```
julia> @set begin
    x > 1.0
    y < 2.0
    1.0 < z < 2.0
    p ~ q
end
```

See also [@values](#), [@specs](#), [@bounds](#)

[source](#)

MassBalanceOpt.@values – Macro.

```
@values expr
```

Alias for @set.

See also [@set](#), [@specs](#), [@bounds](#)

[source](#)

MassBalanceOpt.@specs – Macro.

```
@specs expr
```

Alias for @set.

See also [@set](#), [@values](#), [@bounds](#)

[source](#)

MassBalanceOpt.@bounds – Macro.

```
@bounds expr
```

Alias for @set.

See also [@set](#), [@values](#), [@specs](#)

[source](#)

MassBalanceOpt.make_eq! – Function.

```
make_eq!(m::Model, name::AbstractString, con::ConstraintRef, eq_list::Vector{ConstraintRef})
```

Register the equation con in model m using name as the base name. Add the equation to eq_list.

See also [make_var!](#)

[source](#)

2.3 Printing and Output

MassBalanceOpt.print_vars – Function.

```
print_vars([io::IO], vars::Vector{VariableRef})
```

Print the variables in vars in a table.

See also [print_fixed](#), [print_free](#), [print_bounds](#), [print_active](#), [print_eqs](#), [write_vars](#)

[source](#)

```
print_vars([io::IO], var::VariableRef)
```

Print a single variable var.

[source](#)

```
print_vars([io::IO], m::Model)
```

Print all the variables in model m.

[source](#)

```
print_vars([io::IO], blk::AbstractBlock)
```

Print all the variables in block blk.

[source](#)

```
print_vars([io::IO], m::Model, glob::AbstractString)
```

Print the variables in model m that match glob, where glob is a string containing one or more * wildcard characters.

Examples

```
julia> print_vars(m, "*mass")
```

```
julia> print_vars(m, "*cgcfeed*")
```

[source](#)

```
print_vars([io::IO], m::Model, pattern::Regex)
```

Print the variables in model m that match the regular expression pattern.

Examples

```
julia> print_vars(m, r"^.*mass$")
```

[source](#)

MassBalanceOpt.write_vars - Function.

```
write_vars([io::IO], m::Model)
```

Write the values of all the variables in m to io in the format:

```
@values begin
    var_1 = 1.0
    var_2 = 2.0
    ...
end
```

[source](#)

MassBalanceOpt.print_fixed - Function.

```
print_fixed([io::IO], m::Model)
```

Print all the fixed variables in model m.

See also [print_free](#), [print_bounds](#), [print_active](#)

[source](#)

```
print_fixed([io::IO], blk::AbstractBlock)
```

Print all the fixed variables in block blk.

[source](#)

MassBalanceOpt.print_free – Function.

```
print_free([io::IO], m::Model)
```

Print all the free variables in model m.

See also [print_fixed](#), [print_bounds](#), [print_active](#)

[source](#)

```
print_free([op::IO], blk::AbstractBlock)
```

Print all the free variables in block blk.

[source](#)

MassBalanceOpt.print_bounds – Function.

```
print_bounds([io::IO], m::Model)
```

Print the variables in m with lower or upper bounds.

See also [print_fixed](#), [print_free](#), [print_active](#)

[source](#)

```
print_bounds([io::IO], blk::AbstractBlock)
```

Print the variables in blk with lower or upper bounds.

[source](#)

MassBalanceOpt.print_active – Function.

```
print_active([io::IO], m::Model)
```

Print the variables in m whose values equal their lower or upper bounds.

See also [print_fixed](#), [print_free](#), [print_bounds](#)

[source](#)

MassBalanceOpt.print_eqs – Function.

```
print_eqs([io::IO], m::Model)
```

Print all the equations in model m.

See also [print_vars](#)

[source](#)

```
print_eqs([io::IO], blk::AbstractBlock)
```

Print all the equations in block blk.

[source](#)

```
print_eqs([io::IO], m::Model, eq::Symbol)
```

Print the equation registered in model `m` as `m[:eq]`

[source](#)

`MassBalanceOpt.print_model` – Function.

```
print_model([io::IO], model_or_block::Union{Model, AbstractBlock})
```

Print all the variables and equations in a model or block specified by `model_or_block`.

See also [print_vars](#), [print_eqs](#)

[source](#)

2.4 Solving Models

`MassBalanceOpt.@solve` – Macro.

```
@solve
```

Run the function `optimize!(m)` where `m` is a JuMP model in the current scope.

[source](#)

`MassBalanceOpt.eval_obj` – Function.

```
eval_obj(m::Model)
```

Evaluate the objective function in `m`, using the solution values if a solution exists, otherwise using the start values.

[source](#)

2.5 Index

- [MassBalanceOpt.AbstractBlock](#)
- [MassBalanceOpt.Flowsheet](#)
- [MassBalanceOpt.Mixer](#)
- [MassBalanceOpt.MultiYieldReactor](#)
- [MassBalanceOpt.Separator](#)
- [MassBalanceOpt.Splitter](#)
- [MassBalanceOpt.StoicReactor](#)
- [MassBalanceOpt.Stream](#)
- [MassBalanceOpt.StreamBasis](#)
- [MassBalanceOpt.YieldReactor](#)
- [JuMP.fix](#)
- [JuMP.set_start_values](#)
- [JuMP.set_value](#)

- `MassBalanceOpt.connect`
- `MassBalanceOpt.copy_stream`
- `MassBalanceOpt.copy_streams`
- `MassBalanceOpt.delete_lower`
- `MassBalanceOpt.delete_upper`
- `MassBalanceOpt.eval_obj`
- `MassBalanceOpt.flip`
- `MassBalanceOpt.free`
- `MassBalanceOpt.is_flow`
- `MassBalanceOpt.is_frac`
- `MassBalanceOpt.make_eq!`
- `MassBalanceOpt.make_mass_flow_vars!`
- `MassBalanceOpt.make_stream_vars!`
- `MassBalanceOpt.make_var!`
- `MassBalanceOpt.print_active`
- `MassBalanceOpt.print_bounds`
- `MassBalanceOpt.print_eqs`
- `MassBalanceOpt.print_fixed`
- `MassBalanceOpt.print_free`
- `MassBalanceOpt.print_model`
- `MassBalanceOpt.print_vars`
- `MassBalanceOpt.set_lower`
- `MassBalanceOpt.set_stream_var_specs!`
- `MassBalanceOpt.set_upper`
- `MassBalanceOpt.write_vars`
- `MassBalanceOpt.@Block_fields`
- `MassBalanceOpt.@Block_finish`
- `MassBalanceOpt.@Block_init`
- `MassBalanceOpt.@block`
- `MassBalanceOpt.@bounds`
- `MassBalanceOpt.@components`
- `MassBalanceOpt.@set`

- `MassBalanceOpt.solve`
- `MassBalanceOpt.specs`
- `MassBalanceOpt.stoic`
- `MassBalanceOpt.stream`
- `MassBalanceOpt.streams`
- `MassBalanceOpt.values`