luacoolprop

LuaTeX access to CoolProp for PGFPlots thermodynamic diagrams

Christophe Jorssen
christophe.jorssen@gmail.com

Version 1.2.0 — 2026-11-20

Contents

I Getting started **7**

1 Choose your starting point **7**
1.1 A first thermodynamic picture . . . 7
1.2 Choose coordinates before choosing curves . . . 8

2 Customize a figure without changing its model **9**
2.1 Select a small, legible network . . . 9
2.2 Transparent worksheet and solution styles . . . 9
2.3 Monochrome is a drawing style, not a sampling mode . . . 10

3 Own the axis, the process or the data **11**
3.1 An axis with individually selected families and a process . . . 11
3.2 A direct Lua calculation, without drawing . . . 12

4 Find the right family of keys **12**
4.1 Recipes by task . . . 13

II Tutorials **14**

5 A teaching pair for a real-fluid refrigeration machine **14**
5.1 The exercise behind the diagram . . . 14
5.2 One shared source and two thin drivers . . . 14
5.3 Step 1: choose the worksheet's reading domain . . . 15
5.4 Step 2: select purposeful isoline families . . . 16
5.5 Result: the student worksheet diagram . . . 17
5.6 Step 3: establish the five thermodynamic states . . . 17
5.7 Step 4: draw each process from its invariant . . . 18
5.8 Step 5: add state names and reading aids . . . 19
5.9 Result: the model-solution diagram . . . 21
5.10 Step 6: turn readings into an energy balance . . . 21
5.11 Embedding the two PDFs in course material . . . 22
5.12 Adapting the tutorial safely . . . 22
5.13 Instructor's final checklist . . . 23

6 Advanced tutorial: reconstructing an R134a air conditioner **24**
6.1 Translate the statement into thermodynamic constraints . . . 24
6.2 Use a stable plotting domain . . . 24
6.3 Stage 1: identify the three phase regions . . . 25
6.4 Stage 2: recognize the ideal-gas region . . . 26
6.5 Stage 3: locate state 1 from two measured properties . . . 27
6.6 Stage 4: construct the compressor outlet . . . 28
6.7 Stage 5: cool at constant pressure to state 3 . . . 29
6.8 Stage 6: throttle, close the cycle, and locate state 4 . . . 30
6.9 Obtain every numerical value from process exports . . . 31
6.10 Files, rebuild, and adaptation checklist . . . 32

7 PV tutorial: butane in a domestic gas lighter **34**
7.1 Translate the exercise into thermodynamic data . . . 34
7.2 Keep one executable source for all five stages . . . 34
7.3 Step 1: build a readable log–log worksheet . . . 35
7.4 Step 2: read phase regions and the critical isotherm . . . 37
7.5 Step 3: recognize the ideal-gas region geometrically . . . 38
7.6 Step 4: locate the initial state with TikZ intersections . . . 39
7.7 Step 5: draw the rigid-tank heating process . . . 41
7.8 Rebuild and adapt the tutorial . . . 42

8 Teaching pair: an R1234yf swimming-pool heat pump **44**
8.1 Translate the statement into four idealized devices . . . 44
8.2 Establish the steady-flow first law . . . 44
8.3 Use one source and two thin drivers . . . 45
8.4 Step 1: reproduce the blank PH chart . . . 45
8.5 Step 2: derive both operating pressures . . . 47
8.6 Step 3: compute the cycle in dependency order . . . 48
8.7 Step 4: place states by intersections, not copied coordinates . . . 49
8.8 Step 5: calculate and interpret the condenser power . . . 51
8.9 Rebuild and adapt the teaching pair . . . 51

9 Expert tutorial: methane liquefaction by the Linde process **52**
9.1 Read the process as a directed network . . . 52
9.2 Use a shared worksheet and progressive solution . . . 52
9.3 Step 1: regenerate the blank methane PH chart . . . 53
9.4 Step 2: calculate the primitive states . . . 55
9.5 Step 3: throttle and separate the two-phase outlet . . . 56
9.6 Step 4: close the separator mass balance . . . 58
9.7 Step 5: close the two-stream regenerator balance . . . 59
9.8 Step 6: close the mixer balance and reveal state 1 . . . 59
9.9 Step 7: construct the first compression and cooler . . . 59
9.10 Step 8: repeat the invariant pattern for stages 2 and 3 . . . 60
9.11 Step 9: calculate powers with a declared sign convention . . . 62
9.12 Step 10: read the complete process without inventing a cycle . . . 63
9.13 Graphical estimates versus live calculations . . . 64
9.14 Numerical API and reproducible build . . . 64

III User manual **65**

10 Purpose and scope **65**
10.1 How to cite CoolProp . . . 66

11 Installation and runtime requirements **66**
11.1 Install the TeX files . . . 66
11.2 Security boundary and `-shell-escape` . . . 67
11.3 Choose a compatible CoolProp binary . . . 67
11.4 macOS: download or compile the `.dylib` . . . 68
11.5 Linux: download or compile the `.so` . . . 69
11.6 Windows: download or compile `CoolProp.dll` . . . 69
11.7 Verify the runtime installation . . . 70
11.8 LaTeX, plain TeX, and ConTeXt . . . 71
11.9 LaTeX number and unit formatting with `siunitx` . . . 72
11.10 Repository build workflow . . . 73

12 Basic users: quick start **73**
12.1 A fully automatic diagram . . . 73
12.2 Another fluid . . . 74

13 Diagram identifiers and stable API **75**

14 Advanced users: manual PGFPlots axes **78**
14.1 Curve-family macros . . . 79
14.2 Value grids . . . 79

15 Automatic labels with pgfplots-autonode **80**
15.1 LuaCoolProp keys forwarded to autonode . . . 82

16 Curve names and manual overrides **83**

17 Thermodynamic process paths **83**

IV Complete TeX reference **84**

18 How to read this reference **84**
18.1 Scope, units, and validation contracts . . . 85

19 Loading in the three supported formats **86**

20 Global configuration and fluid constants **87**
20.1 Global keys: `/luacoolprop` . . . 89

21 Reusing numerical curves and processes **91**

22 Externalizing complete pictures with TikZ **93**

23 Exporting numerical tables for PGFPlots **94**
23.1 Columns, units and disconnected domains . . . 95
23.2 Provenance and file ownership . . . 96

24 The generic diagram API **96**
24.1 A symmetric capability contract . . . 98
24.2 Pure-fluid and reference-state contracts . . . 98
24.3 Disconnected thermodynamic domains . . . 99

25 PH convenience commands **100**
25.1 Alternate command names . . . 102

26 Shared diagram controls and PH axis units **103**
26.1 Fluid, families, ranges, and axis . . . 103
26.2 Quality grid . . . 106
26.3 Temperature, entropy, and specific-volume grids . . . 107
26.3.1 Temperature and isotherms . . . 107
26.3.2 Specific entropy and isentropes . . . 108
26.3.3 Specific volume and isochores . . . 109
26.4 Adaptive sampling and numerical output . . . 111
26.5 Curve styles, colours, and legend participation . . . 113
26.6 Labels and family defaults . . . 115
26.7 Per-curve overrides . . . 120
26.8 Autonode options forwarded by PH diagrams . . . 121
26.9 Alternate label-control keys . . . 122

27 Shared process paths and the PH projection **124**
27.1 Process reference, diagnostics, and axis units . . . 124
27.2 Thermodynamic definition . . . 126
27.3 Identity, appearance, and labels . . . 128
27.4 Named intersections and numeric endpoint access . . . 130
27.5 Process scaling and sampling . . . 131

28 PV pressure–specific-volume diagrams **132**
28.1 PV convenience commands . . . 132
28.2 PV keys and defaults . . . 136
28.3 PV curve names, intersections, and overrides . . . 137
28.4 PV process keys and numerical exports . . . 138
28.5 Plain TeX and ConTeXt . . . 139

29 TS temperature–entropy diagrams **139**
29.1 TS commands . . . 140
29.2 Shared diagram vocabulary . . . 143
29.3 TS family, scale, and grid keys . . . 144
29.4 TS isenthalp appearance and labels . . . 145
29.5 TS sampling and axis bounds . . . 147
29.6 TS process keys and exported values . . . 148
29.7 Plain TeX and ConTeXt . . . 149

30 HS enthalpy–entropy diagrams **149**
30.1 HS commands . . . 150
30.2 Shared vocabulary and physical ranges . . . 154
30.3 HS switches, scale, and error metric . . . 155
30.4 Isobar grid and units . . . 157
30.5 Isobar appearance and automatic labels . . . 158
30.6 Isobar sampling and HS axis bounds . . . 160
30.7 HS process keys and exported coordinates . . . 161
30.8 Plain TeX and ConTeXt . . . 162

31 PT pressure–temperature diagrams **162**
31.1 PT commands . . . 163
31.2 Families, ranges, and coordinate units . . . 166
31.3 Enthalpy grid and isenthalp presentation . . . 169
31.4 Phase-envelope presentation and sampling . . . 171
31.5 Shared styling, adaptive sampling, and stable names . . . 172
31.6 PT process projection and exports . . . 174
31.7 Plain TeX and ConTeXt . . . 174

32 PGFPlots fluid style **175**

33 Named states and reusable cycles **175**
33.1 Model definition and view keys . . . 177
33.2 One refrigerator, two views . . . 180
33.3 Scopes, units and deliberate alternatives . . . 181
33.4 Model values versus graphical readings . . . 182
33.5 The same construction from Lua . . . 182

34 Steady-flow components and cycle balances **183**
34.1 Component and balance keys . . . 184
34.2 A refrigerator with an explicit compressor efficiency . . . 186
34.3 Limits, consistency and the entropy balance . . . 187
34.4 Supplying a custom relation . . . 188
34.5 A steam power plant: liquid pump and turbine . . . 188
34.6 Machine phase limits and ideal references . . . 191
34.7 Component balances from Lua . . . 192
34.8 An adiabatic pressure loss in a refrigerant line . . . 192
34.9 A cooler with an imposed pressure loss . . . 194
34.10 Prescribed heat, existing outlets and unavailable entropy . . . 195

V Developer reference **195**

35 Architecture **196**
35.1 Lua runtime components and installation . . . 196

36 PGFPlots lifecycle and label emission **197**

37 Adaptive curve sampling **197**

38 Thermodynamic state resolution **197**

39 Naming conventions **198**

40 Lua API: scope, contracts, and safe use **198**
40.1 Loading the module and identifying both versions . . . 198
40.2 SI-unit contract . . . 199
40.3 Errors and recovery boundaries . . . 199
40.4 Library metadata and direct property wrappers . . . 200
40.5 Managed AbstractState objects . . . 201
40.6 Fluid-constant table . . . 202
40.7 Structured diagram interface . . . 202
40.8 Point-record schema and adaptive sampling . . . 203
40.9 Shared option contract and declared capabilities . . . 204

40.10Lua process API 206
40.11The built-in registry and custom diagram types 207
40.12TeX bridge and direct entry points 207

41 Curve records and the autonode bridge **208**

42 Process validation **208**

A Complete executable tutorial definitions **209**
A.1 R717 worksheet and solution 209
A.2 R134a air conditioner 213
A.3 Butane PV construction 217
A.4 R1234yf swimming-pool heat pump 221
A.5 Methane Linde process 225

Command Index **233**

Key and Value Index **234**

Part I
Getting started

1 Choose your starting point

Start with a fluid and a physical question, then choose how much of the picture you want to control. The same thermodynamic implementation serves all three paths below; moving to a manual axis does not change the fluid model.

First figure. Use `\LCPDiagram` with a diagram identifier and a fluid. The package creates the picture, axis, units and default background.

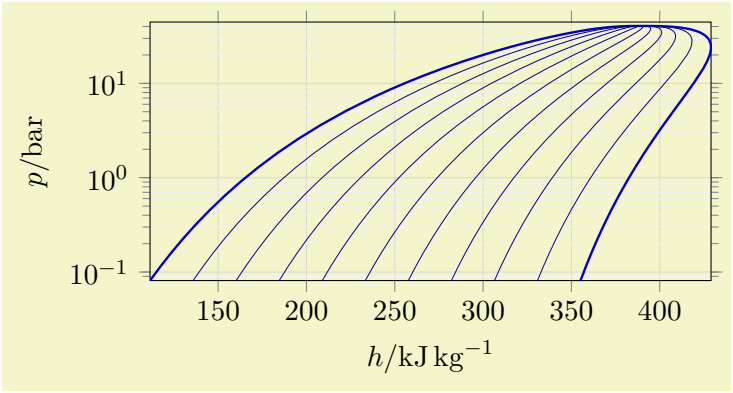
Everyday customization. Keep that command and select curve values, labels and presentation with ordinary keys. Save repeated choices in your own PGF styles; see Section 2.

Complete control. Own the PGFPlots axis, add families and processes individually, or obtain numerical data through Lua and table exports; see Section 3.

1.1 A first thermodynamic picture

Install LuaTeX, PGFPlots and the independent `pgfplots-autonode` package, and make a compatible CoolProp shared library available. The library must match the architecture of the engine. Section 11 explains installation and library selection on each platform. In particular, either configure `library` with `\LCPSetup` before the first calculation, or set `LUACOOLPROP_LIB` when starting the compiler. A library path is machine dependent; it is not a thermodynamic setting to copy from a figure. Compile trusted LaTeX sources with `lualatex -shell-escape`. Shell escape enables arbitrary commands with your permissions; it is needed here for LuaTeX FFI. It is not needed by autonode alone.

This is the complete figure body. R134a is a named pure fluid; the background shows vapour-quality curves between saturated liquid and saturated vapour. The axes give specific enthalpy in $\mathrm{kJ\,kg^{-1}}$ and pressure in bar, with logarithmic pressure. No refrigeration cycle is implied by this background.




```
\LCPDiagram{PH}[fluid=R134a,
  axis options={width=9cm,height=5cm}]
```

Use the following complete LaTeX document around the figure command. The independent diagram library is loaded by `luacoolprop`. The document owns its PGFPlots compatibility setting; LuaCoolProp does not change it.

```
\documentclass{article}
\usepackage{luacoolprop}
\pgfplotsset{compat=1.18}
\begin{document}
\LCPDiagram{PH}[fluid=R134a,
  axis options={width=9cm,height=5cm}]
\end{document}
```

Plain LuaTeX and ConTeXt MkIV use the same figure command and options. These are complete programs, compiled separately by the documentation build. Use `luatex -shell-escape` for the first and `context -luatex -shell-escape` for the second; MkIV uses LuaTeX.

```
\input p-luacoolprop.tex
\pgfplotsset{compat=1.18}
\LCPDiagram{PH}[fluid=R134a,
  axis options={width=9cm,height=5cm}]
\bye
```

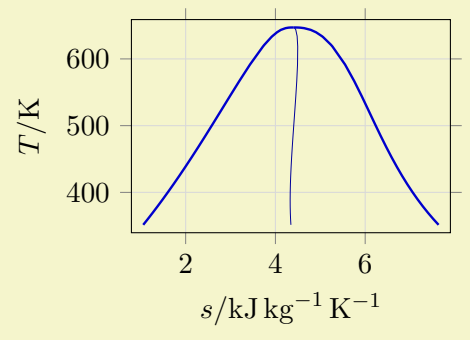
```
\usemodule[t][luacoolprop]
\pgfplotsset{compat=1.18}
\starttext
\LCPDiagram{PH}[fluid=R134a,
  axis options={width=9cm,height=5cm}]
\stoptext
```

1.2 Choose coordinates before choosing curves

All five identifiers use `\LCPDiagram{TYPE}[fluid=...]`. The following map gives the default axis scaling and the supported background families. It describes background generation, not a restriction on process paths. A quality value is undefined at the critical point; the common endpoint of the saturation boundaries is a graphical limit, not a state with a specified quality.

Type	Horizontal coordinate	Vertical coordinate	Background families
PH	specific enthalpy	pressure (log)	quality, isotherm, isentrope, isochore
PV	specific volume (log)	pressure (log)	quality, isotherm, isentrope
TS	specific entropy	temperature	quality, isenthalp
HS	specific entropy	specific enthalpy	quality, isochore, isotherm, isobar
PT	temperature	pressure (log)	phase envelope, isentrope, isenthalp, isochore

For example, switching coordinates and fluid needs no new loading mechanism:




```
\LCPDiagram{TS}[
  fluid=Water,
  quality values={0,0.5,1},
  axis options={width=5.8cm,height=4.4cm}]
```

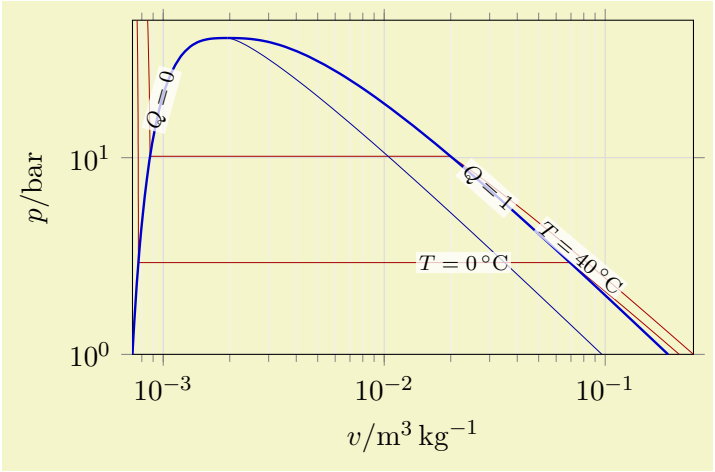
The package treats these projections as peers. Do not copy an unsupported family switch into another projection: TS has no isotherm background, for example, and PT has a single

liquid–vapour coexistence curve rather than separate quality backgrounds. See the capability contract in Section 24. Mixtures and pseudo-pure blends are outside the supported diagram scope. A familiar refrigerant trade name is not evidence that it is a pure fluid.

2 Customize a figure without changing its model

2.1 Select a small, legible network

Choose explicit values when only a few curves are useful. Here the quality values are dimensionless, and the isotherm values use the default Celsius input convention. Pressure bounds are in Pa. Axis unit selectors convert both the coordinates and their labels; changing an axis title alone does not convert the data. The result is a PV background, not an invented process cycle.



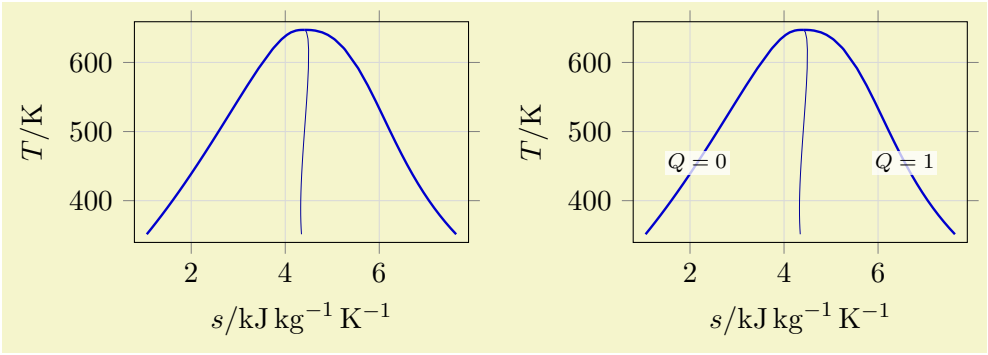

```
\LCPDiagram{PV}[
  fluid=R134a,
  pressure min=100000, pressure max=5000000,
  quality values={0,0.5,1},
  isotherm=true, temperature values={0,40},
  labels=true, sloped labels=true,
  axis options={width=9cm,height=6cm}]
```

Adding labels requires TeX measurement and joint placement. A smaller curve network often makes a better teaching figure than a dense network with every label forced into it. Prefer fewer labelled curves or a larger axis to allowing labels outside the plot rectangle. If a label cannot fit, the documented failure policy decides whether to warn, hide it or stop; hiding a label does not remove the thermodynamic curve.

2.2 Transparent worksheet and solution styles

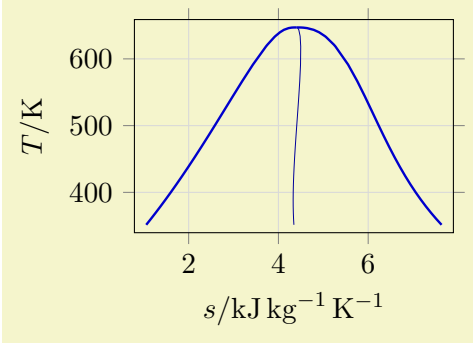
These are *document-defined styles*, not additional package options or installed presets. Their complete definitions are shown. The common namespace makes the two label policies usable in PH, PV, TS, HS and PT calls. They do not change fluids, units, sampling, numerical tolerances or reference states. The solution style enables annotation; it does not solve an exercise or add a cycle. Supply the physical states and transformations separately.

PGF applies an option list in order. The second picture first selects the worksheet style, then the solution style; the latter enables labels again. The surrounding group makes the definitions local.




```
\begingroup
\pgfkeys{
  /luacoolprop/diagram/common/my worksheet/.style={labels=false},
  /luacoolprop/diagram/common/my solution/.style={labels=true}}
\LCPDiagram{TS}[
  fluid=Water, quality values={0,0.5,1},
  my worksheet,
  axis options={width=6cm,height=4.5cm}]
\quad
\LCPDiagram{TS}[
  fluid=Water, quality values={0,0.5,1},
  my worksheet, my solution,
  axis options={width=6cm,height=4.5cm}]
\endgroup
```

An explicit key after a style overrides that assignment. Conversely, a later style can overwrite an earlier explicit key. This example deliberately turns off the labels selected by the solution style:



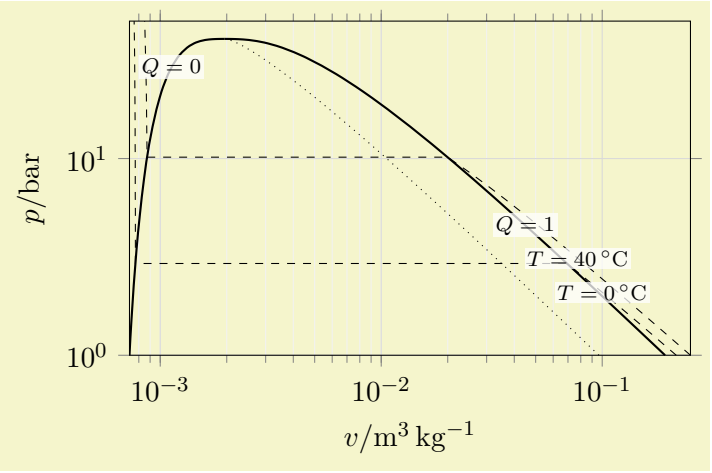

```
\begingroup
\pgfkeys{
  /luacoolprop/diagram/common/my solution/.style={labels=true}}
\LCPDiagram{TS}[
  fluid=Water, quality values={0,0.5,1},
  my solution, labels=false,
  axis options={width=5.8cm,height=4.4cm}]
\endgroup
```

A diagram call also groups its settings: its `axis options` do not change the following independent axis. Styles defined outside a call remain available until their enclosing TeX group ends. This ordinary TeX scope does *not* unload a native library or undo a thermodynamic reference-state selection: those have process-lifetime constraints, documented in Section 18.1.

2.3 Monochrome is a drawing style, not a sampling mode

Use both line patterns and weights, so meaning does not depend on colour. This PV recipe covers each of PV's background families. It is intentionally defined in the PV namespace: a different

projection can have additional families whose colour/style keys must also be set. Family colours are explicit; a common `curve style=black` alone is not a reliable way to override a family colour applied later in the emitted plot options.



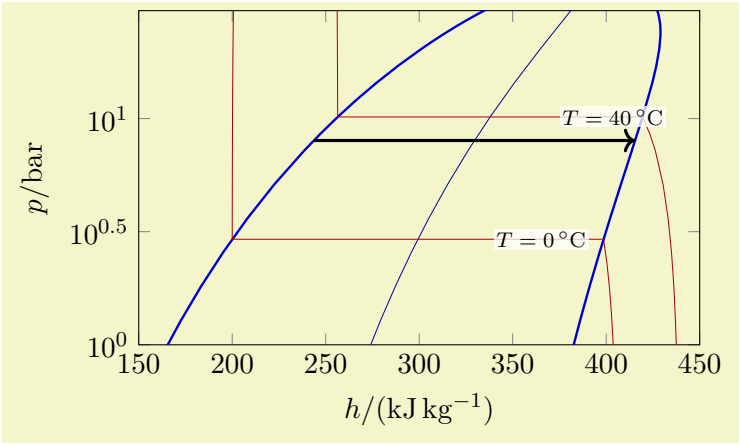

```
\begingroup
\pgfkeys{/luacoolprop/diagram/PV/my monochrome/.style={
  quality color=black, quality boundary color=black,
  isotherm color=black, isentrope color=black,
  boundary style={thick}, interior style={thin,dotted},
  isotherm style={thin,dashed}, isentrope style={thin,dash dot},
  label style={luacoolprop label node,text=black}}}
\LCPDiagram{PV}[
  fluid=R134a,
  pressure min=100000, pressure max=5000000,
  quality values={0,0.5,1},
  isotherm=true, temperature values={0,40},
  my monochrome, labels=true,
  axis options={width=9cm,height=6cm}]
\endgroup
```

There is no automatic "draft means inaccurate, final means correct" profile. For a fast layout check, omit labels or choose fewer background values while keeping numerical tolerances unchanged. Fewer curves change the information shown, not the accuracy of the retained curves. Changing sampling or solver budgets is a separate advanced decision; verify the effect before reusing a figure for quantitative reading. Numerical caching and TikZ externalization have different reuse boundaries (Sections 21 and 22).

3 Own the axis, the process or the data

3.1 An axis with individually selected families and a process

A complete diagram command creates an axis; do not nest it inside another axis. Instead, use the same generic identifiers with `\LCPAddDiagramFamily` and `\LCPAddDiagramProcess`. On a manually owned axis, set the coordinate scales and units yourself, and enable `auto node placement` if you request automatic labels. The following example uses the default PH plot units and shows isobaric evaporation at 8 bar. Its endpoints are saturation states computed by CoolProp, not coordinates estimated from the picture.




```
\begin{tikzpicture}
\begin{axis}[
  width=9cm,height=6cm,ymode=log,
  xlabel={$h/({\rm kJ\,kg^{-1}})$},
  ylabel={$p/{\rm bar}$},
  xmin=150,xmax=450,ymin=1,ymax=30,
  auto node placement]
  \LCPAddDiagramFamily{PH}{quality}[
    fluid=R134a,quality values={0,0.5,1},labels=false]
  \LCPAddDiagramFamily{PH}{isotherm}[
    fluid=R134a,temperature values={0,40},labels=true]
  \LCPAddDiagramProcess{PH}[
    fluid=R134a,type=isobar,pressure=8bar,
    from={quality=0},to={quality=1},
    process style={black,very thick,->}]
\end{axis}
\end{tikzpicture}
```

A process definition and its diagram projection are independent choices. Use `PV`, `TS`, `HS` or `PT` in the same process API when those coordinates answer your question; adapt the manually owned axis units and scales accordingly. The process endpoint export and table APIs provide computed values without reading them back from a rendered PDF. See Sections 27 and 23.

3.2 A direct Lua calculation, without drawing

The Lua facade is also usable from `texlua`. This complete program computes the saturation pressure of ammonia at 273.15 K. Inputs and output of `propsSI` are SI, independently of a diagram's displayed units. It uses the configured library discovery mechanism and reports a calculation error rather than supplying a plausible replacement value.

```
kpse.set_program_name("texlua")
local cp = dofile(assert(kpse.find_file("luacoolprop.lua", "lua")))
local pressure = cp.propsSI("P", "T", 273.15, "Q", 0, "Ammonia")
assert(pressure > 0 and pressure < math.huge,
  "CoolProp did not return a finite positive pressure")
print(string.format("Saturation pressure: %.9g Pa", pressure))
```

Use the managed state and diagram APIs for reference-state consistency and structured numerical curve records; see Section 40. The low-level native wrappers retain CoolProp's contracts. In particular, TeX grouping is not a reset mechanism for global native configuration.

4 Find the right family of keys

A bare option is interpreted in the namespace of the command that receives it. The same spelling can therefore belong to different families. The exhaustive entries describe precedence, defaults

and units; this map is a route to them, not another set of defaults.

Package setup: `/luacoolprop`. Native library discovery, global fluid/reference policy, formatting and cache defaults. Start with `\LCPSetup`; see Section 20.

Diagram: `/luacoolprop/diagram/common`. Shared fluid, label, grid and presentation controls. Projection namespaces such as `/luacoolprop/diagram/TS` add or specialize capabilities; see Sections 26 and 24.

Process: `/luacoolprop/process/common`. The constraint, endpoint states, process appearance and exports, specialized by the selected projection; see Section 27.

Axis: `/pgfplots`. Size, bounds, ticks and the independent autonode axis keys. Pass native axis options through `axis options` in a complete diagram or write them directly on your own axis; see Section 14.

Label node: `/tikz`. Font, fill and node appearance. Keep the documented LuaCoolProp node styles when adding a text colour or other adjustment; replacing a style replaces its previous contents. Placement controls are documented in Section 15 and in the independent autonode manual.

4.1 Recipes by task

Prepare a worksheet and a correction. Use the local label styles above; then follow the R717 teaching tutorial. A correction cycle needs the exercise's physical constraints.

Choose only a few isolines. Use an explicit `quality values`, `temperature values` or other supported family list. Consult the capability map before switching coordinates.

Change units or symbols. Use the semantic axis-unit keys and the quantity-symbol keys in the diagram reference. Keep decimal input written with a full stop, even when output uses a localized `siunitx` format.

Add a cycle and obtain numbers. Construct process paths from states and constraints, then use the endpoint exports or explicit tables (Sections 27 and 23). The butane PV and methane Linde tutorials cover progressively more demanding constraints.

Reuse work between compilations. Cache numerical curves or externalize complete pictures; read the respective invalidation rules before expecting a speedup. Label layout remains a separate operation, owned by the independent autonode library.

Find a particular command or assignment. Use the command/key indices and the normative reference. The short path above does not remove any numerical, geometric or typographic controls from that API.

Part II
Tutorials

5 A teaching pair for a real-fluid refrigeration machine

This tutorial is written for an instructor preparing both a problem sheet and its model solution. It follows the progressive style of the TikZ and PGFPlots tutorials: we first decide what the picture must communicate, then construct a clean student diagram, and finally add the thermodynamic cycle, state names, process arrows, and reading aids for the teacher version.

The case study is a single-stage R717 (ammonia) refrigeration machine operating between 3 bar and 10 bar. All thermodynamic curves and cycle segments are generated from the external CoolProp shared library.

What you will build

The student receives a dense but unannotated PH chart suitable for graphical readings. The instructor uses the same source with one Boolean switch enabled; the second PDF adds the five-state cycle, direction arrows, process names, projection lines, and numerical enthalpy readings. Keeping one graphical source prevents the problem sheet and model solution from drifting apart.

5.1 The exercise behind the diagram

The refrigerant follows this idealized sequence:

1. state A is dry saturated vapor at 3 bar;
2. $A \to B$ is reversible adiabatic compression to 10 bar, hence an isentrope;
3. $B \to C$ is isobaric desuperheating at 10 bar until saturated vapor is reached;
4. $C \to D$ is complete isobaric condensation at 10 bar;
5. $D \to E$ is throttling to 3 bar, hence an isenthalpic process; and
6. $E \to A$ is complete isobaric evaporation at 3 bar.

A PH chart is particularly effective here. Both heat-exchanger processes are horizontal because they are isobaric, while the throttle is vertical because specific enthalpy is conserved. The compressor follows a constant-entropy curve. The geometry therefore carries physical meaning before any number is read.

The exercise can ask learners to locate the states, complete a state table, identify whether the vapor behaves as an ideal gas during compression, and calculate compressor work, evaporator heat, coefficient of performance, mass flow, compressor power, and latent enthalpy of vaporization.

5.2 One shared source and two thin drivers

The example uses three files:

File	Responsibility
`tutorial/r717-worksheet.tex`	Disables the solution overlay.
`tutorial/r717-solution.tex`	Enables the solution overlay.
`tutorial/source/r717-machine-diagram.tex`	Owns the axis, isolines, and conditional solution overlay.

The student driver is deliberately small:

Student driver

```
\documentclass[tikz,border=5mm]{standalone}
\newif\ifRSevenSolution
\RSevenSolutionfalse
\input{tutorial/source/r717-machine-diagram.tex}
```

The teacher driver changes only the Boolean value:

Teacher driver

```
\documentclass[tikz,border=5mm]{standalone}
\newif\ifRSevenSolution
\RSevenSolutiontrue
\input{tutorial/source/r717-machine-diagram.tex}
```

This pattern scales well to a course containing many statement/correction pairs. Shared data, colors, axis limits, and thermodynamic grids are edited once. Only pedagogical information belongs inside `\ifRSevenSolution`.

5.3 Step 1: choose the worksheet's reading domain

Start from the physical operating region, not from the full validity range of the fluid model. The cycle lies between 3 and 10 bar, but learners need enough context to recognize the saturation dome and extrapolate nearby families. The example therefore shows 0.07 to 140 bar, and 100 to 2200 $\mathrm{kJ\,kg^{-1}}$.

Axis prepared for graphical reading

```
\LCPSetup{fluid=R717}
\begin{tikzpicture}
  \begin{axis}[
    lcp fluid=R717,
    width=24cm,
    height=15cm,
    xmin=100,
    xmax=2200,
    ymin=0.07,
    ymax=140,
    ymode=log,
    xlabel={Specific enthalpy $h$ ($\mathrm{kJ\,kg^{-1}}$)},
    ylabel={Pressure $p$ (bar)},
    title={R717 pressure-enthalpy diagram},
    grid=both,
    clip mode=individual,
    auto node placement,
    auto node algorithm=repair,
    auto node bbox mode=oriented,
    auto node failure mode=hide-low-priority
  ]
    % Isoline families and, conditionally, the solution go here.
  \end{axis}
\end{tikzpicture}
```

The logarithmic pressure scale is essential: equal vertical distances represent equal pressure ratios. This is also why projected horizontal readings must be drawn with axis coordinates rather than with page coordinates.

5.4 Step 2: select purposeful isoline families

An instructional chart should contain enough curves to support interpolation, but not so many that the cycle disappears. Explicit value lists make the worksheet stable and auditable. They are preferable here to a preset because the list itself documents what learners are expected to read.

Family	Values used	Teaching role
quality	$0, 0.1, \ldots, 1$	Locate wet-vapor states and estimate Q_E.
isotherm	$-60, -40, \ldots, 240\,^{\circ}\mathrm{C}$	Read state temperatures and inspect superheated-vapor behavior.
isentrope	$1, 2, \ldots, 8\,\mathrm{kJ\,kg^{-1}\,K^{-1}}$	Construct the ideal compressor path and read entropy.
isochore	$0.01, 0.02, \ldots, 20\,\mathrm{m^3\,kg^{-1}}$	Estimate specific volume and test the incompressible-flow assumption.

The quality family draws the saturation boundaries as well as the interior quality lines:

Saturation dome and quality lines

```
\LCPAddPHQuality[
  pressure min=7000,
  pressure max=14000000,
  quality values={0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1},
  quality color=black,
  quality boundary color=black,
  interior style={line width=.32pt},
  boundary style={line width=.9pt},
  labels=true,
  quality label every=1,
  quality label pos=.18,
  quality label sloped=true
]
```

Add the three remaining families with visually distinct encodings. Color is helpful on screen, but dash pattern and line weight still distinguish the families after grayscale photocopying.

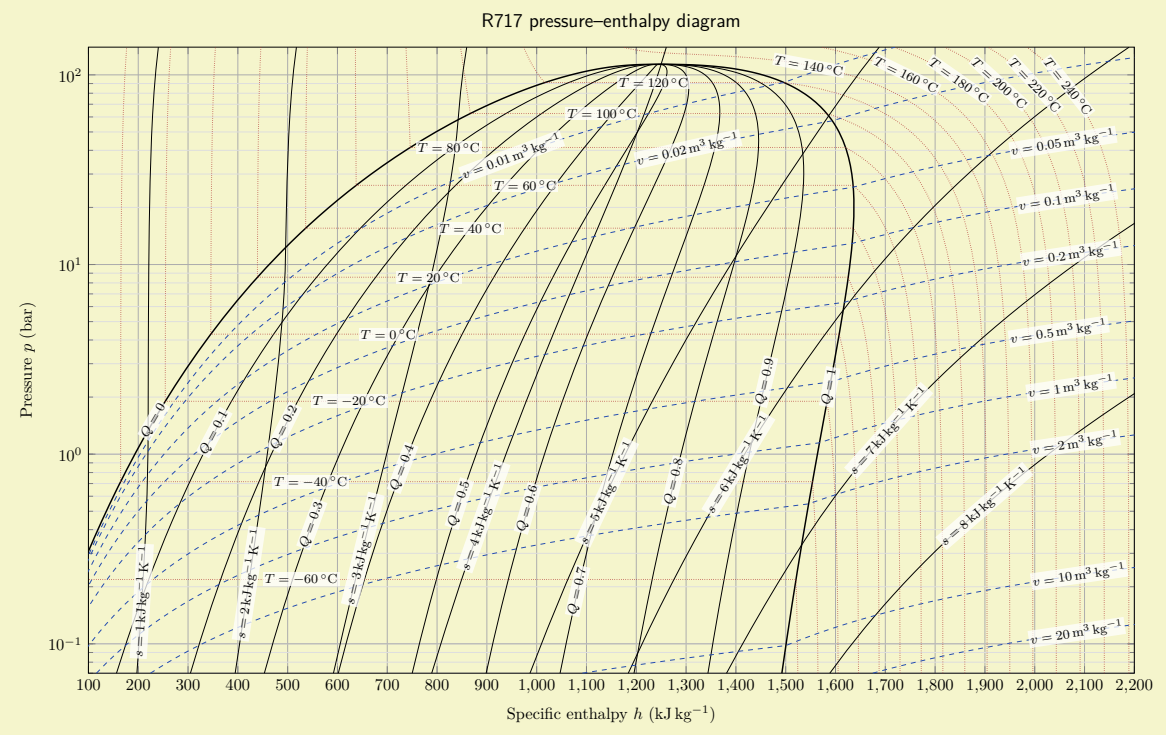
Temperature, entropy, and volume grids

```
\LCPAddPHIsotherms[
  temperature values={-60,-40,-20,0,20,40,60,80,
    100,120,140,160,180,200,220,240},
  isotherm color=RSevenTemperature,
  isotherm style={line width=.34pt,densely dotted},
  labels=true,
  isotherm label every=1,
  isotherm label sloped=true
]
\LCPAddPHIsentropes[
  entropy values={1,2,3,4,5,6,7,8},
  isentrope color=black,
  isentrope style={line width=.32pt},
  labels=true,
  isentrope label every=1,
  isentrope label sloped=true
]
\LCPAddPHIsochores[
  specific volume values={0.01,0.02,0.05,0.1,0.2,0.5,1,2,10,20},
  isochore color=RSevenVolume,
  isochore style={line width=.36pt,dashed},
  labels=true,
  isochore label every=1,
  isochore label sloped=true
]
```

All automatic labels still belong to `pgfplots-autonode`. The tutorial does not calculate label boxes or collision positions. The `hide-low-priority` failure policy is appropriate for a dense background grid: losing one redundant isoline label is preferable to obscuring the cycle.

5.5 Result: the student worksheet diagram

The first generated PDF is the diagram supplied with the exercise. It contains no answer-specific marks, so it can be printed, embedded in a statement, or used as a digital annotation background.




```
\RSevenSolutionfalse
\RSevenDiagram
```

Compile the standalone file with the same external library used by the package:

Build only the student diagram

```
export LUACOOLPROP_LIB=/path/to/libCoolProp.dylib
lualatex -shell-escape -output-directory=tutorial \
  tutorial/r717-worksheet.tex
```

The repository build places the PDF next to its source and the transcript in `tutorial/logs`. Building through the script is recommended because it also checks the log and uses the same policy as the manual.

5.6 Step 3: establish the five thermodynamic states

Before drawing arrows, write each state as two independent properties or as one property plus the conserved property of the process. This makes the solution reviewable: every segment states its own physical hypothesis.

The following table was calculated with CoolProp 8.0.0. It is intentionally more precise than a student can read from a printed chart.

State	p/bar	T/°C	Q	v/m^3 kg^{-1}	h/kJ kg^{-1}	s/kJ kg^{-1} K^{-1}
A	3	−9.22	1	0.40597	1596.90	6.22769
B	10	74.39	—	0.15947	1765.42	6.22769
C	10	24.91	1	0.12853	1628.94	5.80290
D	10	24.91	0	0.001658	462.76	1.89037
E	3	−9.22	0.1233	0.051386	462.76	1.93059

Quality outside the saturation dome

Vapor quality is a two-phase property. State B is superheated vapor, so CoolProp reports quality as undefined there; writing $Q_B = 1$ may be an informal way to say "all vapor", but it is not a valid equilibrium quality. The distinction is worth making explicit in a model solution.

Graphical values should be reported with graphical precision. Depending on print size and interpolation, reasonable readings from the generated diagram are $h_A \simeq 1580$, $h_B \simeq 1760$, $h_C \simeq 1610$, and $h_D = h_E \simeq 460\,\mathrm{kJ\,kg^{-1}}$. The difference from the table is not an error: the table evaluates the equation of state directly, whereas the exercise assesses chart reading. CoolProp versions and enthalpy reference states should always be recorded when exact absolute enthalpies are published.

5.7 Step 4: draw each process from its invariant

The five calls below are inside `\ifRSevenSolution`. Notice how the option `type` names the invariant rather than merely describing the appearance of the line.

Compression $A \to B$. State A is fixed by pressure and saturated-vapor quality. The target pressure fixes B, because the process supplies constant entropy.

Reversible adiabatic compression

```
\LCPAddPHProcess[
  type=isentrope,
  from={pressure=3bar,quality=1},
  to={pressure=10bar},
  color=RSevenCycle,
  style={line width=1.35pt,-{Latex[length=2.2mm]}},
  name=r717-ab,
  export coordinates=RSevenAB,
  log coordinates=true
]
```

Desuperheating $B \to C$. The high-side pressure is conserved. The entropy copied from state A identifies B; `quality=1` identifies saturated vapor at C.

High-pressure desuperheating

```
\LCPAddPHProcess[
  type=isobar,
  pressure=10bar,
  from={entropy=6.2276889kJkgK},
  to={quality=1},
  color=RSevenCycle,
  style={line width=1.35pt,-{Latex[length=2.2mm]}},
  name=r717-bc,
  export coordinates=RSevenBC
]
```

Condensation $C \to D$. Both endpoints are saturation states on the same isobar.

Complete condensation

```
\LCPAddPHProcess[
  type=isobar,
  pressure=10bar,
  from={quality=1},
  to={quality=0},
  color=RSevenCycle,
  style={line width=1.35pt,-{Latex[length=2.2mm]}},
  name=r717-cd,
  export coordinates=RSevenCD
]
```

Throttle $D \to E$. The inlet is saturated liquid at 10 bar. The outlet pressure is enough to resolve E, because throttling conserves specific enthalpy.

Isenthalpic expansion

```
\LCPAddPHProcess[
  type=isenthalp,
  from={pressure=10bar,quality=0},
  to={pressure=3bar},
  color=RSevenCycle,
  style={line width=1.35pt,-{Latex[length=2.2mm]}},
  name=r717-de,
  export coordinates=RSevenDE
]
```

Evaporation $E \to A$. The inlet enthalpy equals h_D; the outlet is dry saturated vapor. Supplying the high-precision inlet value makes the independently generated segment meet the throttle exactly.

Low-pressure evaporation

```
\LCPAddPHProcess[
  type=isobar,
  pressure=3bar,
  from={enthalpy=462.75654kJkg},
  to={quality=1},
  color=RSevenCycle,
  style={line width=1.35pt,-{Latex[length=2.2mm]}},
  name=r717-ea,
  export coordinates=RSevenEA
]
```

The repeated style is kept visible in the tutorial so every call can be copied independently. In a larger course project, factor it into a TikZ style such as `r717 cycle/.style={line width=1.35pt,-Latex}`. The five `name` values become TikZ path names. The five `export coordinates` values are bare control-sequence prefixes, without a leading backslash. For example, the first call globally defines `\RSevenABFromX`, `\RSevenABFromY`, `\RSevenABToX`, and `\RSevenABToY`. Enabling `log coordinates` on that call also records the two plotted coordinate pairs in the terminal and log file.

5.8 Step 5: add state names and reading aids

The process commands calculate and name the thermodynamic paths. Load TikZ's `intersections` library, then define each state as the meeting point of its incoming and outgoing process. No enthalpy–pressure pair is copied from a calculation table, so the annotations follow the paths automatically when the fluid or operating conditions change.

States obtained from named-path intersections

```
\usetikzlibrary{intersections}
\path[name intersections={of=r717-ea and r717-ab,by=RSevenA}];
\path[name intersections={of=r717-ab and r717-bc,by=RSevenB}];
\path[name intersections={of=r717-bc and r717-cd,by=RSevenC}];
\path[name intersections={of=r717-cd and r717-de,by=RSevenD}];
\path[name intersections={of=r717-de and r717-ea,by=RSevenE}];

\filldraw[draw=RSevenCycle,fill=white,line width=1pt]
(RSevenA) circle[radius=2pt];
\filldraw[draw=RSevenCycle,fill=white,line width=1pt]
(RSevenB) circle[radius=2pt];
\filldraw[draw=RSevenCycle,fill=white,line width=1pt]
(RSevenC) circle[radius=2pt];
\filldraw[draw=RSevenCycle,fill=white,line width=1pt]
(RSevenD) circle[radius=2pt];
\filldraw[draw=RSevenCycle,fill=white,line width=1pt]
(RSevenE) circle[radius=2pt];
\node[r717 state,anchor=south east] at (RSevenA) {A};
\node[r717 state,anchor=south west] at (RSevenB) {B};
\node[r717 state,anchor=south east] at (RSevenC) {C};
\node[r717 state,anchor=south east] at (RSevenD) {D};
\node[r717 state,anchor=north east] at (RSevenE) {E};
```

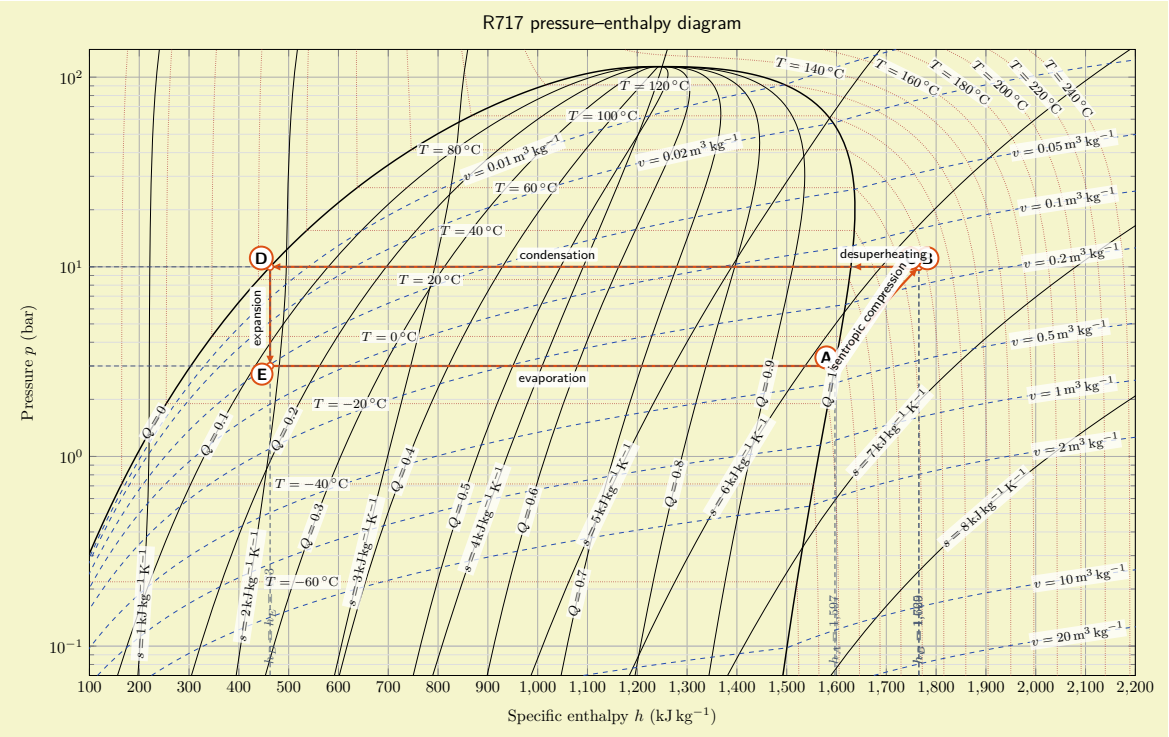
For reading aids, combine the intersection with a coordinate on the axis edge. The TikZ operators `|-` and `-|` copy one canvas component from each coordinate. This remains correct when the axis is resized and when the pressure axis is logarithmic.

Projecting an enthalpy reading

```
\coordinate (RSevenAxisBottom) at (axis cs:100,.07);
\coordinate (RSevenLabelBottom) at (axis cs:100,.075);
\path (RSevenA |- RSevenAxisBottom) coordinate (RSevenABottom);
\path (RSevenA |- RSevenLabelBottom) coordinate (RSevenALabel);
\draw[RSevenGuide,densely dashed,line width=.55pt]
(RSevenABottom) - (RSevenA);
\node[font=\sffamily\scriptsize,anchor=west,rotate=90,
text=RSevenGuide]
at (RSevenALabel)
{$h_A=\pgfmathprintnumber[fixed,precision=0]{\RSevenABFromX}$};
```

Here `\RSevenABFromX` is the numeric horizontal plot coordinate exported by the process command. With the tutorial's enthalpy scale it is expressed in $\mathrm{kJ\,kg^{-1}}$; `\RSevenABFromY` is the corresponding pressure in bar. The suffixes `ToX` and `ToY` provide the other endpoint. The complete source uses the same technique for the four projected enthalpies, enough to reproduce a model reading without covering the property grid. Process-name nodes are placed manually because their instructional composition is deliberate. Automatic isoline labels remain the responsibility of `pgfplots-autonode`.

5.9 Result: the model-solution diagram




```
\RSevenSolutiontrue
\RSevenDiagram
```

The result is suitable as a model solution: the thermodynamic grid remains available for checking interpolations, while the orange cycle has a clear direction and each process is named.

5.10 Step 6: turn readings into an energy balance

The chart is an intermediate model, not the end of the solution. Retain units at every step and avoid using more significant figures than the diagram supports.

Using the direct CoolProp values, the specific compressor work is

$$w_{A\to B} = h_B - h_A = 1765.42 - 1596.90 = 168.52\,\mathrm{kJ\,kg^{-1}}.$$

The heat received by the fluid in the evaporator is

$$q_{E\to A} = h_A - h_E = 1596.90 - 462.76 = 1134.14\,\mathrm{kJ\,kg^{-1}},$$

and the ideal coefficient of performance is therefore

$$\mathrm{COP} = \frac{q_{E\to A}}{w_{A\to B}} \simeq 6.73.$$

Coarser hand readings from the generated diagram give approximately $w_{A\to B} = 180\,\mathrm{kJ\,kg^{-1}}$, $q_{E\to A} = 1120\,\mathrm{kJ\,kg^{-1}}$, and COP $\simeq 6.2$. Both sets are defensible when their origin and precision are stated.

For an evaporator load of $9.0 \times 10^4\,\mathrm{kJ\,h^{-1}}$, first convert the rate:

$$\dot{Q}_{E\to A} = \frac{9.0 \times 10^4}{3600}\,\mathrm{kJ\,s^{-1}} = 25\,\mathrm{kW}.$$

Then

$$\dot{m} = \frac{\dot{Q}_{E\to A}}{q_{E\to A}} \simeq \frac{25\,\mathrm{kJ\,s^{-1}}}{1134.14\,\mathrm{kJ\,kg^{-1}}} = 2.20 \times 10^{-2}\,\mathrm{kg\,s^{-1}},$$

and

$$\dot{W}_{A\to B} = \dot{m}\, w_{A\to B} \simeq 3.71\,\mathrm{kW}.$$

A useful dimensional check

A result of $22\,\mathrm{kg\,s^{-1}}$ is three orders of magnitude too large for a 25 kW evaporator in this example. The intended value is about $22\,\mathrm{g\,s^{-1}}$, or $2.2\times10^{-2}\,\mathrm{kg\,s^{-1}}$. Keeping both the power and the specific enthalpy in kilojoule-based units exposes the conversion immediately.

At state A, the latent enthalpy of vaporization is the horizontal distance between the saturated-liquid and saturated-vapor boundaries at the low-side pressure. A chart reading gives roughly $1.3\times10^{3}\,\mathrm{kJ\,kg^{-1}}$.

5.11 Embedding the two PDFs in course material

The standalone output can be reused without recompiling the thermodynamic network inside every handout. For example:

Problem-sheet inclusion

```
\begin{figure}[htbp]
  \centering
  \includegraphics[width=\linewidth]{tutorial/r717-worksheet.pdf}
  \caption{R717 PH diagram for graphical readings.}
\end{figure}
```

The model solution changes only the file name:

Model-solution inclusion

```
\begin{figure}[htbp]
  \centering
  \includegraphics[width=\linewidth]{tutorial/r717-solution.pdf}
  \caption{R717 refrigeration cycle and model readings.}
\end{figure}
```

Compile the diagram PDFs before the statement and correction. The repository script builds these independent documents as well. The manual itself evaluates the shared drawing definitions directly and does not import these PDFs.

5.12 Adapting the tutorial safely

For a different exercise, proceed in this order:

1. change the fluid and operating pressures;
2. recalculate every state from two independent properties;
3. choose axis bounds around the new cycle;
4. select explicit isolines appropriate to the questions;
5. update process constraints before moving any annotation;
6. compile the blank version and check that no answer is visible;
7. compile the model solution and verify every arrow, state, and projected reading; and
8. redo the energy balance with units carried through every equation.

Do not obtain a new fluid merely by replacing `R717` in the finished graphic. State coordinates, saturation temperatures, entropy values, volume grids, axis limits, and annotations all depend on the fluid.

If the background becomes crowded, first reduce the number of labelled curves with the family-specific `... label every` keys. Then adjust preferred positions or use a per-curve override. Do not add a separate label-placement algorithm to the tutorial: collision avoidance belongs to the required external `pgfplots-autonode` library.

5.13 Instructor's final checklist

Before distributing the exercise, verify the following:

- the student PDF contains the property network but no cycle or numerical answer;
- the model solution follows the stated process order and every arrow points from the inlet to the outlet;
- $A \to B$ follows an isentrope, $D \to E$ is vertical in PH coordinates, and both heat exchangers are horizontal;
- quality is only reported inside or on the saturation dome;
- graphical readings have realistic precision;
- exact values identify the CoolProp version when reproducibility matters;
- power, specific energy, and mass-flow units are mutually consistent; and
- both PDFs can be regenerated with `scripts/build-examples.sh`.

The complete executable source is `tutorial/source/r717-machine-diagram.tex`. It is intentionally kept beside its two drivers so instructors can copy the three-file pattern directly into their own teaching repository.

6 Advanced tutorial: reconstructing an R134a air conditioner

This tutorial develops a complete graphical and numerical solution for a real-fluid air conditioner. It focuses on the refrigeration process: the working fluid, four device models, cycle construction, and energy balances. Every diagram is generated by LuaCoolProp rather than imported as an image.

The exercise is more demanding than the R717 example in the preceding chapter. The initial state is superheated rather than saturated, condensation ends at a specified temperature rather than at an imposed quality, and the graphical construction is built in stages. We shall reproduce that reasoning while making every thermodynamic assumption explicit and every computed value reproducible.

What the six generated pages show

The executable file `tutorial/r134a-air-conditioner.tex` produces one six-page PDF. Pages 1 and 2 identify phase regions and the low-pressure ideal-gas limit. Pages 3, 4, and 5 construct states 1, 2, and 3 successively. Page 6 introduces state 4, closes the cycle, projects the readings, and reports values obtained directly from the process endpoint exports.

6.1 Translate the statement into thermodynamic constraints

The working fluid is R134a and the mass flow rate is $\dot{m} = 0.1\,\mathrm{kg\,s^{-1}}$. The four states and four device models are:

State or path	Given data	Consequence in a PH diagram
1	$p_1 = 3\,\mathrm{bar}$, $T_1 = 5\,^\circ\mathrm{C}$	Superheated-vapour state at the intersection of an isobar and an isotherm.
$1 \to 2$	Isentropic compression, $p_2/p_1 = 6$	$p_2 = 18\,\mathrm{bar}$ and $s_2 = s_1$.
$2 \to 3$	Isobaric cooling, $T_3 = 60\,^\circ\mathrm{C}$	Horizontal path at 18 bar ending on the $60\,^\circ\mathrm{C}$ isotherm.
$3 \to 4$	Adiabatic throttle to $p_4 = p_1$	Steady-flow energy balance gives $h_4 = h_3$, hence a vertical path.
$4 \to 1$	Isobaric evaporation and superheating	Horizontal path at 3 bar returning to state 1.

This translation is the central modelling step. Words such as “isentropic”, “isobaric”, and “isenthalpic” become the `type` values passed to `\LCPAddPHProcess`; no curve is drawn merely because it looks similar to the expected correction.

6.2 Use a stable plotting domain

An initial worksheet spans a broad R134a domain. The progressive correction benefits from a closer view around the cycle, so all six generated pages share the following axis:

Common axis for every construction stage

```
\LCPSetup{fluid=R134a}
\begin{tikzpicture}
  \begin{axis}[
    lcp fluid=R134a,
    width=17cm,
    height=11cm,
    xmin=140,
    xmax=520,
    ymin=.7,
    ymax=55,
    ymode=log,
    xlabel={Specific enthalpy $h$ ($\mathrm{kJ\,kg^{-1}}$)},
    ylabel={Pressure $p$ (bar)},
    grid=both,
    clip mode=individual
  ]
    % One construction stage goes here.
  \end{axis}
\end{tikzpicture}
```

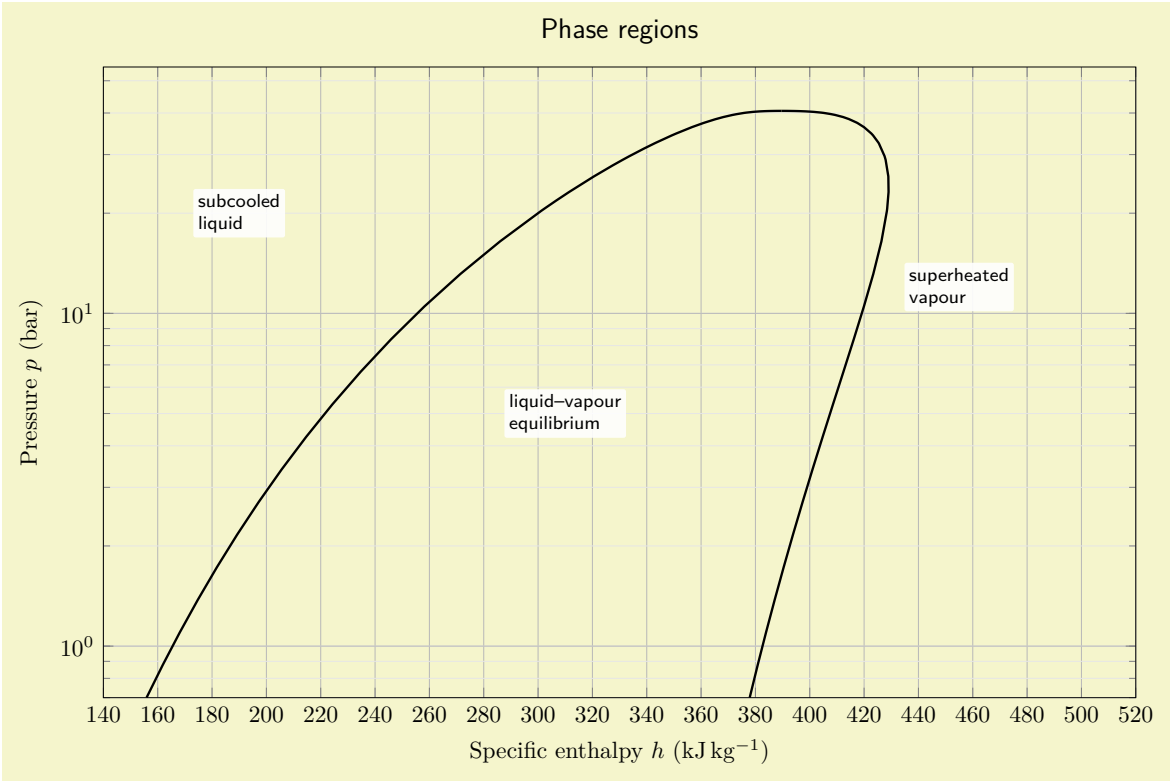
The pressure range extends beyond both operating pressures and shows the top of the saturation dome. The enthalpy range leaves room for the compressed-vapour state and for annotations. Keeping identical limits across the sequence makes movement from one construction to the next visually meaningful.

6.3 Stage 1: identify the three phase regions

The saturated-liquid and saturated-vapour boundaries are the two curves generated by qualities 0 and 1. They delimit subcooled liquid on the left, liquid–vapour equilibrium below the dome, and superheated vapour on the right. The two branches remain distinct at the low-pressure triple point and converge to the single critical state at the top of the dome. LuaCoolProp refines this neighbourhood automatically and gives both curves the same canonical critical endpoint, independently of the selected pure fluid.

Saturation boundaries used as a phase map

```
\LCPAddPHQuality[
  pressure min=70000,
  pressure max=5500000,
  quality values={0,1},
  quality boundary color=black,
  boundary style={line width=1pt},
  labels=false
]
\node at (axis cs:190,20) {subcooled liquid};
\node at (axis cs:310,5) {liquid-vapour equilibrium};
\node at (axis cs:455,12) {superheated vapour};
```




```
\LCPSetup{fluid=R134a}\RAirDiagram{0}
```

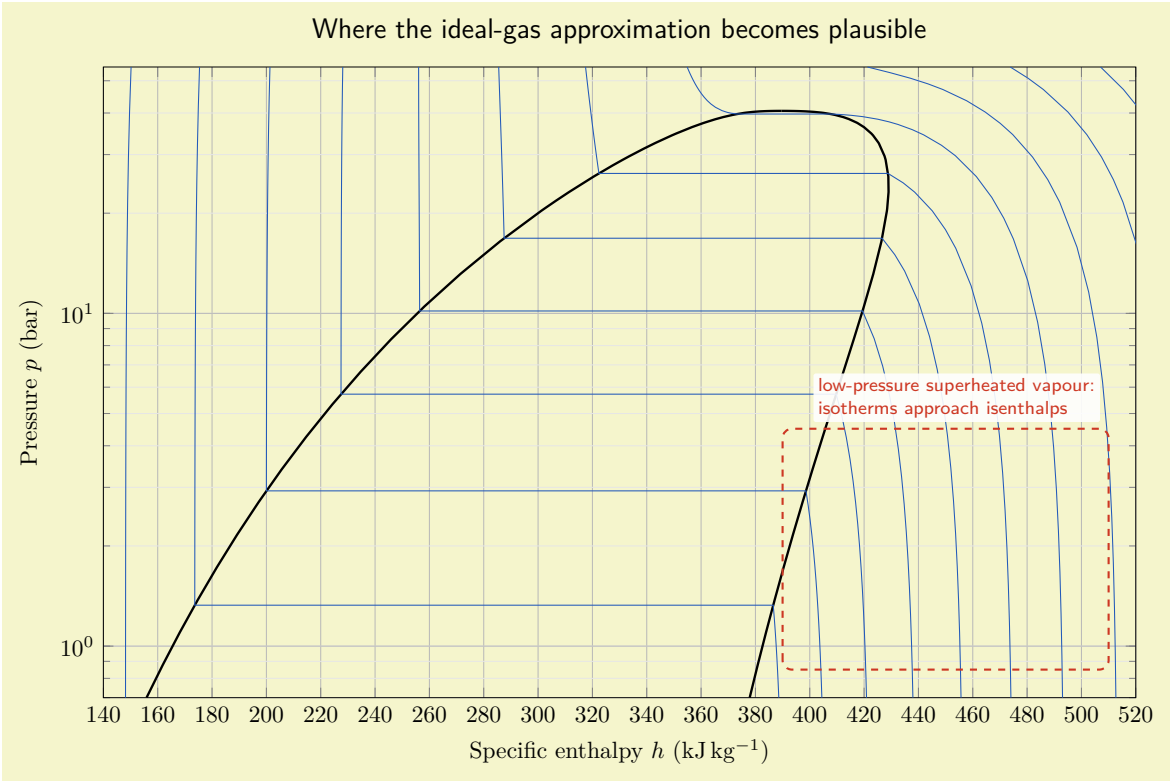
Only phase labels are placed manually. If isoline labels are enabled in an adapted version, their automatic placement remains the exclusive responsibility of `pgfplots-autonode`.

6.4 Stage 2: recognize the ideal-gas region

For an ideal gas, specific enthalpy depends only on temperature. Isotherms and isenthalps therefore coincide, and isotherms become nearly vertical in a PH diagram. Plotting a temperature family shows that this behaviour is approached for sufficiently low-pressure superheated vapour, far to the right of the dome.

Isotherms used as a diagnostic rather than decoration

```
\LCPAddPHIsotherms[
  pressure min=70000,
  pressure max=5500000,
  temperature values={-40,-20,0,20,40,60,80,100,120,140,160},
  isotherm color=blue,
  isotherm style={line width=.45pt},
  labels=false
]
```




```
\LCPSetup{fluid=R134a}\RAirDiagram{1}
```

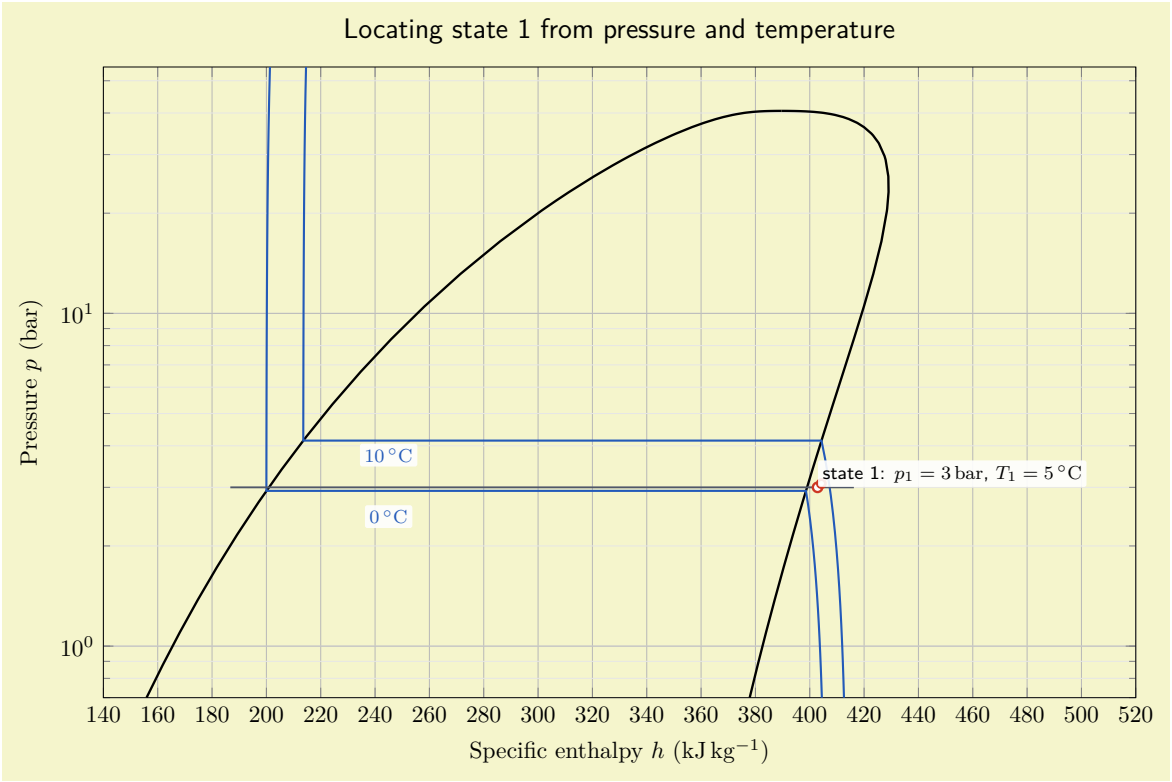
This is a qualitative conclusion, not a sharp phase boundary. The dashed box on the generated page indicates where the isotherms are already close to vertical over the displayed pressure interval.

6.5 Stage 3: locate state 1 from two measured properties

State 1 is defined by two independent properties. We name a 3 bar isobar and a 5 °C isotherm, then ask TikZ to find their intersection. The neighbouring 0 and 10 °C isotherms provide a graphical interpolation exercise.

State 1 as an intersection of named thermodynamic paths

```
\LCPAddPHProcess[
  type=isobar,
  pressure=3bar,
  from={temperature=-10C},
  to={temperature=20C},
  name=r134a-low-isobar
]
\LCPAddPHProcess[
  type=isotherm,
  temperature=5C,
  from={pressure=2bar},
  to={pressure=5bar},
  name=r134a-state-one-isotherm,
  style={draw=none}
]
\path[name intersections={of=r134a-low-isobar and
  r134a-state-one-isotherm,by=RAirOne}];
\fill (RAirOne) circle[radius=2pt];
\node[above right] at (RAirOne) {1};
```




```
\LCPSetup{fluid=R134a}\RAirDiagram{2}
```

CoolProp gives $h_1 = 402.876\,\mathrm{kJ\,kg^{-1}}$ and $s_1 = 1.74078\,\mathrm{kJ\,kg^{-1}\,K^{-1}}$. A chart reading near $403\,\mathrm{kJ\,kg^{-1}}$ is consistent with the resolution of the grid.

6.6 Stage 4: construct the compressor outlet

The compression ratio gives

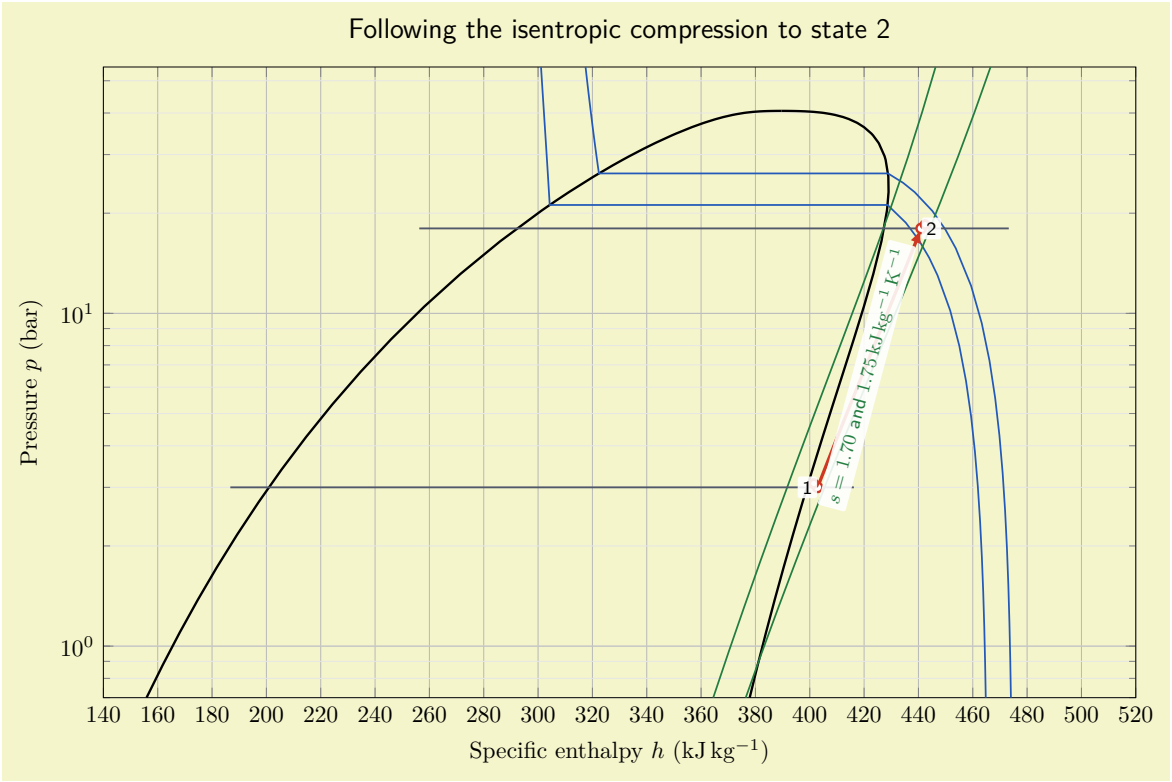
$$p_2 = 6p_1 = 18\,\mathrm{bar}.$$

The compressor is adiabatic and reversible in the model, so the path is an isentrope. The process command can compute its conserved entropy directly from state 1:

Isentropic compression from state 1 to state 2

```
\LCPAddPHProcess[
  type=isentrope,
  from={pressure=3bar,temperature=5C},
  to={pressure=18bar},
  name=r134a-one-two,
  export coordinates=RAirOneTwo,
  color=red,
  style={line width=1.4pt,-{Latex[length=2.2mm]}}
]
```

As in the correction, two neighbouring isentropes at 1.70 and $1.75\,\mathrm{kJ\,kg^{-1}\,K^{-1}}$ explain the graphical construction. State 2 is the intersection of the named compression path and a named 18 bar guide; it is not placed with an explicit coordinate.




```
\LCPSetup{fluid=R134a}\RAirDiagram{3}
```

The exported macros give $h_2 = 441.019\,\mathrm{kJ\,kg^{-1}}$ and $T_2 = 346.205\,\mathrm{K} = 73.05\,^\circ\mathrm{C}$. A hand reading of the same chart at moderate resolution would give roughly $440\,\mathrm{kJ\,kg^{-1}}$ and $73\,^\circ\mathrm{C}$.

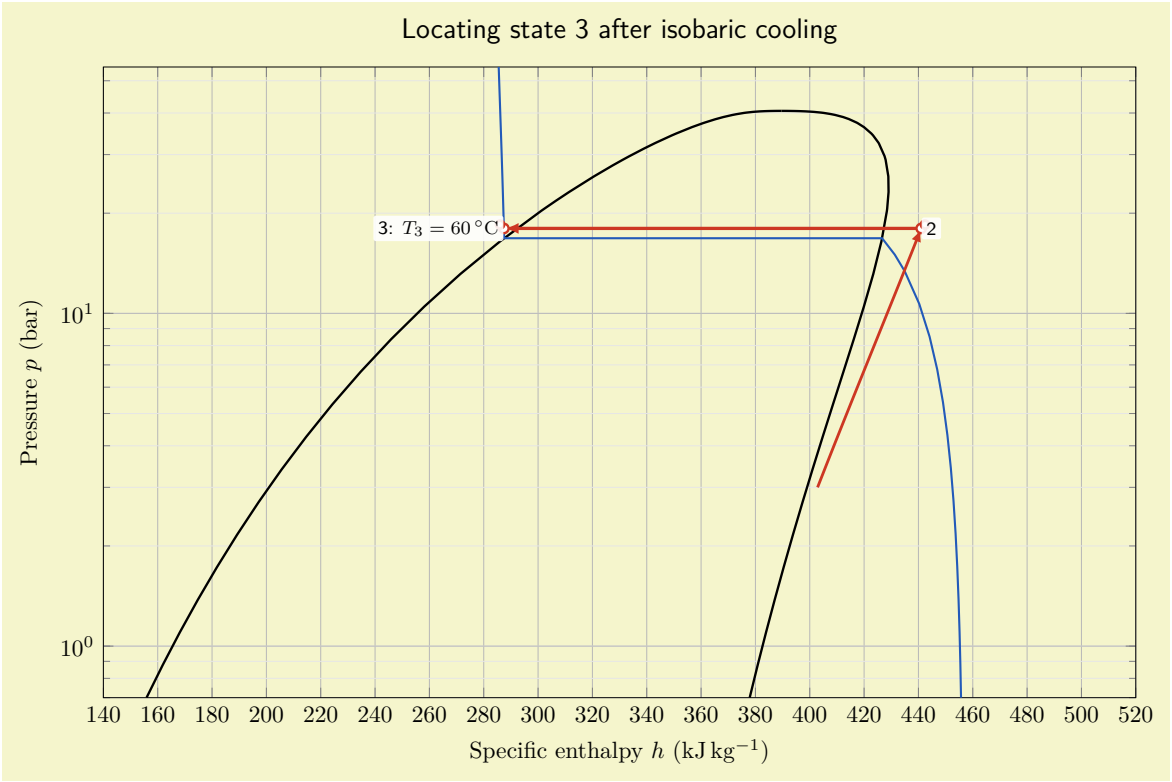
6.7 Stage 5: cool at constant pressure to state 3

State 3 is fixed by $p_3 = p_2 = 18\,\mathrm{bar}$ and $T_3 = 60\,^\circ\mathrm{C}$. The complete $2 \to 3$ path crosses the saturation dome: it includes desuperheating, condensation, and a small subcooling region. Using one isobaric process is physically clearer than splitting that continuous heat-exchanger path merely to match the visual phases.

The high-pressure heat-exchanger path

```
\LCPAddPHProcess[
  type=isobar,
  pressure=18bar,
  from={entropy=1740.778583426JkgK},
  to={temperature=60C},
  name=r134a-two-three,
  export coordinates=RAirTwoThree,
  color=red,
  style={line width=1.4pt,-{Latex[length=2.2mm]}}
]
```

The explicit entropy in the initial-state specification is the full-precision value calculated for state 1. It makes the independently generated $1 \to 2$ and $2 \to 3$ paths meet exactly. The TikZ intersection remains the sole source of the displayed point position.




```
\LCPSetup{fluid=R134a}\RAirDiagram{4}
```

The endpoint export gives $h_3 = 287.411\,\mathrm{kJ\,kg^{-1}}$. A graphical estimate near $285\,\mathrm{kJ\,kg^{-1}}$ is consistent with the limited precision of a printed chart.

6.8 Stage 6: throttle, close the cycle, and locate state 4

For a steady, adiabatic throttle with no useful shaft work and negligible changes in kinetic and potential energy, the open-system first law gives $h_4 = h_3$. In PH coordinates the path is therefore vertical.

Expansion and low-pressure return

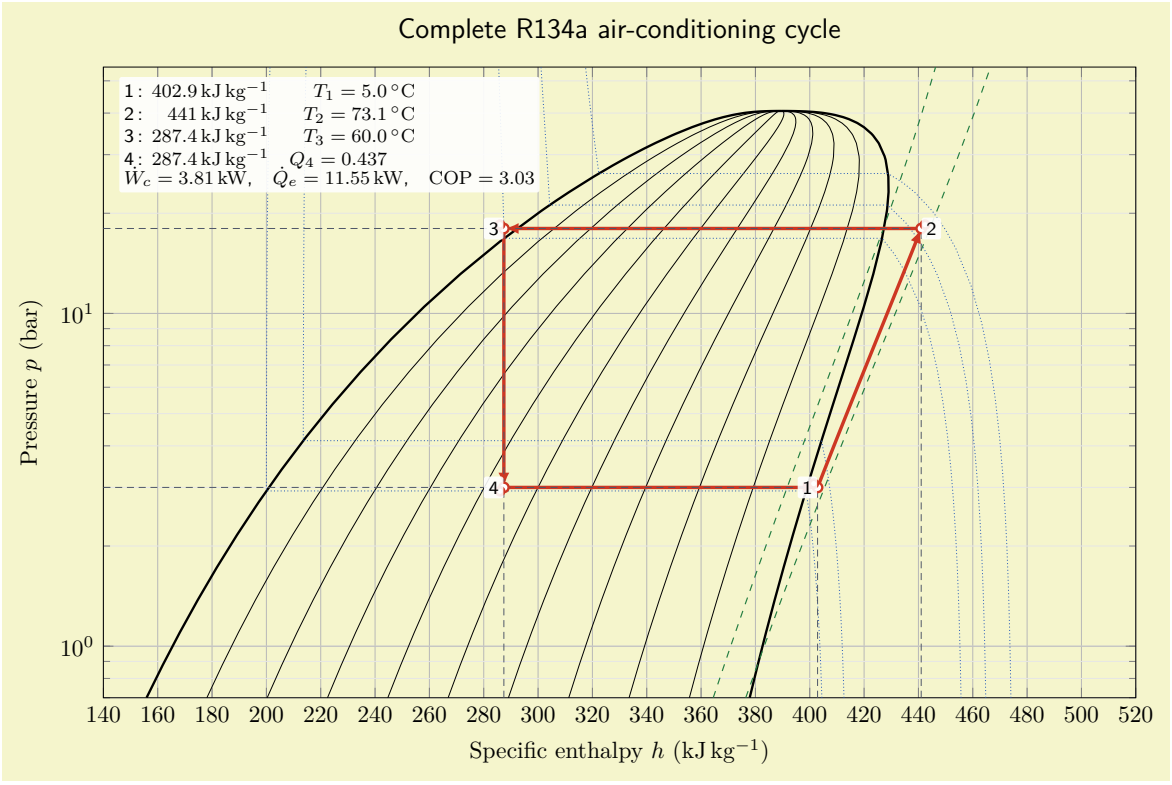
```
\LCPAddPHProcess[
  type=isenthalp,
  from={pressure=18bar,temperature=60C},
  to={pressure=3bar},
  name=r134a-three-four,
  export coordinates=RAirThreeFour,
  color=red,
  style={line width=1.4pt,-Latex}
]
\LCPAddPHProcess[
  type=isobar,
  pressure=3bar,
  from={enthalpy=287.4107701858kJkg},
  to={temperature=5C},
  name=r134a-four-one,
  export coordinates=RAirFourOne,
  color=red,
  style={line width=1.4pt,-Latex}
]
```

The four state positions are now obtained from adjacent named paths:

Closing the topology with four intersections

```
\path[name intersections={of=r134a-four-one and r134a-one-two,
  by=RAirOne}];
\path[name intersections={of=r134a-one-two and r134a-two-three,
  by=RAirTwo}];
\path[name intersections={of=r134a-two-three and r134a-three-four,
  by=RAirThree}];
\path[name intersections={of=r134a-three-four and r134a-four-one,
  by=RAirFour}];
```

Quality lines at 0.4 and 0.5 bracket state 4. CoolProp resolves the process endpoint as $T_4 = 273.822\,\mathrm{K} = 0.672\,^\circ\mathrm{C}$ and $Q_4 = 0.4367$, compared with the correction's graphical estimates of 1 °C and 0.43.




```
\LCPSetup{fluid=R134a}\RAirDiagram{5}
```

The final page deliberately keeps both the construction curves and the cycle. The student can see why every state exists, while the compact result box uses the numbers exported by exactly the same process calls that draw the red paths.

6.9 Obtain every numerical value from process exports

The option `export coordinates=RAirOneTwo` exports plotted coordinates and SI state properties. The macros `\RAirOneTwoFromX` and `\RAirOneTwoToX` contain h_1 and h_2 in the displayed unit.

The compressor outlet temperature is available as `\RAirOneTwoToTemperatureK`. The throttle similarly defines `\RAirThreeFourToQuality`. No separate property table is embedded in the diagram source.

Powers and coefficient of performance from exported values

```
\pgfmathsetmacro{\CompressorPower}{
  .1*(\RAirOneTwoToX-\RAirOneTwoFromX)}
\pgfmathsetmacro{\EvaporatorPower}{
  .1*(\RAirFourOneToX-\RAirFourOneFromX)}
\pgfmathsetmacro{\AirConditionerCOP}{
  \EvaporatorPower/\CompressorPower}

\typeout{State 2 temperature (K):
  \RAirOneTwoToTemperatureK}
\typeout{State 4 quality:
  \RAirThreeFourToQuality}
```

Because enthalpy coordinates are in $\mathrm{kJ\,kg^{-1}}$ and mass flow is in $\mathrm{kg\,s^{-1}}$, their product is directly in kilowatts:

$$\dot{W}_c = \dot{m}(h_2 - h_1) = 0.1(441.019 - 402.876) = 3.81\,\mathrm{kW},$$

$$\dot{Q}_e = \dot{m}(h_1 - h_4) = 0.1(402.876 - 287.411) = 11.55\,\mathrm{kW},$$

and

$$\mathrm{COP} = \frac{\dot{Q}_e}{\dot{W}_c} = 3.03.$$

The positive evaporator power means that the refrigerant receives heat from the passenger compartment, thereby cooling it.

For comparison, using the temperatures requested by the statement gives

$$\mathrm{COP}_{\mathrm{Carnot}} = \frac{T_1}{T_3 - T_1} = \frac{278.15}{333.15 - 278.15} = 5.06.$$

The modelled cycle is therefore less effective than the reversible benchmark, as expected.

Keep graphical and computed precision distinct

A printed chart yields approximate readings, while the generated process exports evaluate the selected CoolProp state model directly. The resulting cycle gives a COP of 3.03. In teaching material, state explicitly whether a number is a graphical estimate or an equation-of-state evaluation; do not silently mix the two levels of precision in one energy balance.

6.10 Files, rebuild, and adaptation checklist

The driver and its six-stage implementation are, respectively,

`tutorial/r134a-air-conditioner.tex`
`tutorial/source/r134a-air-conditioner-diagrams.tex`.

Rebuild every tutorial, example, log, index, and the manual with:

Reproducible repository build

```
export LUACOOLPROP_LIB=/path/to/libCoolProp.dylib
./scripts/build-examples.sh
```

When adapting this example to a different air conditioner:

1. replace the fluid and the two operating pressures;
2. express every state with two independent properties;
3. let each `\LCPAddPHProcess` compute its own path;
4. give all cycle paths stable `name` values;

5. obtain state coordinates only through named-path intersections;
6. export numerical endpoints instead of maintaining a second table;
7. compare graphical and equation-of-state precision explicitly; and
8. verify the signs and units of every steady-flow energy balance.

7 PV tutorial: butane in a domestic gas lighter

This tutorial uses a rigid lighter reservoir containing liquid and vapour butane to build a reproducible LuaCoolProp teaching sequence. The statement, graphical reasoning, and numerical checks are presented in English; every diagram is generated from CoolProp data.

The four tasks are to identify phase regions, recognize the ideal-gas limit, locate the initial state, and predict the effect of heating the closed rigid reservoir. A separate mechanical ignition analysis is outside this PV tutorial.

What the five generated pages show

The executable file `tutorial/butane-pv.tex` produces one five-page PDF. Page 1 is the unannotated student worksheet. Page 2 identifies the phase regions and critical isotherm. Page 3 overlays the ideal-gas prediction on the 515 K isotherm. Page 4 obtains the initial tank state from a TikZ intersection and reports its pressure, specific volume, and mass. Page 5 draws the isochoric heating path and obtains the new vapour quality from the same process call.

7.1 Translate the exercise into thermodynamic data

The working fluid is pure normal butane. CoolProp accepts `Butane` for this fluid. The tank dimensions and initial state are

$$V = (4\,\mathrm{cm})(1\,\mathrm{cm})(1\,\mathrm{cm}) = 4.0 \times 10^{-6}\,\mathrm{m}^3, \qquad T_A = 305\,\mathrm{K},$$

and 70% of the butane mass is liquid. LuaCoolProp and CoolProp use vapour mass quality, so the remaining 30% gives

$$x_A = 0.30.$$

For a pure substance the liquid and vapour contain the same molecular species, but `quality` should still be read as a mass fraction in the package API.

The exercise can be assigned progressively:

1. delimit compressed liquid, liquid–vapour equilibrium, and superheated vapour; identify the critical isotherm;
2. use the shape of the 515 K isotherm to identify where the ideal-gas approximation becomes credible;
3. locate A from T_A and x_A, then read p_A and v_A and calculate $m = V/v_A$; and
4. deduce the path followed when the sealed rigid tank is heated and the direction in which its vapour quality changes.

This is an especially useful introduction to PV coordinates because every piece of graphical geometry has a direct physical interpretation: an isotherm has a horizontal two-phase plateau, an isochore is vertical, and the dilute-gas limit is a straight line of slope -1 on logarithmic axes.

7.2 Keep one executable source for all five stages

The tutorial consists of a thin multi-page driver and one shared graphical implementation:

File	Responsibility
`tutorial/butane-pv.tex`	Selects the standalone multi-picture document.
`tutorial/source/butane-pv-diagrams.tex`	Defines the common background, annotations, named construction paths, state exports, and five stages.

The driver contains no thermodynamic value:

Five-page standalone driver

```
\documentclass[tikz,border=5mm,multi={tikzpicture}]{standalone}
\input{tutorial/source/butane-pv-diagrams.tex}
```

Keeping the diagram in one implementation means that a changed pressure range, fluid model, or quality grid updates both the worksheet and every solution stage. No correction is drawn on top of a stale imported image.

7.3 Step 1: build a readable log–log worksheet

The worksheet uses pascals, while LuaCoolProp's default PV pressure coordinate is bar. Set `pressure scale=1` on every generated family and label the Y axis in pascals. Specific volume already uses $\mathrm{m^3\,kg^{-1}}$, so its default scale remains one.

Common PV axis in SI units

```
\LCPSetup{fluid=Butane}
\begin{tikzpicture}
  \begin{loglogaxis}[
    lcp fluid=Butane,
    width=17cm,
    height=11cm,
    xmin=.001,
    xmax=1.4,
    ymin=50000,
    ymax=15000000,
    xlabel={Specific volume $v$ (\si{\cubic\metre\per\kilogram})},
    ylabel={Pressure $p$ (\si{\pascal})},
    grid=both,
    clip mode=individual,
    auto node placement,
    auto node algorithm=repair,
    auto node bbox mode=oriented,
    auto node failure mode=hide-low-priority
  ]
    % Quality curves and isotherms go here.
  \end{loglogaxis}
\end{tikzpicture}
```

The range spans roughly three orders of magnitude in specific volume and two and a half in pressure. A linear plot would compress the liquid states into an unreadable strip. In a `loglogaxis`, equal distances represent equal ratios and positive coordinates are mandatory.

Add the saturated boundaries and interior qualities in blue. Explicit values make the pedagogical interpolation grid reproducible:

Butane quality grid

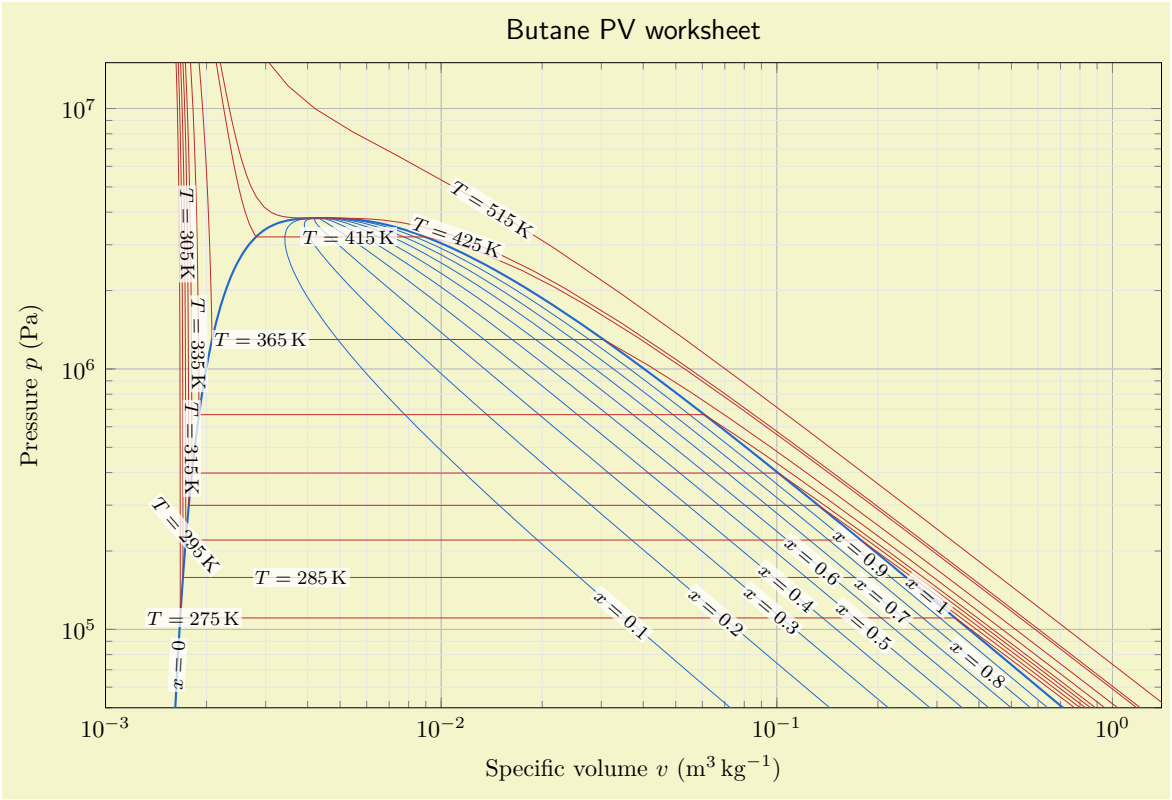
```
\LCPAddPVQuality[
  fluid=Butane,
  pressure min=50000,
  pressure max=15000000,
  pressure scale=1,
  quality values={0,0.1,0.2,0.3,0.4,0.5,
    0.6,0.7,0.8,0.9,1},
  quality symbol=x,
  quality color=blue,
  quality boundary color=blue,
  interior style={line width=.38pt},
  boundary style={line width=.9pt},
  labels=true,
  quality label pos=.2,
  quality label sloped=true
]
```

The red isotherm family is specified in kelvin to match the exercise. Below the critical temperature, LuaCoolProp generates the compressed-liquid branch, the exact saturation-pressure plateau, and the superheated-vapour branch as one named curve.

Butane isotherm grid in kelvin

```
\LCPAddPVIsotherms[
  fluid=Butane,
  pressure min=50000,
  pressure max=15000000,
  pressure scale=1,
  temperature unit=K,
  temperature values={275,285,295,305,315,335,
    365,415,425.125,515,615,715},
  isotherm color=red,
  isotherm style={line width=.42pt},
  curve style for={lcp-pv-T-425p125-K}{
    densely dashed,line width=.9pt},
  labels=true,
  isotherm label pos=.76,
  isotherm label sloped=true
]
```

Both families use adaptive sampling in log–log display coordinates. The executable source tightens `tolerance`, increases `max depth`, and increases `log x weight` so that the critical neighbourhood and narrow liquid branch remain smooth at worksheet size. Label placement is still entirely delegated to `pgfplots-autonode`.




```
\LCPSetup{fluid=Butane}
\LCPDeclareFluid{Butane}
\pgfmathsetmacro{\ButaneCriticalVolume}{1/\LCPFluidRhocritSI}
\ButanePVDiagram{0}
```

The first page is ready to place in an exercise statement. It contains the data needed for graphical reasoning but no phase names, selected state, or heating arrow.

7.4 Step 2: read phase regions and the critical isotherm

The $x = 0$ and $x = 1$ branches bound the liquid–vapour coexistence region and meet at the critical point. Compressed liquid lies to the left, superheated vapour to the right, and two-phase equilibrium lies between them. Intermediate quality curves are meaningful only inside this coexistence region.

Butane's critical constants can be queried instead of copied from a drawing:

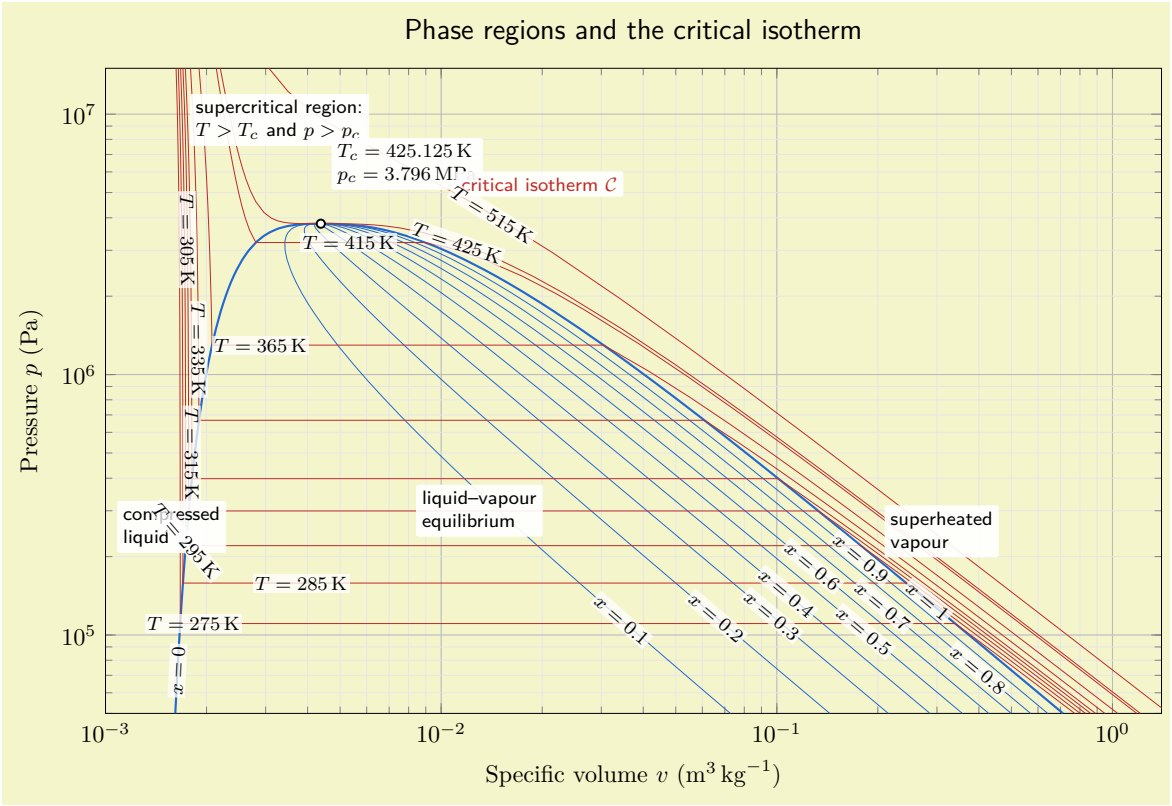
Critical coordinates from the fluid macros

```
\LCPDeclareFluid{Butane}
\pgfmathsetmacro{\ButaneCriticalVolume}{1/\LCPFluidRhocritSI}
% Critical point in an SI-scaled PV axis:
\fill (axis cs:\ButaneCriticalVolume,\LCPFluidPcritSI)
circle[radius=2pt];
```

For the CoolProp release used to build this manual,

$$T_c = 425.125\,\mathrm{K} = 151.975\,^\circ\mathrm{C}, \qquad p_c = 3.796\,\mathrm{MPa}.$$

The dashed curve at 425.125 K is therefore the critical isotherm $\mathcal{C}$ requested by the exercise.




```
\LCPSetup{fluid=Butane}
\LCPDeclareFluid{Butane}
\pgfmathsetmacro{\ButaneCriticalVolume}{1/\LCPFluidRhocritSI}
\ButanePVDiagram{1}
```

Use “supercritical” precisely

Crossing above T_c removes the liquid–vapour phase transition, but a state is strictly in the supercritical region when both $T > T_c$ and $p > p_c$. At $T > T_c$ and $p < p_c$ it is often clearer to say “gas-like single-phase fluid”. Simplified school diagrams sometimes label every state above the critical isotherm as supercritical; an instructor should state which convention is expected.

7.5 Step 3: recognize the ideal-gas region geometrically

For a perfect gas,

$$pV = nRT, \qquad m = nM, \qquad v = \frac{V}{m},$$

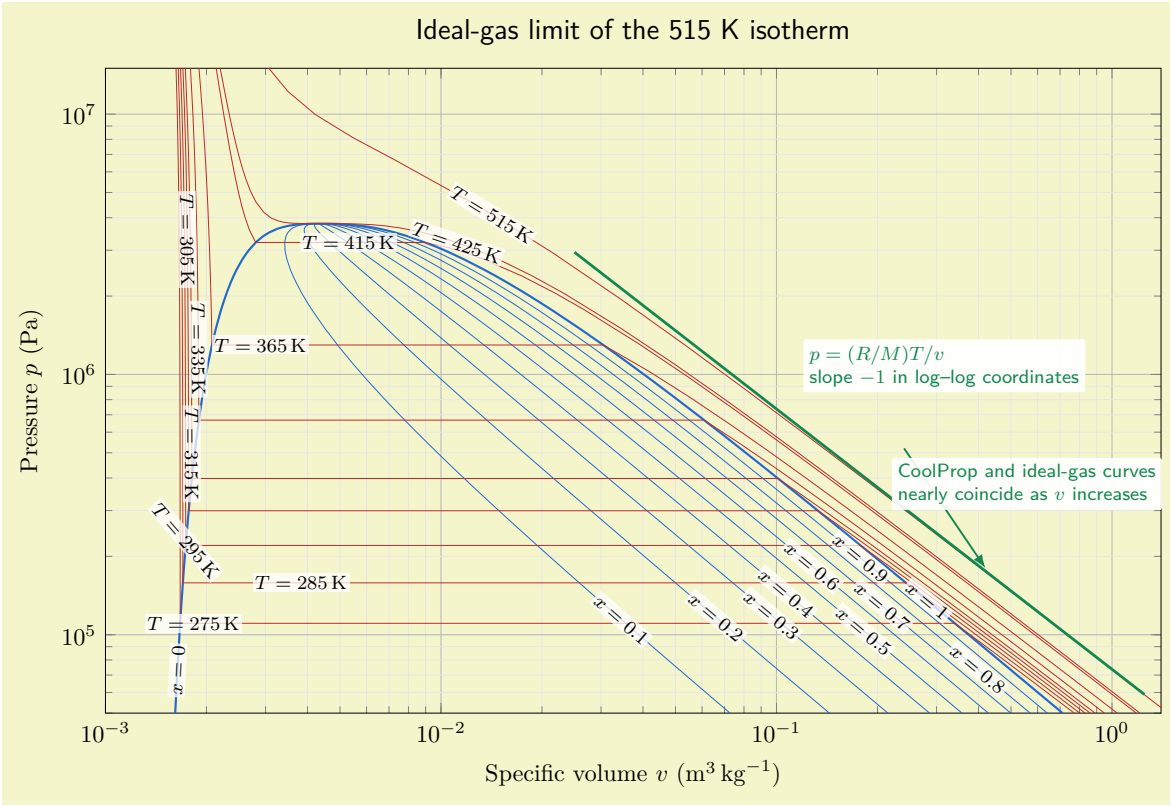
so a fixed-temperature line satisfies

$$pv = \frac{R}{M}T, \qquad \log p = -\log v + \log\left(\frac{R}{M}T\right).$$

An ideal-gas isotherm is therefore a straight line of slope -1 in log–log PV coordinates. For butane, with $M = 0.05812\,\mathrm{kg\,mol^{-1}}$, the 515 K reference is generated by:

Ideal-gas comparison line

```
\addplot[green!60!black,line width=1.15pt,
domain=.025:1.25,samples=2]
{(8.314462618/0.05812*515)/x};
```




```
\LCPSetup{fluid=Butane}
\LCPDeclareFluid{Butane}
\pgfmathsetmacro{\ButaneCriticalVolume}{1/\LCPFluidRhocritSI}
\ButanePVDiagram{2}
```

The green ideal-gas line and red CoolProp isotherm become indistinguishable as specific volume increases and pressure decreases. A useful numerical audit is the compressibility factor

$$Z = \frac{pvM}{RT}.$$

At 515 K, CoolProp gives approximately $Z = 0.9835$ at $v = 0.2\,\mathrm{m^3\,kg^{-1}}$ and $Z = 0.9934$ at $v = 0.5\,\mathrm{m^3\,kg^{-1}}$. The phrase “ideal-gas region” therefore expresses an accuracy decision, not an exact boundary.

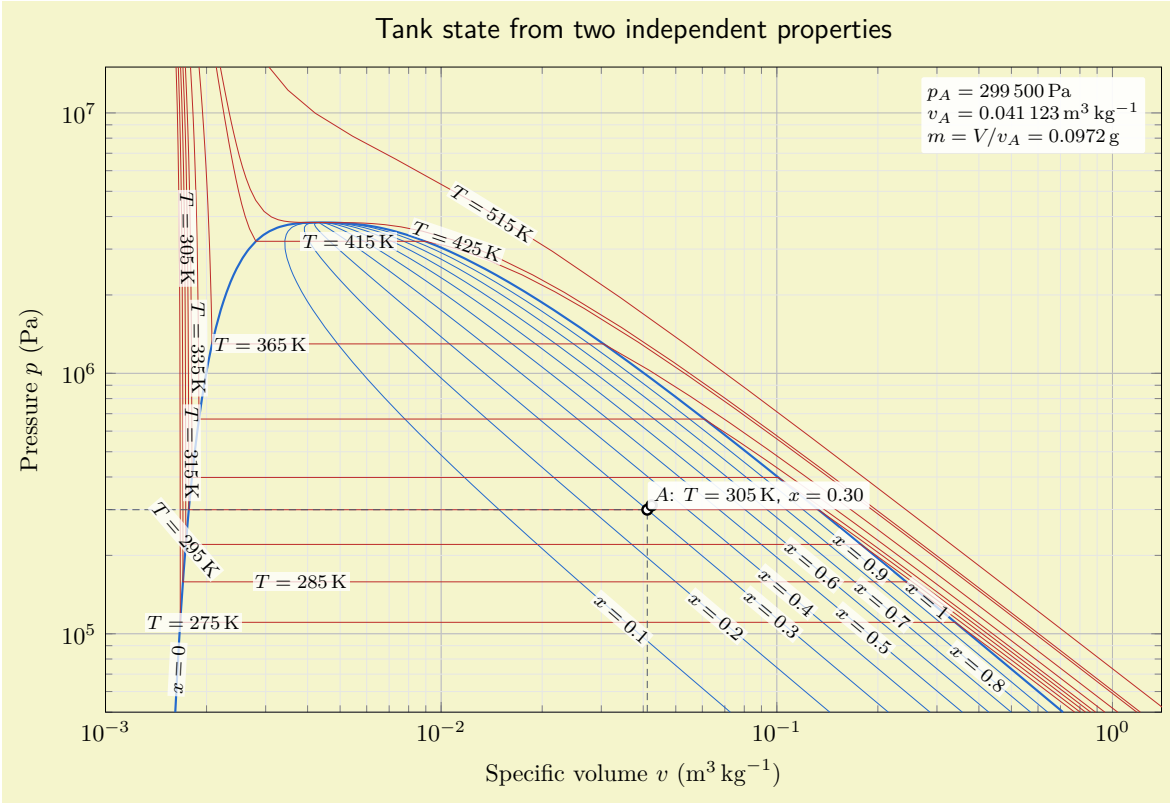
7.6 Step 4: locate the initial state with TikZ intersections

The initial state must satisfy two independent constraints: $T_A = 305\,\mathrm{K}$ and $x_A = 0.30$. Do not copy a coordinate from the model solution. Create two invisible but named thermodynamic paths, then let the TikZ `intersections` library locate their crossing:

Named paths defining state A

```
\LCPAddPVProcess[
  fluid=Butane,
  type=isotherm,
  temperature=305K,
  pressure scale=1,
  from={pressure=1bar},
  to={pressure=10bar},
  name=butane-state-a-isotherm,
  style={draw=none}
]
\LCPAddPVProcess[
  fluid=Butane,
  type=quality,
  quality=0.3,
  pressure scale=1,
  from={temperature=275K},
  to={temperature=415K},
  name=butane-state-a-quality,
  style={draw=none}
]
\path[name intersections={of=butane-state-a-isotherm and
  butane-state-a-quality,by=ButaneA}];
\node[above right] at (ButaneA) {$A$};
```

The named paths extend beyond the immediate crossing so the construction remains stable if the plotting range changes. Their styles are invisible, but their geometry is genuine CoolProp output. State A is consequently tied to the two thermodynamic definitions, not to an explicit pair of plot numbers.




```
\LCPSetup{fluid=Butane}
\LCPDeclareFluid{Butane}
\pgfmathsetmacro{\ButaneCriticalVolume}{1/\LCPFluidRhocritSI}
\ButanePVDiagram{3}
```

The equation-of-state evaluation gives

$$p_A = 2.99460 \times 10^5\,\mathrm{Pa}, \qquad v_A = 4.11233 \times 10^{-2}\,\mathrm{m^3\,kg^{-1}}.$$

These values support graphical estimates of about $3 \times 10^5\,\mathrm{Pa}$ and $4 \times 10^{-2}\,\mathrm{m^3\,kg^{-1}}$. The butane mass is

$$m = \frac{V}{v_A} = \frac{4.0 \times 10^{-6}}{4.11233 \times 10^{-2}} = 9.73 \times 10^{-5}\,\mathrm{kg} = 0.0973\,\mathrm{g}.$$

The printed exercise quite reasonably rounds this graphical result to $10^{-4}\,\mathrm{kg}$.

7.7 Step 5: draw the rigid-tank heating process

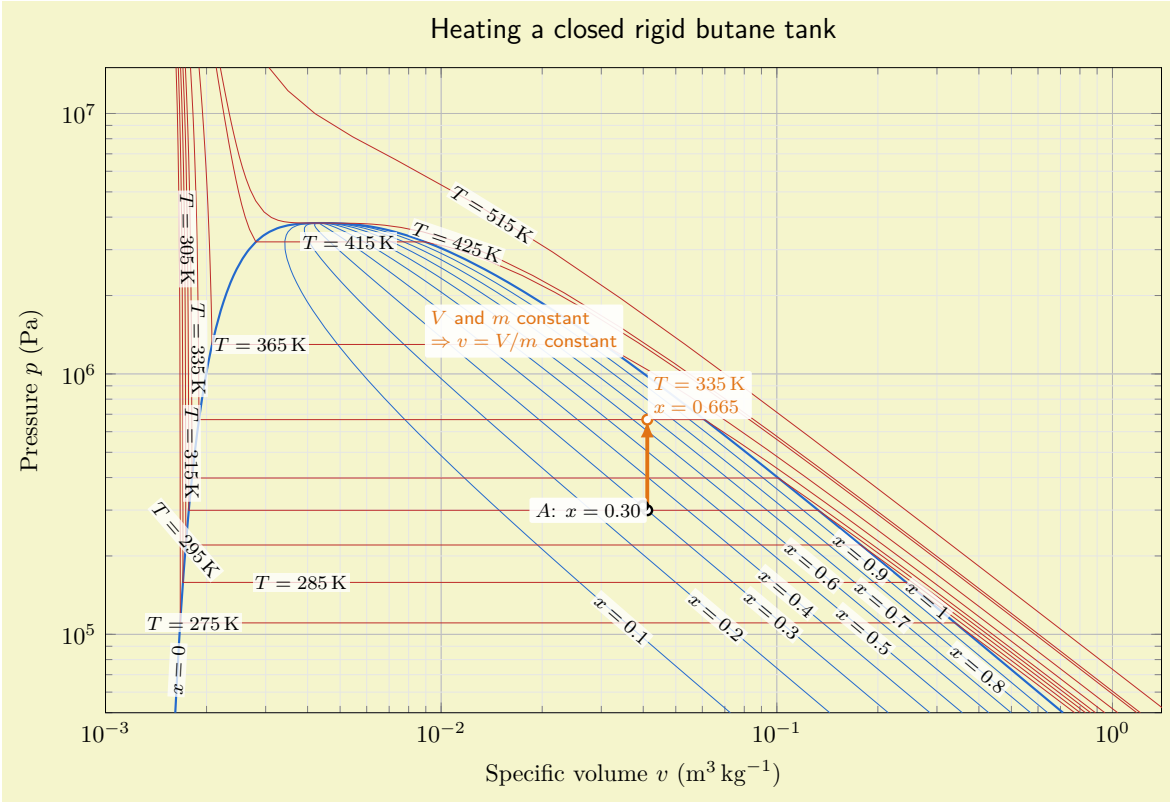
The reservoir is closed, so m is constant. It is rigid, so V is constant. It follows immediately that

$$v = \frac{V}{m} = \text{constant}.$$

The state therefore moves vertically upward on the log–log PV chart as temperature and pressure rise. LuaCoolProp expresses the modelling hypothesis directly with `type=isochore`. The starting state supplies the conserved specific volume; the endpoint needs only its new temperature:

Isochoric heating with exported states

```
\LCPAddPVProcess[
  fluid=Butane,
  type=isochore,
  pressure scale=1,
  from={temperature=305K,quality=0.3},
  to={temperature=335K},
  name=butane-isochoric-heating,
  color=orange,
  style={line width=1.6pt,-Latex},
  export coordinates=ButaneHeatingState,
  coord digits=10,
  log coordinates=true
]
```



```
\LCPSetup{fluid=Butane}
\LCPDeclareFluid{Butane}
\pgfmathsetmacro{\ButaneCriticalVolume}{1/\LCPFluidRhocritSI}
\ButanePVDiagram{4}
```

The endpoint macros come from the same call that draws the orange segment. No second numerical model is maintained:

Use and log the exported PV values

```
Initial pressure:
\qty{\ButaneHeatingStateFromPressureSI}{\pascal}.

Initial specific volume:
\qty{\ButaneHeatingStateFromSpecificVolumeSI}
  {\cubic\metre\per\kilogram}.

Quality at 335 K:
\num{\ButaneHeatingStateToQuality}.

\typeout{Initial butane pressure (Pa):
  \ButaneHeatingStateFromPressureSI}
\typeout{Butane quality at 335 K:
  \ButaneHeatingStateToQuality}
```

At 335 K, the process gives

$$p = 6.67608 \times 10^5\,\mathrm{Pa}, \qquad x = 0.66525.$$

The vapour mass fraction has increased from 0.30 to about 0.665: liquid is evaporating as the tank warms. If heating continues at the same specific volume, the path reaches saturated vapour near 351.70 K and 9.801×10^5 Pa. Beyond that boundary the state is single-phase vapour and quality is undefined; it must not be extrapolated beyond one.

Independent properties on an isochore

For an isochore whose initial state is given by T, Q and whose target specifies only T, LuaCoolProp must retain the target's derived T, ρ pair while requesting every other property from CoolProp. Replacing it midway with the saturation pair p, T is ambiguous.

7.8 Rebuild and adapt the tutorial

The generated PDF and its log are part of the reproducible documentation workflow. Build it together with every example, tutorial, index, and the manual using:

Reproducible repository build

```
export LUACOOLPROP_LIB=/path/to/libCoolProp.dylib
./scripts/build-examples.sh
```

The build places `butane-pv.pdf` next to its driver. Its transcript is written to `tutorial/logs/butane-pv.log`. To adapt the exercise:

1. select another pure fluid with `fluid`;
2. query its critical constants rather than copying butane's values;
3. choose pressure and volume limits from the intended physical problem;
4. state whether temperature values are Celsius or kelvin;
5. define every marked state through two independent properties and named TikZ paths;

6. use `\LCPAddPVProcess` for the physical transformation;
7. export endpoint properties for tables, calculations, and logs; and
8. distinguish diagram-reading precision from direct equation-of-state precision in the model solution.

The central lesson is methodological: the diagram, annotations, and numerical answer should be different views of one thermodynamic construction. When a given property, fluid, or process changes, CoolProp and LuaCoolProp regenerate all three together.

8 Teaching pair: an R1234yf swimming-pool heat pump

This tutorial constructs a student worksheet and a model solution for a swimming-pool heat pump. The heat pump operates cyclically between outside air and pool water. R1234yf flows steadily through an evaporator, compressor, condenser, and expansion valve at $\dot{m} = 5.0\,\mathrm{kg\,s^{-1}}$.

Every background curve, process path, state coordinate, and numerical value is computed through the external CoolProp library; neither diagram depends on an imported chart.

What you will build

The two standalone drivers share one graphical source. The worksheet contains only the dense PH chart needed for graphical construction. The solution adds the four directed processes, states A through D, construction guides, thermodynamic readings, and the condenser power. The operating pressures are themselves calculated from the two specified saturation temperatures; they are not transcribed from a printed graph.

8.1 Translate the statement into four idealized devices

Begin by separating saturation temperatures from inlet and outlet temperatures. The evaporator operates at $-10\,^\circ\mathrm{C}$ and its outlet receives 10 K of superheating, so

$$p_e = p_{\mathrm{sat}}(-10\,^\circ\mathrm{C}), \qquad T_B = 0\,^\circ\mathrm{C}.$$

The condenser operates at $50\,^\circ\mathrm{C}$ and its outlet receives 10 K of subcooling, so

$$p_c = p_{\mathrm{sat}}(50\,^\circ\mathrm{C}), \qquad T_D = 40\,^\circ\mathrm{C}.$$

The four paths can then be expressed without reference to their expected shape:

Path	Device model	Thermodynamic constraint
$A \to B$	Evaporator, no moving part	Isobar at p_e; evaporation followed by superheating to $T_B = 0\,^\circ\mathrm{C}$.
$B \to C$	Adiabatic reversible compressor	Isentrope from p_e to p_c, hence $s_C = s_B$.
$C \to D$	Condenser, no moving part	Isobar at p_c; desuperheating, condensation at $50\,^\circ\mathrm{C}$, then subcooling to $40\,^\circ\mathrm{C}$.
$D \to A$	Insulated expansion valve, no moving part	Isenthalpic throttle, hence $h_A = h_D$.

This translation determines the cycle uniquely. In particular, state A should not be guessed on a quality line: its quality is a result of throttling the subcooled state D to p_e.

8.2 Establish the steady-flow first law

For a one-inlet, one-outlet control volume in steady operation, with heat and non-flow mechanical powers counted positive when received by the refrigerant, the industrial first law is

$$\dot{Q} + \dot{W} = \dot{m}\left[h_{\mathrm{out}} - h_{\mathrm{in}} + \frac{c_{\mathrm{out}}^2 - c_{\mathrm{in}}^2}{2} + g(z_{\mathrm{out}} - z_{\mathrm{in}})\right].$$

Neglecting kinetic- and potential-energy changes gives

$$\dot{Q} + \dot{W} = \dot{m}(h_{\mathrm{out}} - h_{\mathrm{in}}).$$

The sign convention must be stated before interpreting the answer. For the condenser, $\dot{W} = 0$, and the thermal power received by the fluid is

$$\mathcal{P}_1 = \dot{m}(h_D - h_C).$$

A heat pump is expected to reject heat from the refrigerant to the pool in this device, so $\mathcal{P}_1$ should be negative under the stated convention.

The same equation explains the graphical invariants used later. In the adiabatic valve, both $\dot{Q}$ and $\dot{W}$ vanish, giving $h_A = h_D$. In the evaporator and condenser, absence of moving parts makes the enthalpy difference directly proportional to heat received by the fluid.

8.3 Use one source and two thin drivers

The generated pair uses these files:

File	Responsibility
`tutorial/r1234yf-heat-pump-worksheet.tex`	Selects the blank chart.
`tutorial/r1234yf-heat-pump-solution.tex`	Enables the solution overlay.
`tutorial/source/r1234yf-heat-pump-diagram.tex`	Owns the axis, background families, operating-pressure helpers, process paths, intersections, and numerical calculations.

The drivers differ in one Boolean value:

Worksheet and solution switches

```
% In r1234yf-heat-pump-worksheet.tex:
\newif\ifRHeatPumpSolution
\RHeatPumpSolutionfalse

% In r1234yf-heat-pump-solution.tex:
\newif\ifRHeatPumpSolution
\RHeatPumpSolutiontrue
```

This arrangement prevents an edited background from making the problem sheet and model solution geometrically inconsistent.

8.4 Step 1: reproduce the blank PH chart

The worksheet uses SI coordinates, so the tutorial overrides the usual LuaCoolProp scales: h remains in $\mathrm{J\,kg^{-1}}$ and p in pascals. Pressure is logarithmic and enthalpy linear.

Worksheet axis

```
\LCPSetup{fluid=R1234yf}
\begin{tikzpicture}
  \begin{axis}[
    lcp fluid=R1234yf,
    width=24cm,
    height=15cm,
    xmin=100000,
    xmax=620000,
    ymin=100000,
    ymax=8000000,
    ymode=log,
    xlabel={Specific enthalpy $h$ (\si{\joule\per\kilogram})},
    ylabel={Pressure $p$ (\si{\pascal})},
    grid=both,
    clip mode=individual,
    auto node placement,
    auto node algorithm=greedy,
    auto node candidates=21,
    auto node bbox mode=oriented,
    auto node failure mode=hide-low-priority
  ]
    % The three isoline families go here.
  \end{axis}
\end{tikzpicture}
```

The background combines vapour qualities, isotherms in kelvin, and isentropes in SI units:

The three PH families

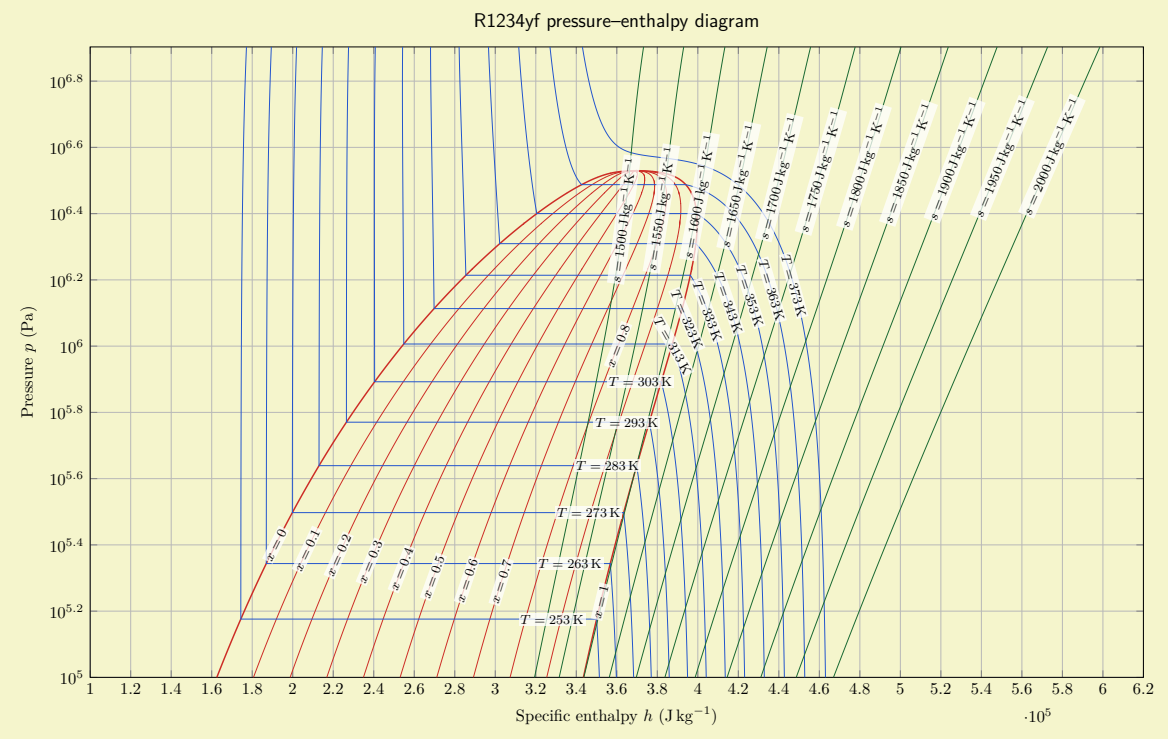
```
\LCPAddPHQuality[
  fluid=R1234yf,
  pressure min=100000,
  pressure max=8000000,
  enthalpy scale=1,
  pressure scale=1,
  quality values={0,0.1,0.2,0.3,0.4,0.5,
    0.6,0.7,0.8,0.9,1},
  quality color=red,
  quality boundary color=red,
  labels=true,
  quality label sloped=true
]

\LCPAddPHIsotherms[
  fluid=R1234yf,
  pressure min=100000,
  pressure max=8000000,
  enthalpy scale=1,
  pressure scale=1,
  temperature unit=K,
  temperature values={253,263,273,283,293,303,313,
    323,333,343,353,363,373},
  isotherm color=blue,
  labels=true,
  isotherm label sloped=true
]

\LCPAddPHIsentropes[
  fluid=R1234yf,
  pressure min=100000,
  pressure max=8000000,
  enthalpy scale=1,
  pressure scale=1,
```

```
  entropy unit=SI,
  entropy values={1500,1550,1600,1650,1700,
    1750,1800,1850,1900,1950,2000},
  isentrope color=green!50!black,
  labels=true,
  isentrope label sloped=true
]
```

The complete implementation also tightens adaptive sampling for print-quality curves. Automatic label placement remains delegated to `pgfplots-autonode`; the tutorial does not contain a competing placement algorithm.




```
\RHeatPumpSolutionfalse
\RHeatPumpDiagram
```

8.5 Step 2: derive both operating pressures

Only temperatures are supplied for the two heat exchangers. Two short, invisible quality paths query the saturation pressure and export it to TeX. The low-pressure helper starts at saturated vapour at $-10\,^{\circ}\mathrm{C}$; the high-pressure helper starts at saturated liquid at $50\,^{\circ}\mathrm{C}$.

Saturation pressures exported as TeX macros

```
\LCPAddPHProcess[
  fluid=R1234yf,
  type=quality,
  quality=1,
  from={temperature=-10C},
  to={temperature=-9C},
  name=r1234yf-low-saturation-helper,
  style={draw=none},
  export coordinates=RHeatPumpLowSaturation
]

\LCPAddPHProcess[
  fluid=R1234yf,
  type=quality,
  quality=0,
  from={temperature=50C},
  to={temperature=49C},
  name=r1234yf-high-saturation-helper,
  style={draw=none},
  export coordinates=RHeatPumpHighSaturation
]
```

The two reusable values are exposed separately:

- `\RHeatPumpLowSaturationFromPressureSI` stores p_e;
- `\RHeatPumpHighSaturationFromPressureSI` stores p_c.

They give

$$p_e = 2.21785\,\text{bar}, \qquad p_c = 13.0235\,\text{bar}.$$

Changing either saturation temperature regenerates the pressure, the four states, the cycle, and the power result in one build.

8.6 Step 3: compute the cycle in dependency order

Although the physical cycle is narrated from A, the most convenient calculation begins at B, whose pressure and temperature are known. First draw the isentropic compressor path and export state C:

Adiabatic reversible compression B to C

```
\LCPAddPHProcess[
  fluid=R1234yf,
  type=isentrope,
  from={pressure=\RHeatPumpLowSaturationFromPressureSI Pa,
    temperature=0C},
  to={pressure=\RHeatPumpHighSaturationFromPressureSI Pa},
  enthalpy scale=1,
  pressure scale=1,
  name=r1234yf-bc,
  export coordinates=RHeatPumpBC,
  color=orange,
  style={very thick,-Latex}
]
```

State C is now the high-pressure state with entropy `\RHeatPumpBCToEntropySI`. Use it to start the complete isobaric condenser path and finish at the specified 40 °C outlet:

Desuperheating, condensation, and subcooling C to D

```
\LCPAddPHProcess[
  fluid=R1234yf,
  type=isobar,
  pressure=\RHeatPumpHighSaturationFromPressureSI Pa,
  from={entropy=\RHeatPumpBCToEntropySI JkgK},
  to={temperature=40C},
  enthalpy scale=1,
  pressure scale=1,
  name=r1234yf-cd,
  export coordinates=RHeatPumpCD,
  color=orange,
  style={very thick,-Latex}
]
```

The expansion valve fixes A by conserving the enthalpy of D while reducing pressure to p_e:

Adiabatic throttling D to A

```
\LCPAddPHProcess[
  fluid=R1234yf,
  type=isenthalp,
  from={pressure=\RHeatPumpHighSaturationFromPressureSI Pa,
    temperature=40C},
  to={pressure=\RHeatPumpLowSaturationFromPressureSI Pa},
  enthalpy scale=1,
  pressure scale=1,
  name=r1234yf-da,
  export coordinates=RHeatPumpDA,
  color=orange,
  style={very thick,-Latex}
]
```

Finally, state A is known and the evaporator closes the cycle at B:

Evaporation and superheating A to B

```
\LCPAddPHProcess[
  fluid=R1234yf,
  type=isobar,
  pressure=\RHeatPumpLowSaturationFromPressureSI Pa,
  from={enthalpy=\RHeatPumpDAToEnthalpySI Jkg},
  to={temperature=0C},
  enthalpy scale=1,
  pressure scale=1,
  name=r1234yf-ab,
  export coordinates=RHeatPumpAB,
  color=orange,
  style={very thick,-Latex}
]
```

This chain demonstrates why process exports are numerical API values rather than presentation-only strings. Exported macros are expanded inside later `from` and `to` state specifications, allowing one thermodynamic calculation to feed the next without duplicating rounded numbers.

8.7 Step 4: place states by intersections, not copied coordinates

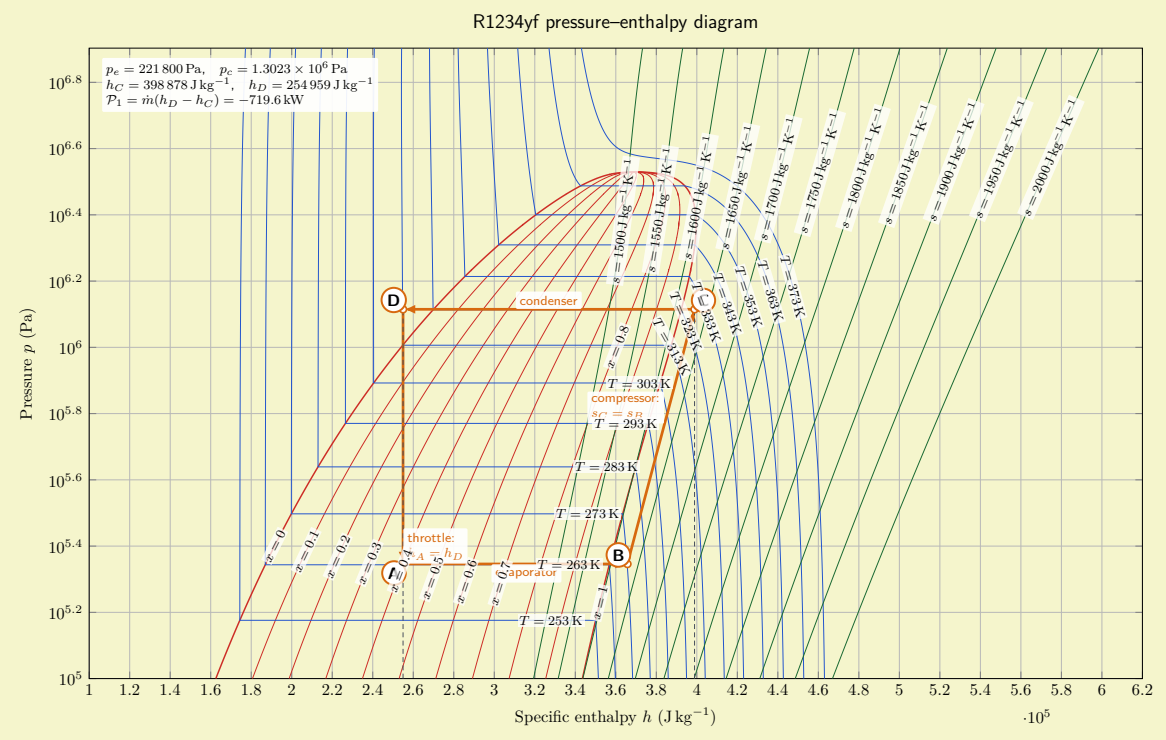
Each state is the common endpoint of two named physical paths. Ask TikZ to resolve the four intersections:

Four topology-derived state positions

```
\path[name intersections={of=r1234yf-da and r1234yf-ab,
  by=RHeatPumpA}];
\path[name intersections={of=r1234yf-ab and r1234yf-bc,
  by=RHeatPumpB}];
\path[name intersections={of=r1234yf-bc and r1234yf-cd,
  by=RHeatPumpC}];
\path[name intersections={of=r1234yf-cd and r1234yf-da,
  by=RHeatPumpD}];

\node at (RHeatPumpA) {$A$};
\node at (RHeatPumpB) {$B$};
\node at (RHeatPumpC) {$C$};
\node at (RHeatPumpD) {$D$};
```

The diagram source also projects C and D vertically toward the enthalpy axis. Those are the two readings needed for the requested condenser power.




```
\RHeatPumpSolutiontrue
\RHeatPumpDiagram
```

The four exact states generated by CoolProp are:

State	p/bar	T/°C	x	h/kJ kg^{-1}	s/kJ kg^{-1} K^{-1}
A	2.21785	−10.000	0.39960	254.959	1.21006
B	2.21785	0.000	—	365.744	1.63042
C	13.0235	55.318	—	398.878	1.63042
D	13.0235	40.000	—	254.959	1.18450

Quality is undefined for the single-phase superheated states B and C and the subcooled-liquid state D. State A is genuinely two-phase. Graphical answers should be rounded according to the printed grid; the table contains extra digits only to audit the generated construction.

8.8 Step 5: calculate and interpret the condenser power

The condenser process export provides both required SI enthalpies: `\RHeatPumpCDFromEnthalpySI` is h_C and `\RHeatPumpCDToEnthalpySI` is h_D. In LaTeX, the expandable floating-point evaluator can calculate with the full SI values without TeX's fixed-point dimension limit:

Power calculated from the plotted process

```
\edef\RHeatPumpCondenserPowerKW{\fpeval{
    5*(\RHeatPumpCDToEnthalpySI
    -\RHeatPumpCDFromEnthalpySI)/1000}}

\typeout{Condenser power received by R1234yf (kW):
  \RHeatPumpCondenserPowerKW}

$\mathcal P_1=
\qty{\RHeatPumpCondenserPowerKW}{\kilo\watt}$.
```

With $h_C = 398.878$ and $h_D = 254.959\,\mathrm{kJ\,kg^{-1}}$,

$$\mathcal{P}_1 = 5.0(254.959 - 398.878) = -719.6\,\mathrm{kW}.$$

The negative sign is physically necessary: the refrigerant does not receive heat in the condenser; it transfers approximately 720 kW to the pool-water side. If the requested quantity were instead the heating power received by the pool, and external losses were neglected, its value would be $+719.6\,\mathrm{kW}$.

The other two non-zero energy transfers provide an independent closure check:

$$\dot{Q}_{AB} = +553.9\,\mathrm{kW}, \qquad \dot{W}_{BC} = +165.7\,\mathrm{kW}, \qquad \dot{Q}_{CD} = -719.6\,\mathrm{kW}.$$

Their sum is zero to rounding, as required for a cyclic refrigerant with no net energy accumulation. This check is more informative than accepting the sign of one graphical subtraction in isolation.

8.9 Rebuild and adapt the teaching pair

Build the complete documentation set with the same external library used for the other examples:

Reproducible repository build

```
export LUACOOLPROP_LIB=/path/to/libCoolProp.dylib
./scripts/build-examples.sh
```

The two generated PDFs remain next to their drivers, while their transcripts are stored in `tutorial/logs`. The solution log records both saturation pressures, h_C, h_D, and the signed condenser power.

To adapt the construction to another heat pump:

1. replace the fluid and the two saturation temperatures;
2. distinguish superheating and subcooling from those saturation values;
3. derive operating pressures through CoolProp rather than a copied axis reading;
4. encode every device by its conserved property;
5. export each path before using its endpoint in the next path;
6. derive state markers from named-path intersections;
7. calculate powers from the exported SI enthalpies; and
8. state the sign convention and verify the cycle energy balance.

The worksheet, graphical correction, and numerical correction then remain three synchronized views of the same model.

9 Expert tutorial: methane liquefaction by the Linde process

This tutorial constructs a PH worksheet and a complete correction for an advanced methane-liquefaction exercise based on the Linde cycle. Its scope is the coupled thermodynamic process: chemistry, preliminary pure-substance questions, and methane-release modelling are outside the exercise.

LuaCoolProp regenerates every isoline and resolves every state against the external CoolProp shared library. Consequently, the generated worksheet remains useful with a different chart size, another reviewed CoolProp release, or deliberately changed operating data.

An advanced coupled-flow problem

This is not a single closed refrigeration cycle. It contains a fresh feed, a recirculated vapour stream, two streams exchanging heat in one regenerator, a phase separator, and a product stream. Several state enthalpies cannot be calculated until the separator mass balance and the regenerator energy balance have been closed. Drawing compressors first would silently assume a value of h_1 that the problem has not yet established.

9.1 Read the process as a directed network

The fresh methane flow rate is $\dot{m} = 1.0\,\mathrm{kg\,s^{-1}}$ at state 0, with $p_0 = 1\,\mathrm{bar}$ and $T_0 = 7\,^\circ\mathrm{C}$. The principal high-pressure stream passes through three ideal compressors and three isobaric coolers:

$$1 \xrightarrow{C_1} 2 \xrightarrow{E_1} 3 \xrightarrow{C_2} 4 \xrightarrow{E_2} 5 \xrightarrow{C_3} 6 \xrightarrow{E_3} 7.$$

The intermediate pressures and the two prescribed cooler outlet enthalpies are

$$p_2 = p_3 = 5\,\mathrm{bar}, \qquad p_4 = p_5 = 25\,\mathrm{bar}, \qquad h_3 = 866\,\mathrm{kJ\,kg^{-1}}, \qquad h_5 = 840\,\mathrm{kJ\,kg^{-1}}.$$

State 7 is fixed by $p_7 = 100\,\mathrm{bar}$ and $T_7 = -63\,^\circ\mathrm{C}$. The hot side of the regenerator cools this stream isobarically to $T_{7'} = -82\,^\circ\mathrm{C}$. The valve then produces state 8 at 1 bar with $h_8 = h_{7'}$.

The separator replaces state 8 by two saturated outlet streams at 1 bar: saturated vapour 9 is recycled, while saturated liquid 10 is the product. The recycled vapour is heated from 9 to $1'$ by the cold side of the regenerator and mixed with fresh state 0 to form state 1.

This gives the dependency order used in the implementation:

$$\begin{aligned}(0, 7, 7', 9, 10) &\longrightarrow 8 \longrightarrow x_8 \longrightarrow (\dot{m}_1, \dot{m}_{1'}) \\ &\longrightarrow h_{1'} \longrightarrow h_1 \longrightarrow (2, 4, 6) \longrightarrow (\dot{Q}_{E1}, \dot{Q}_{E2}, \dot{Q}_{E3}, \dot{W}_{C1}).\end{aligned}$$

9.2 Use a shared worksheet and progressive solution

The generated teaching pair consists of three files:

File	Responsibility
`tutorial/methane-linde-worksheet.tex`	One-page blank PH worksheet.
`tutorial/methane-linde-solution.tex`	Six-page progressive correction.
`tutorial/source/methane-linde-diagrams.tex`	Shared isolines, state solver, balances, process paths, annotations, and log output.

The two drivers change only one Boolean. This prevents a later modification of the pressure or enthalpy domain from desynchronizing the student and teacher documents.

Thin worksheet and solution drivers

```
% Student worksheet
\newif\ifMethaneLindeSolution
\MethaneLindeSolutionfalse

% Progressive model solution
\newif\ifMethaneLindeSolution
\MethaneLindeSolutiontrue
```

9.3 Step 1: regenerate the blank methane PH chart

The worksheet covers approximately $-100 \le h \le 1200\,\mathrm{kJ\,kg^{-1}}$ and $0.2\,\mathrm{bar} \le p \le 200\,\mathrm{bar}$. A logarithmic pressure axis is essential: the one-bar separator and the hundred-bar regenerator must be readable on the same sheet.

Axis used by the regenerated worksheet

```
\begin{axis}[
  lcp fluid=Methane,
  width=24cm,
  height=15.8cm,
  xmin=-100,
  xmax=1200,
  ymin=.2,
  ymax=200,
  ymode=log,
  xlabel={Specific enthalpy $h$
    (\si{\kilo\joule\per\kilogram})},
  ylabel={Pressure $p$ (bar)},
  grid=both,
  auto node placement,
  auto node algorithm=repair,
  auto node candidates=19,
  auto node bbox mode=oriented
]
  % Four independently calculated families follow.
\end{axis}
```

Four families provide the property information needed for the exercise:

1. qualities $x = 0, 0.1, \ldots, 1$ define the two-phase region;
2. isotherms from $103\,\mathrm{K}$ to $373\,\mathrm{K}$ locate the directly specified temperatures and support interpolation;
3. isentropes from $1\,\mathrm{kJ\,kg^{-1}\,K^{-1}}$ to $6.5\,\mathrm{kJ\,kg^{-1}\,K^{-1}}$ construct the ideal compressor outlets; and
4. isochores from $0.005\,\mathrm{m^3\,kg^{-1}}$ to $5\,\mathrm{m^3\,kg^{-1}}$ provide a useful secondary property grid.

Quality and temperature families

```
\LCPAddPHQuality[
  fluid=Methane,
  pressure min=20000,
  pressure max=20000000,
  quality values={0,0.1,0.2,0.3,0.4,0.5,
    0.6,0.7,0.8,0.9,1},
  quality symbol=x,
  labels=true
]

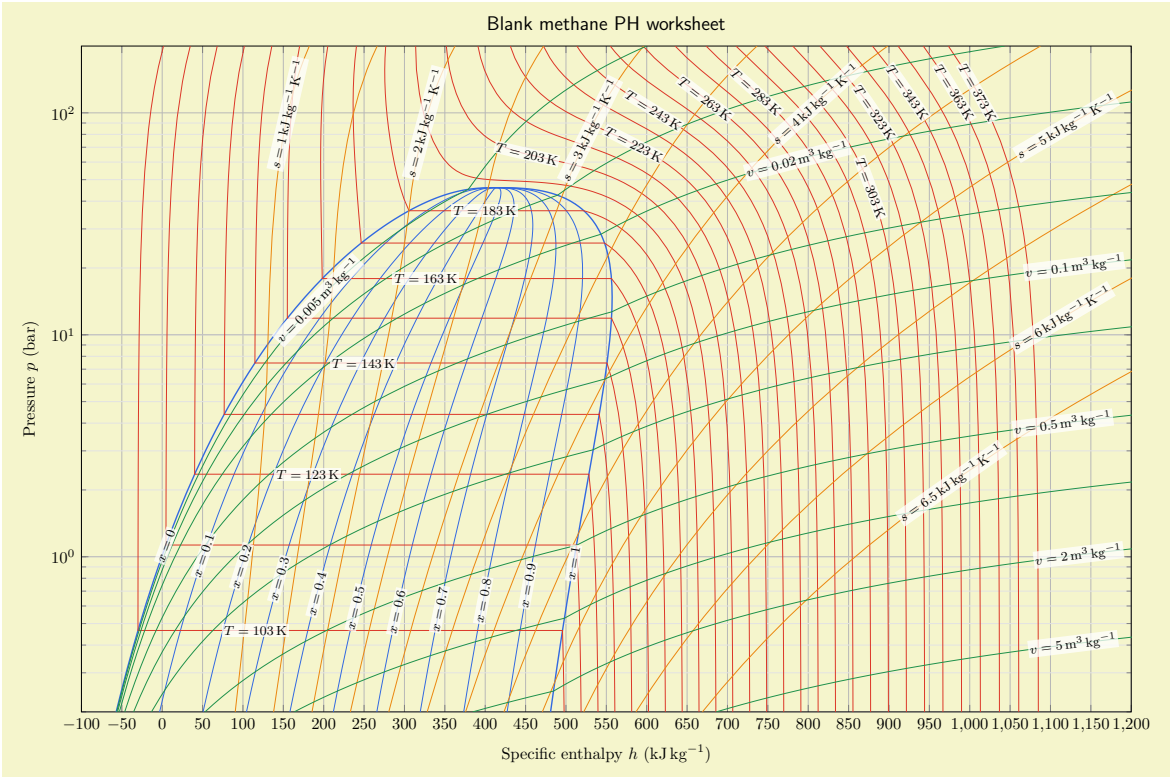
\LCPAddPHIsotherms[
  fluid=Methane,
  pressure min=20000,
  pressure max=20000000,
  temperature unit=K,
  temperature values={103,113,123,133,143,153,
    163,173,183,193,203,213,223,233,243,253,
    263,273,283,293,303,313,323,333,343,353,
    363,373},
  temperature symbol=T,
  isotherm label every=2,
  labels=true
]
```

Entropy and specific-volume families

```
\LCPAddPHIsentropes[
  fluid=Methane,
  pressure min=20000,
  pressure max=20000000,
  entropy unit=kjkgk,
  entropy values={1,1.5,2,2.5,3,3.5,4,4.5,
    5,5.5,6,6.5},
  entropy symbol=s,
  isentrope label every=2,
  labels=true
]

\LCPAddPHIsochores[
  fluid=Methane,
  pressure min=20000,
  pressure max=20000000,
  specific volume unit=m3kg,
  specific volume values={0.005,0.01,0.02,0.05,
    0.1,0.2,0.5,1,2,5},
  specific volume symbol=v,
  isochore label every=2,
  labels=true
]
```

Alternate members of the isotherm, isentrope, and isochore families are labelled, but every requested curve remains in the plot. The distinction matters: label density is a typographic choice, whereas curve density determines graphical interpolation accuracy. Label placement is handled exclusively by `pgfplots-autonode`.




```
\MethaneLindeSolutionfalse
\MethaneLindeDiagram{0}
```

9.4 Step 2: calculate the primitive states

The first calculations use only data explicitly supplied by the exercise. Short invisible paths export complete endpoint states without introducing hand-written chart coordinates.

Feed and one-bar saturation endpoints

```
\LCPAddPHProcess[
  fluid=Methane,
  type=isobar,
  pressure=1bar,
  from={temperature=7C},
  to={quality=1},
  style={draw=none},
  export coordinates=MethaneLindeFeedVapour
]

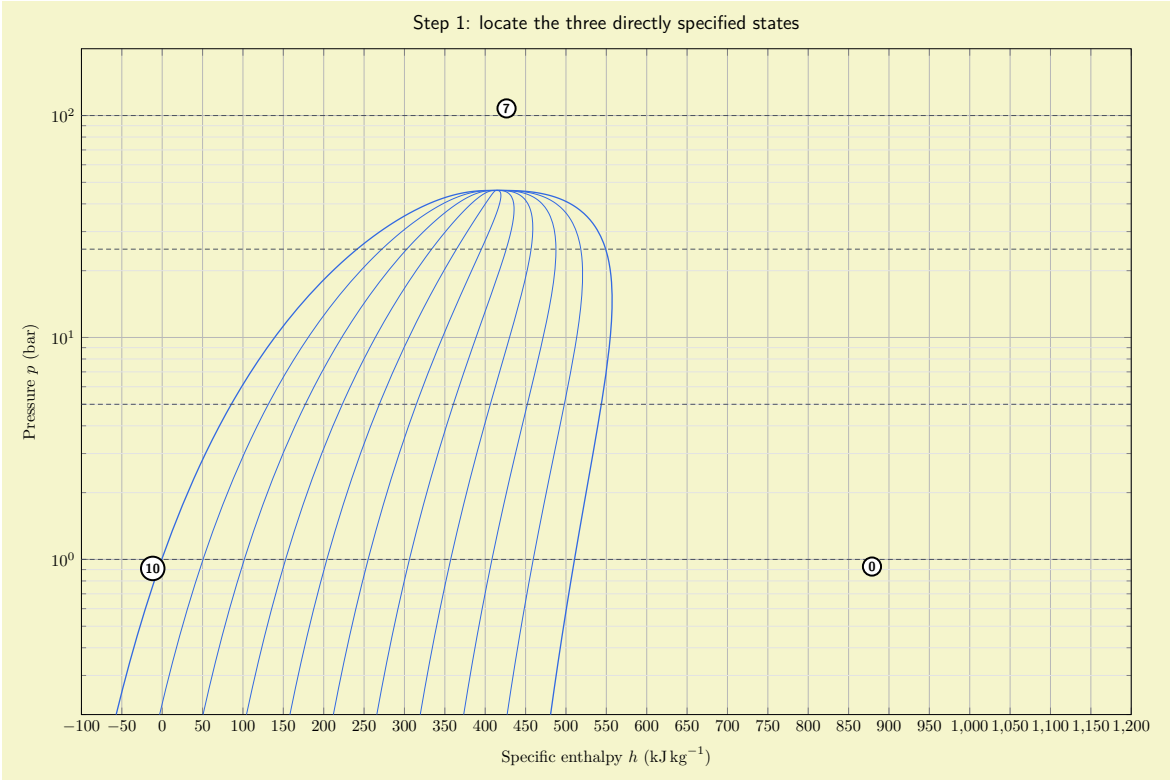
\LCPAddPHProcess[
  fluid=Methane,
  type=isobar,
  pressure=1bar,
  from={quality=0},
  to={quality=1},
  style={draw=none},
  export coordinates=MethaneLindeSaturation
]
```

This resolves states 0, 9, and 10. The high-pressure side supplies states 7 and 7′:

High-pressure regenerator endpoints

```
\LCPAddPHProcess[
  fluid=Methane,
  type=isobar,
  pressure=100bar,
  from={temperature=-63C},
  to={temperature=-82C},
  style={draw=none},
  export coordinates=MethaneLindeHighRegenerator
]
```

The first page of the progressive correction shows only the one-, five-, twenty-five-, and hundred-bar construction levels and the directly specified states 0, 7, and 10.




```
\MethaneLindeSolutiontrue
\MethaneLindeDiagram{1}
```

9.5 Step 3: throttle and separate the two-phase outlet

The insulated valve has no moving part. The steady-flow first law therefore gives

$$h_8 = h_{7'}.$$

The complete state 8, including its quality, is obtained from pressure and enthalpy:

Joule–Thomson throttle from 100 bar to 1 bar

```
\LCPAddPHProcess[
  fluid=Methane,
  type=isenthalp,
  from={pressure=100bar,temperature=-82C},
  to={pressure=1bar},
  export coordinates=MethaneLindeThrottle,
  log coordinates=true
]

\edef\MethaneLindeQualityEight{\fpeval{
    \MethaneLindeThrottleToQuality}}
```

CoolProp returns

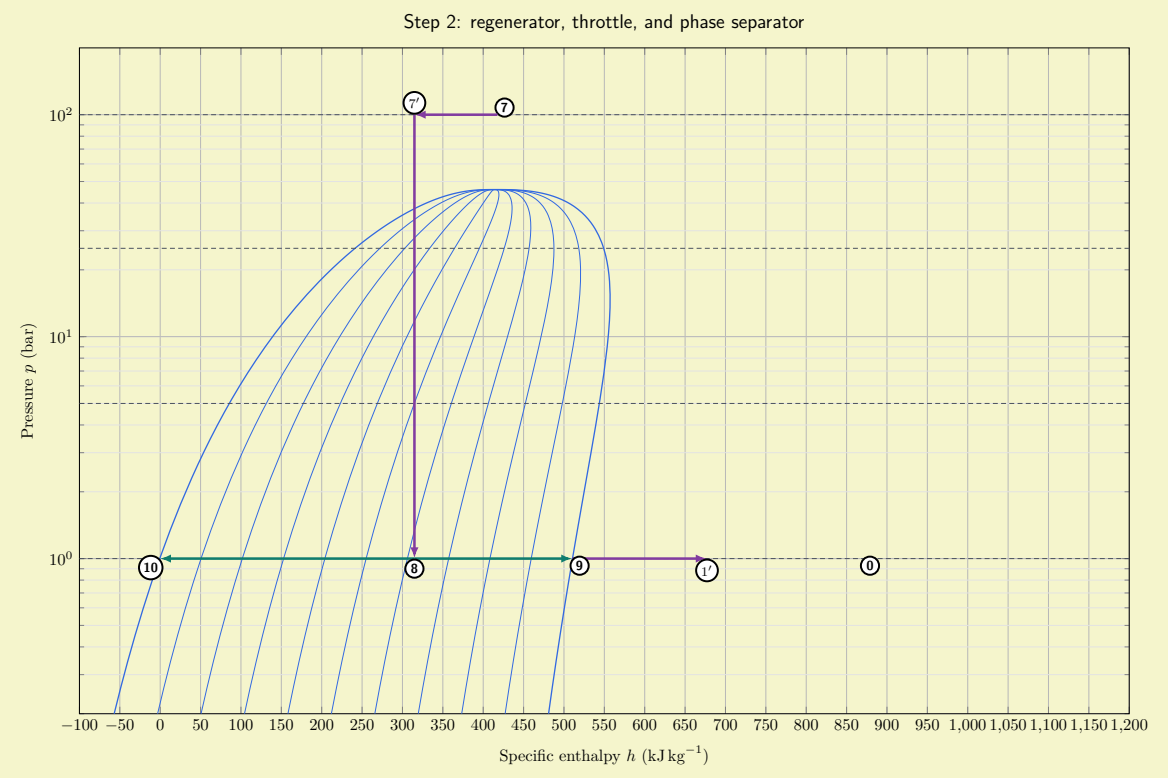
$$h_8 = h_{7'} = 314.835\,\mathrm{kJ\,kg^{-1}}, \qquad x_8 = 0.617062.$$

The separator is isobaric. It does not follow a reversible thermodynamic path between 8, 9, and 10; it routes the two phase fractions into two outlet streams. The two horizontal arrows are therefore a flow-topology representation, not equilibrium trajectories undergone by individual fluid particles.

Represent both separator outlet streams

```
\LCPAddPHProcess[
  fluid=Methane,
  type=isobar,
  pressure=1bar,
  from={enthalpy=\MethaneLindeThrottleToEnthalpySI Jkg},
  to={quality=1},
  name=ml-eight-nine
]

\LCPAddPHProcess[
  fluid=Methane,
  type=isobar,
  pressure=1bar,
  from={enthalpy=\MethaneLindeThrottleToEnthalpySI Jkg},
  to={quality=0},
  name=ml-eight-ten
]
```




```
\MethaneLindeSolutiontrue
\MethaneLindeDiagram{2}
```

9.6 Step 4: close the separator mass balance

Let $\dot{m}_1$ be the flow rate through the high-pressure train and state 8, and let $\dot{m}_{1'}$ be the recycled saturated-vapour flow rate. By the definition of vapour quality,

$$\dot{m}_{1'} = x_8 \dot{m}_1.$$

At steady state, the saturated-liquid product flow equals the fresh feed flow:

$$\dot{m} = (1 - x_8)\dot{m}_1.$$

Hence

$$\dot{m}_1 = \frac{\dot{m}}{1 - x_8}, \qquad \dot{m}_{1'} = \frac{x_8 \dot{m}}{1 - x_8}.$$

Mass-flow rates calculated from the exported quality

```
\edef\MethaneLindeMassFlowOne{\fpeval{
    1/(1-\MethaneLindeQualityEight)}}
\edef\MethaneLindeMassFlowOneBis{\fpeval{
    \MethaneLindeQualityEight
    *\MethaneLindeMassFlowOne}}
```

For the current library, the automatically generated values are

$$\dot{m}_1 = 2.611\,39\,\mathrm{kg\,s^{-1}}, \qquad \dot{m}_{1'} = 1.611\,39\,\mathrm{kg\,s^{-1}}.$$

Both equalities $\dot{m}_1 = \dot{m} + \dot{m}_{1'}$ and $\dot{m}_{1'} = x_8 \dot{m}_1$ should be checked before proceeding.

9.7 Step 5: close the two-stream regenerator balance

The regenerator is globally insulated and has no moving part. The two streams have different flow rates, so equating their specific-enthalpy changes would be wrong. Its steady-flow balance is

$$\dot{m}_1(h_{7'} - h_7) + \dot{m}_{1'}(h_{1'} - h_9) = 0.$$

Solving for the only unknown gives

$$h_{1'} = h_9 - \frac{\dot{m}_1}{\dot{m}_{1'}}(h_{7'} - h_7).$$

Cold-stream outlet enthalpy from the regenerator balance

```
\edef\MethaneLindeEnthalpyOneBisSI{\fpeval{
    \MethaneLindeSaturationToEnthalpySI
    -(\MethaneLindeMassFlowOne
    /\MethaneLindeMassFlowOneBis)
    *(\MethaneLindeHighRegeneratorToEnthalpySI
    -\MethaneLindeHighRegeneratorFromEnthalpySI)}}
```

The live result is $h_{1'} = 677.122\,\mathrm{kJ\,kg^{-1}}$. It lies to the right of state 9, as required for the cold stream that receives heat, while $7'$ lies to the left of state 7 because the high-pressure stream is cooled.

9.8 Step 6: close the mixer balance and reveal state 1

The adiabatic one-bar mixer has no moving part. Mass and enthalpy-flow conservation give

$$\dot{m}_1 = \dot{m} + \dot{m}_{1'}, \qquad \dot{m}_1 h_1 = \dot{m} h_0 + \dot{m}_{1'} h_{1'}.$$

Therefore

$$h_1 = \frac{\dot{m} h_0 + \dot{m}_{1'} h_{1'}}{\dot{m}_1}.$$

Mixer outlet enthalpy

```
\edef\MethaneLindeEnthalpyOneSI{\fpeval{
    (\MethaneLindeFeedVapourFromEnthalpySI
    +\MethaneLindeMassFlowOneBis
    *\MethaneLindeEnthalpyOneBisSI)
    /\MethaneLindeMassFlowOne}}
```

LuaCoolProp obtains $h_1 = 751.038\,\mathrm{kJ\,kg^{-1}}$ and, by querying the complete state at 1 bar, $T_1 = -48.371\,^\circ\mathrm{C}$. This is why state 1 must not be placed at the fresh-feed temperature. The recycle stream changes the compressor inlet substantially.

9.9 Step 7: construct the first compression and cooler

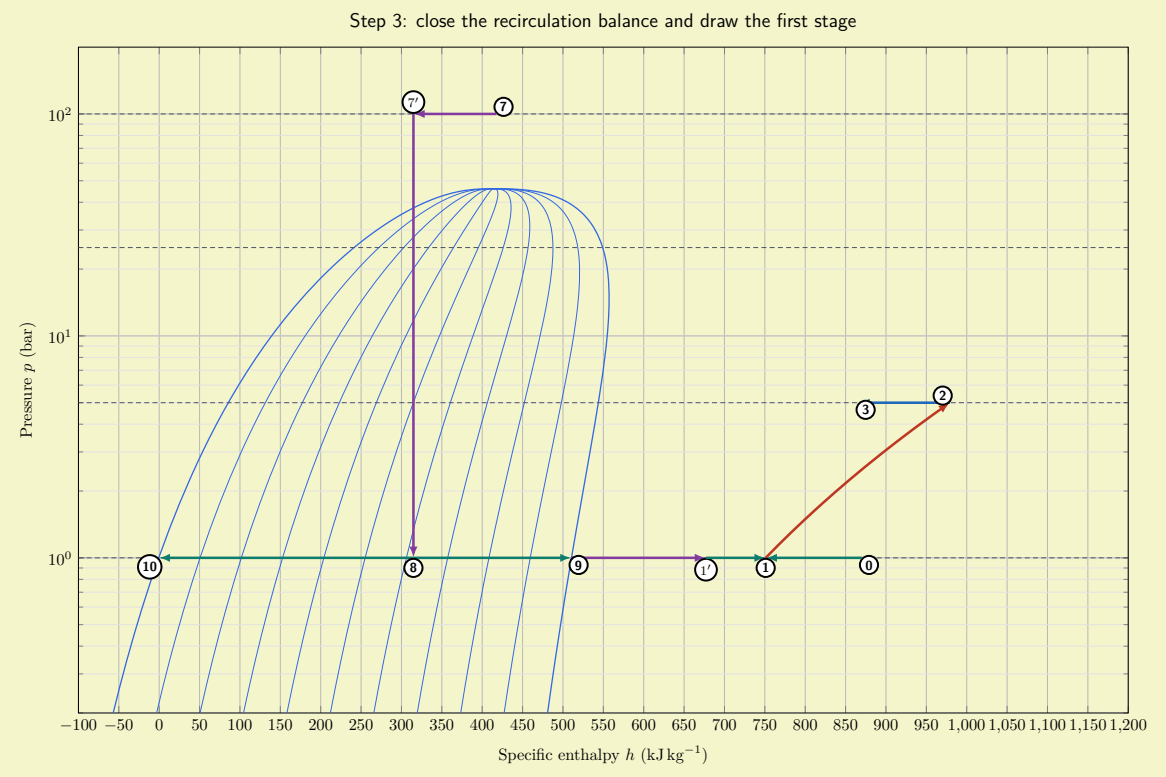
Only now is state 1 fully known. Compressor C_1 is adiabatic and reversible, so $s_2 = s_1$ and $p_2 = 5\,\mathrm{bar}$.

First compressor and first isobaric cooler

```
\LCPAddPHProcess[
  fluid=Methane,
  type=isentrope,
  from={pressure=1bar,
    enthalpy=\MethaneLindeEnthalpyOneSI Jkg},
  to={pressure=5bar},
  export coordinates=MethaneLindeCOne,
  log coordinates=true
]

\LCPAddPHProcess[
  fluid=Methane,
  type=isobar,
  pressure=5bar,
  from={entropy=\MethaneLindeCOneToEntropySI JkgK},
  to={enthalpy=866kJkg},
  export coordinates=MethaneLindeEOne
]
```

The exact process export gives $h_2 = 979.046\,\mathrm{kJ\,kg^{-1}}$ and $T_2 = 56.861\,^{\circ}\mathrm{C}$. State 3 is not inferred from a temperature curve: its enthalpy is explicit in the statement.




```
\MethaneLindeSolutiontrue
\MethaneLindeDiagram{3}
```

9.10 Step 8: repeat the invariant pattern for stages 2 and 3

The second stage repeats “isentropic compression, then isobaric cooling” at the next pressure level:

Second compression stage

```
\LCPAddPHProcess[
  fluid=Methane,
  type=isentrope,
  from={pressure=5bar,enthalpy=866kJkg},
  to={pressure=25bar},
  export coordinates=MethaneLindeCTwo,
  log coordinates=true
]

\LCPAddPHProcess[
  fluid=Methane,
  type=isobar,
  pressure=25bar,
  from={entropy=\MethaneLindeCTwoToEntropySI JkgK},
  to={enthalpy=840kJkg},
  export coordinates=MethaneLindeETwo
]
```

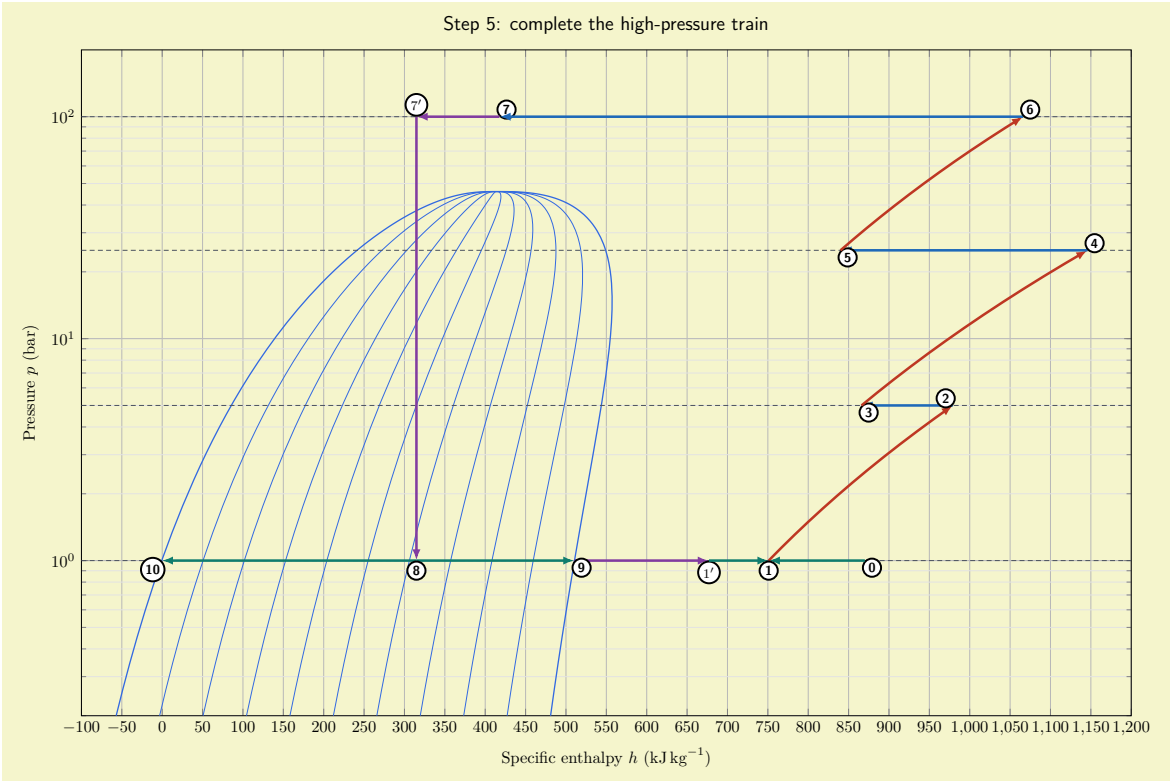
This yields $h_4 = 1145.817\,\mathrm{kJ\,kg^{-1}}$ and $T_4 = 129.797\,^\circ\mathrm{C}$. The third stage ends at the given state 7:

Third compression stage

```
\LCPAddPHProcess[
  fluid=Methane,
  type=isentrope,
  from={pressure=25bar,enthalpy=840kJkg},
  to={pressure=100bar},
  export coordinates=MethaneLindeCThree,
  log coordinates=true
]

\LCPAddPHProcess[
  fluid=Methane,
  type=isobar,
  pressure=100bar,
  from={entropy=\MethaneLindeCThreeToEntropySI JkgK},
  to={temperature=-63C},
  export coordinates=MethaneLindeEThree
]
```

The third compressor outlet is $h_6 = 1066.089\,\mathrm{kJ\,kg^{-1}}$ at $T_6 = 115.236\,^\circ\mathrm{C}$.




```
\MethaneLindeSolutiontrue
\MethaneLindeDiagram{5}
```

9.11 Step 9: calculate powers with a declared sign convention

Heat and useful work are counted positive when received by the methane. With negligible kinetic- and potential-energy changes, the steady-flow first law is

$$\dot{Q} + \dot{W}_u = \dot{m}(h_{\text{out}} - h_{\text{in}}).$$

The coolers have no moving part, hence

$$\begin{aligned}\dot{Q}_{E1} &= \dot{m}_1(h_3 - h_2),\\ \dot{Q}_{E2} &= \dot{m}_1(h_5 - h_4),\\ \dot{Q}_{E3} &= \dot{m}_1(h_7 - h_6).\end{aligned}$$

All three must be negative. Their automated values are

$$\dot{Q}_{E1} = -295.2\,\text{kW}, \qquad \dot{Q}_{E2} = -798.6\,\text{kW}, \qquad \dot{Q}_{E3} = -1693.4\,\text{kW}.$$

For the first ideal compressor,

$$\dot{W}_{C1} = \dot{m}_1(h_2 - h_1) = 595.4\,\text{kW}.$$

The three idealized compressor paths imply $\dot{W}_{C1} + \dot{W}_{C2} + \dot{W}_{C3} = 1916.5\,\text{kW}$. The exercise independently specifies a total motor power of 1.8 MW; it is that prescribed value, rather than the sum of graphically inferred ideal stage powers, that belongs in the requested cooling effectiveness:

$$e_T = -\frac{\dot{Q}_{E1} + \dot{Q}_{E2} + \dot{Q}_{E3}}{\dot{W}_{\text{motor}}} = 1.5485.$$

Power and effectiveness calculations in LaTeX

```
\edef\MethaneLindeThermalPowerOneKW{\fpeval{
    \MethaneLindeMassFlowOne
    *(866000-\MethaneLindeCOneToEnthalpySI)/1000}}

\edef\MethaneLindeCompressorOnePowerKW{\fpeval{
    \MethaneLindeMassFlowOne
    *(\MethaneLindeCOneToEnthalpySI
    -\MethaneLindeEnthalpyOneSI)/1000}}

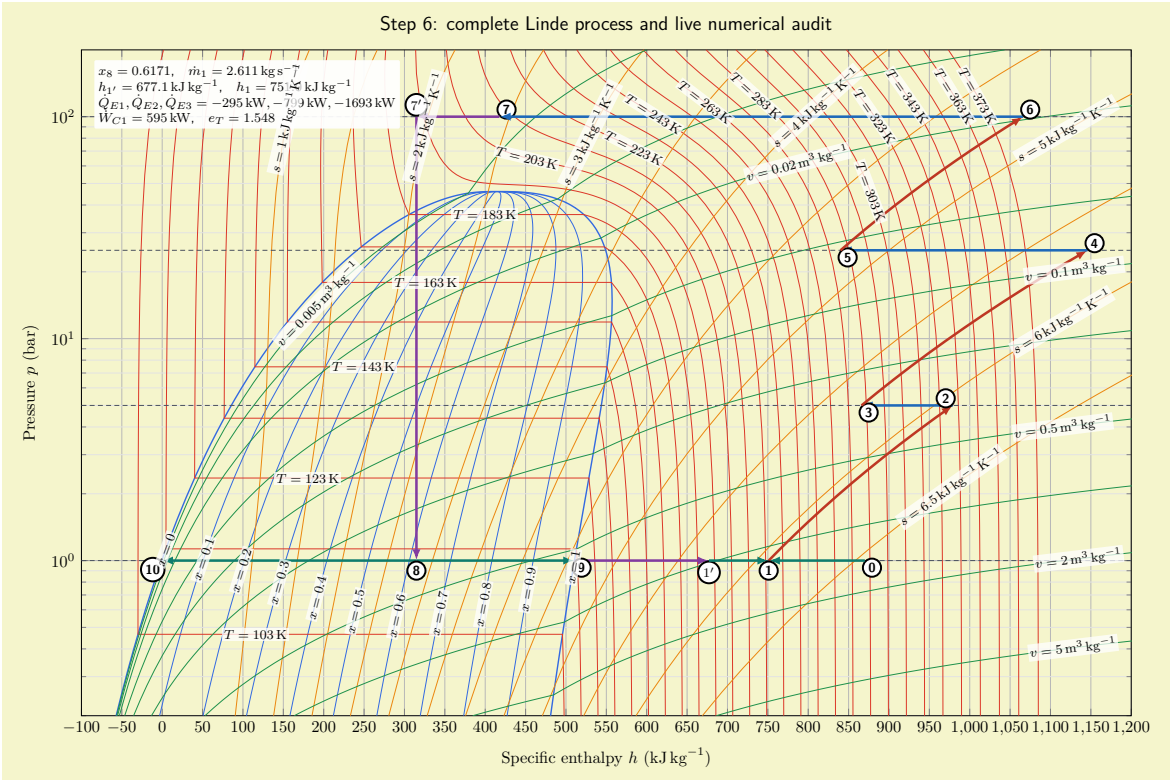
\edef\MethaneLindeCoolingEffectiveness{\fpeval{
    -(\MethaneLindeThermalPowerOneKW
    +\MethaneLindeThermalPowerTwoKW
    +\MethaneLindeThermalPowerThreeKW)/1800}}
```

9.12 Step 10: read the complete process without inventing a cycle

The final page overlays the entire network on the dense regenerated chart. The colour semantics are intentionally structural:

- red paths are the three isentropic compressions;
- blue paths are the three isobaric coolers;
- purple paths are the two sides of the regenerator and the throttle;
- teal paths represent mixing and phase separation.

The graphic is not a single polygon. At state 8 one incoming stream becomes states 9 and 10; at state 1, states 0 and 1′ merge. Drawing an artificial arrow from 10 to 0 would incorrectly suggest that the liquid product is reheated and recycled. State 10 leaves the process, while state 0 is new feed.



```
\MethaneLindeSolutiontrue
\MethaneLindeDiagram{6}
```

The calculated state table is:

State	p/bar	T/°C	h/kJ kg^{-1}	x
0	1	7.000	870.145	—
1	1	−48.371	751.038	—
1′	1	−83.412	677.122	—
2	5	56.861	979.046	—
3	5	7.105	866.000	—
4	25	129.797	1145.817	—
5	25	5.609	840.000	—
6	100	115.236	1066.089	—
7	100	−63.000	417.612	—
7′	100	−82.000	314.835	—
8	1	−161.642	314.835	0.61706
9	1	−161.642	510.562	1
10	1	−161.642	−0.557	0

Values shown as "—" are single-phase states, for which vapour quality is not defined.

9.13 Graphical estimates versus live calculations

The approximate column uses values read from a printed chart, with an uncertainty of approximately $\pm 20\,\mathrm{kJ\,kg^{-1}}$. The current calculation evaluates the exact state model exposed by the installed CoolProp library. The two sets must not be mixed silently.

Quantity	Approximate chart reading	Current live calculation
h_0/kJ kg^{-1}	860	870.145
h_7/kJ kg^{-1}	410	417.612
$h_{7'} = h_8$ /kJ kg^{-1}	300	314.835
h_9/kJ kg^{-1}	440	510.562
x_8	0.60	0.61706
$\dot{m}_1$/kg s^{-1}	2.5	2.61139
$h_{1'}$/kJ kg^{-1}	620	677.122
h_1/kJ kg^{-1}	720	751.038
h_2/kJ kg^{-1}	960	979.046
h_4/kJ kg^{-1}	1110	1145.817
h_6/kJ kg^{-1}	1040	1066.089
e_T	1.4	1.5485

This difference is pedagogically useful. A graphical answer should be reported with graphical precision and compared with the computed state. An automated answer should report its fluid model, CoolProp version, exact inputs, and sign convention. The solution log provides the live values needed for that audit.

9.14 Numerical API and reproducible build

Every calculated process uses `export coordinates`. Among the reusable results are the endpoint enthalpy, temperature, entropy, quality, pressure, specific volume, and plotted coordinates. The implementation deliberately feeds those full-precision SI macros into later processes; rounded numbers are used only for display.

For example, these two definitions are fundamentally different:

Keep calculation data separate from presentation

```
% Full-precision value used as a later state input:
from={entropy=\MethaneLindeCOneToEntropySI JkgK}

% Rounded presentation of the same calculation:
\qty[round-mode=places,round-precision=1]
  {979.045669801632}{\kilo\joule\per\kilogram}
```

Build the worksheet, progressive solution, all examples, and the complete manual through the repository workflow:

Rebuild the complete documentation set

```
export LUACOOLPROP_LIB=/path/to/libCoolProp.dylib
./scripts/build-examples.sh
```

The generated PDFs are installed beside their drivers. Console transcripts are retained under `tutorial/logs`; the methane solution log records x_8, both recirculation flow rates, $h_{1'}$, h_1, the three cooling powers, the first compressor power, the ideal three-compressor total, and e_T.

When adapting the example, preserve the solution order:

1. resolve directly specified and saturated states;
2. apply the isenthalpic valve constraint;
3. calculate quality and close the separator mass balance;
4. close the two-stream regenerator energy balance;
5. close the mixer mass and energy balances;
6. construct each isentropic compressor and isobaric cooler;
7. calculate powers from unrounded exported enthalpies; and
8. audit signs, flow conservation, and the difference between specified motor power and idealized diagram readings.

That order turns an apparently circular network into a deterministic, reproducible calculation.

Part III
User manual

10 Purpose and scope

`luacoolprop` connects LuaTeX to the CoolProp shared library through LuaTeX FFI. Thermodynamic states and isolines are computed in Lua; graphical output is emitted as PGFPlots code. The package provides PH (pressure–enthalpy), PV (pressure–specific-volume), TS (temperature–entropy), HS (enthalpy–entropy), and PT (pressure–temperature) diagrams for pure fluids. PH uses a linear enthalpy axis and logarithmic pressure axis; PV uses logarithmic axes for both specific volume and pressure; TS uses linear mass-specific entropy and absolute temperature axes; HS uses linear mass-specific entropy and enthalpy axes; PT uses temperature and logarithmic pressure axes.

The package supports quality curves, isotherms, isentropes, isochores, TS isenthalps, HS isobars, thermodynamic process paths, named curves, per-curve overrides, and automatic label placement delegated to `pgfplots-autonode`.

This package was vibe-coded with OpenAI Codex and the Astra-6 model under the direction of Christophe Jorssen.

Independent validation

Publication on CTAN is not a correctness or safety certification. Inspect the source and independently validate thermodynamic results before relying on them, especially for industrial, regulated, or safety-critical work.

Copyright © 2026 Christophe Jorssen. LuaCoolProp is distributed under the LaTeX Project Public License 1.3c or later. The maintenance status is *maintained*, and the Current Maintainer is Christophe Jorssen (christophe.jorssen@gmail.com). CoolProp is an external, MIT-licensed dependency and is not included in this work.

10.1 How to cite CoolProp

CoolProp asks users to cite its published reference and the project website when reporting work based on its calculations. The citation is:

> Ian H. Bell, Jorrit Wronski, Sylvain Quoilin, and Vincent Lemort, "Pure and Pseudo-pure Fluid Thermophysical Property Evaluation and the Open-Source Thermophysical Property Library CoolProp," *Industrial & Engineering Chemistry Research* **53**(6), 2498–2508 (2014). DOI: 10.1021/ie4033999. Project website: https://coolprop.org/.

The project's citation page provides a BibTeX entry. Cite the CoolProp version used for reproducibility as well as this article; LuaCoolProp is an independent TeX interface, not an official CoolProp component.

Design principle

LuaCoolProp deliberately separates thermodynamics from label placement. Thermodynamic states, isolines, and process paths are computed in Lua; automatic label placement is delegated to `pgfplots-autonode`. This keeps the package compatible with PGFPlots' survey/render lifecycle and avoids duplicating axis geometry in Lua.

11 Installation and runtime requirements

A working document needs LuaTeX or LuaHBTeX with FFI enabled, PGF/TikZ, PGFPlots 1.18 or later, the independent `pgfplots-autonode` package, and a CoolProp shared library. Both `pgfplots-autonode` and CoolProp are external runtime dependencies; neither is embedded in the LuaCoolProp distribution.

LuaCoolProp does not set PGFPlots' `compat` key. Select the compatibility level required by the complete document in its preamble. Automatic label hooks are local to the axes created by `\LCPDiagram`; a manually constructed axis using low-level family commands must add `auto node placement` when automatic labels are requested.

11.1 Install the TeX files

A TeX distribution can install the package through its normal package manager: use `tlmgr install luacoolprop pgfplots-autonode` with TeX Live, or select both packages in the MiKTeX Console. A distribution normally installs the declared TeX dependency automatically. CoolProp remains a separate native dependency in both cases.

For a manual installation, unpack both `luacoolprop.tds.zip` and `pgfplots-autonode.tds.zip` at the root of a personal TEXMF tree. TeX Live normally reports that directory with

`kpsewhich -var-value=TEXMFHOME`. Refresh its filename database with `mktexlsr` if the local installation uses one. The browsing archive `luacoolprop.zip` is intended for inspection and must not be substituted for the TDS archive.

The shared library is typically named `libCoolProp.so` on Linux, `libCoolProp.dylib` on macOS, and `CoolProp.dll` on Windows. LuaCoolProp does not vendor CoolProp, and installing the CTAN package does not install that library. Obtain it separately from the official CoolProp project:

- source repository: https://github.com/CoolProp/CoolProp;
- shared-library instructions and release downloads: https://coolprop.org/coolprop/wrappers/SharedLibrary/.

11.2 Security boundary and `-shell-escape`

Compile trusted TeX sources only

The required `-shell-escape` option is dangerous. It permits a TeX document, and every class, package, module, or file that it loads, to run external programs with the permissions of the current user. A malicious TeX source can therefore read, modify, or disclose user-accessible data. Compile only sources and dependencies that you trust, preferably in an isolated build directory or sandbox, without elevated privileges or unnecessary secrets. Review downloaded examples before compiling them. Never run the TeX engine as an administrator or as `root`.

LuaCoolProp requires full shell escape because the LuaTeX distribution exposes its Lua FFI module only in a shell-escape-enabled typesetting process. That FFI module loads `libCoolProp` as native executable code and calls its C API. LuaCoolProp itself does not invoke a compiler, download CoolProp, or issue shell commands while typesetting; nevertheless, enabling the option enlarges the capabilities of the *entire* document. Restricted shell escape is not sufficient for this FFI mechanism.

The shared library is another trust boundary: load only a CoolProp binary that you built from a reviewed revision or obtained from an official release. Use an absolute `LUACOOLPROP_LIB` path so an unrelated library with the same name cannot be selected earlier in a platform search path.

11.3 Choose a compatible CoolProp binary

The TeX engine process and the CoolProp library must use the same architecture: normally `arm64` or `x86_64` on macOS, `x86_64` on 64-bit Linux, and `x64` on 64-bit Windows. An architecture mismatch usually appears as a dynamic-loader error before the first diagram is computed.

There are two supported approaches:

1. download a precompiled library for the required operating system and architecture from CoolProp's official shared-library page; or
2. build a pinned CoolProp tag or commit from source, as described below.

LuaCoolProp automatically identifies the binary actually loaded and selects its reviewed interface. There is no native-version `compat` key. The supported version/commit pairs are:

8.0.0b1 `322000f4ddab06fcdf94488b7bfd782f01a0783d`;

8.0.0 `ae81610e7d23efc57f9d051c8e70a4d66e87537f`;

8.0.1dev `6de0bf8c5164e415814bda3d7313e81d201fab07`.

The first commit defines the lower bound called “8.0-dev”, which is not an upstream tag. The last is an additional development snapshot; other commits with the same version string are not implicitly supported. The two upstream tags `v8.0.0` and `gui-v8.0.0` identify the same native revision. All three use the adapter `coolprop-8-core-v1`; their consumed C signatures agree, and configuration capabilities are selected from the exact identity.

Validation covers macOS arm64 with LuaTeX/texlua, LuaLaTeX and ConTeXt MkIV. The same package runtime is tested against all three binaries, including the five projections, critical tails, domain gaps, process endpoints, reference conventions, caches and exported tables. Linux LP64 and Windows x64 LLP64 loading paths are implemented but have not been tested for this version. An unknown identity, missing symbol or unsupported data model is an error before property calculations. Headers and a C compiler are not needed during normal use. Detection does not update the library and is not a binary authenticity check.

The baseline `8.0.0b1` has an upstream limitation: a CO_2 gas query at 300 K and 1 bar fails in its melting-line check below the triple pressure. The interface preserves that failure; it does not invent a value or impose a phase. Use the reviewed `8.0.0` release for such calculations. Version comparisons use a common valid state (300 K, 10 bar for CO_2), and record this additional limitation separately.

For reproducible documents, record the exact CoolProp release or commit beside the document sources. Do not copy the CoolProp checkout, build tree, or binary into the LuaCoolProp package tree or a CTAN archive.

Building requires Git, CMake, Ninja, Python, a C++ compiler, and the tools used by CoolProp’s submodules. Clone outside the LuaCoolProp tree and initialize all submodules. The following example pins CoolProp `v8.0.0`; replace that tag deliberately when adopting another reviewed release.

Common source checkout on macOS or Linux

```
git clone https://github.com/CoolProp/CoolProp.git ../coolprop-src
git -C ../coolprop-src fetch -tags
git -C ../coolprop-src checkout -detach v8.0.0
git -C ../coolprop-src submodule update -init -recursive
```

If `../coolprop-src` already exists, verify that it is the intended Git checkout, fetch the requested revision, check it out, and repeat the recursive submodule update. A detached checkout is intentional here: it makes the exact dependency visible and prevents an accidental branch update.

11.4 macOS: download or compile the `.dylib`

Install Apple’s command-line developer tools and make Git, CMake, and Ninja available. For a reproducible build, first perform the pinned common checkout above. From a LuaCoolProp source or CTAN tree, the supplied script then reuses that checkout, initializes its submodules, and performs the CMake and Ninja build in sibling directories. If the checkout does not yet exist, the script clones the reviewed `v8.0.0` tag:

Native macOS build

```
./scripts/build-coolprop-macos.sh
```

On Apple Silicon the default is `arm64`, including when the script is started from a translated shell. Override it when the TeX engine has a different architecture, or request a universal library explicitly:

macOS architecture overrides

```
COOLPROP_MACOS_ARCH=x86_64 ./scripts/build-coolprop-macos.sh
COOLPROP_MACOS_ARCH='arm64;x86_64' ./scripts/build-coolprop-macos.sh
```

The script prints the absolute path to `libCoolProp.dylib` as its final line. It deliberately leaves both CoolProp source and build products outside the LuaCoolProp directory. The helper builds the external shared library; it does not modify the package's Lua interface. Keep the selected CoolProp revision with the document sources when reproducible calculations are needed.

The equivalent explicit configuration, useful when the helper script is not available, is:

Manual arm64 macOS build

```
cmake -S ../coolprop-src -B ../coolprop-build-macos-arm64 -G Ninja \
  -DCMAKE_BUILD_TYPE=Release \
  -DCMAKE_OSX_ARCHITECTURES=arm64 \
  -DCOOLPROP_SHARED_LIBRARY=ON \
  -DCOOLPROP_STATIC_LIBRARY=OFF \
  -DCOOLPROP_OBJECT_LIBRARY=OFF
cmake -build ../coolprop-build-macos-arm64 -target CoolProp
export LUACOOLPROP_LIB="$PWD/../coolprop-build-macos-arm64/libCoolProp.dylib"
```

No system-wide installation is necessary. Keeping an absolute environment variable beside a pinned build makes it clear which thermodynamic library a document used.

11.5 Linux: download or compile the `.so`

Install Git, CMake, Ninja, Python, a C++ build toolchain, and 7-Zip using the distribution's package manager. For example, Debian and Ubuntu package the usual prerequisites as `git`, `cmake`, `ninja-build`, `build-essential`, `python3`, and `p7zip-full`. After the common source checkout above, configure and build the shared-library target:

Linux shared-library build

```
cmake -S ../coolprop-src -B ../coolprop-build-linux -G Ninja \
  -DCMAKE_BUILD_TYPE=Release \
  -DCOOLPROP_SHARED_LIBRARY=ON \
  -DCOOLPROP_STATIC_LIBRARY=OFF \
  -DCOOLPROP_OBJECT_LIBRARY=OFF
cmake -build ../coolprop-build-linux -target CoolProp
export LUACOOLPROP_LIB="$PWD/../coolprop-build-linux/libCoolProp.so"
```

Again, copying the library to `/usr/local/lib` is optional and normally unnecessary. An unprivileged, versioned build directory plus the absolute `LUACOOLPROP_LIB` path is easier to audit and does not require `sudo`.

11.6 Windows: download or compile `CoolProp.dll`

The simplest Windows installation is an official precompiled x64 DLL. Extract it to a stable directory outside the LuaCoolProp tree and set the environment variable to its absolute path. A source build needs Git, CMake, Python, and either Ninja with a compatible C++ compiler or Visual Studio. In PowerShell, the Ninja workflow is:

Windows PowerShell checkout and Ninja build

```
git clone https://github.com/CoolProp/CoolProp.git ..\coolprop-src
git -C ..\coolprop-src fetch -tags
git -C ..\coolprop-src checkout -detach v8.0.0
git -C ..\coolprop-src submodule update -init -recursive
cmake -S ..\coolprop-src -B ..\coolprop-build-windows -G Ninja `
  -DCMAKE_BUILD_TYPE=Release `
  -DCOOLPROP_SHARED_LIBRARY=ON `
  -DCOOLPROP_STATIC_LIBRARY=OFF `
  -DCOOLPROP_OBJECT_LIBRARY=OFF
cmake -build ..\coolprop-build-windows -target CoolProp
$env:LUACOOLPROP_LIB =
  (Resolve-Path ..\coolprop-build-windows\CoolProp.dll).Path
```

With a Visual Studio generator, select the same architecture as TeX Live and remember that it is a multi-configuration build:

Windows PowerShell and Visual Studio 2022

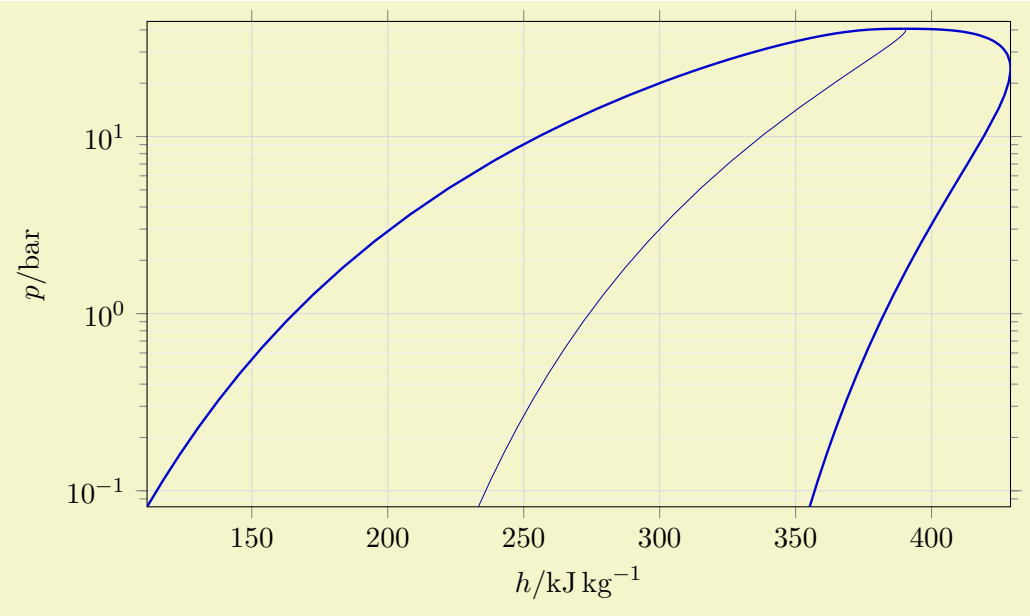
```
cmake -S ..\coolprop-src -B ..\coolprop-build-vs `
  -G "Visual Studio 17 2022" -A x64 `
  -DCOOLPROP_SHARED_LIBRARY=ON `
  -DCOOLPROP_STATIC_LIBRARY=OFF `
  -DCOOLPROP_OBJECT_LIBRARY=OFF
cmake -build ..\coolprop-build-vs -config Release -target CoolProp
$env:LUACOOLPROP_LIB =
  (Resolve-Path ..\coolprop-build-vs\Release\CoolProp.dll).Path
```

CMake versions and generators can place the DLL in a slightly different configuration subdirectory. If the shown path does not exist, locate the newly built `CoolProp.dll` inside the selected build directory rather than copying an unrelated DLL into it. The PowerShell assignment lasts for the current terminal. In `cmd.exe`, use `set "LUACOOLPROP_LIB=C:\absolute\path\CoolProp.dll"`.

11.7 Verify the runtime installation

After setting `LUACOOLPROP_LIB`, compile a minimal trusted document. Use the command appropriate to the format, always in the same terminal in which the environment variable was set:

Minimal installation check




```
\LCPPHDiagram[fluid=R134a,quality values={0,0.5,1}]
```

Run the LaTeX installation check

```
lualatex -shell-escape check-luacoolprop.tex
```

The library path is consumed by the first native request, not by merely requiring the Lua module. If it changes, start a fresh TeX process. Loader errors should first be checked for an incorrect absolute path, an architecture mismatch, or a missing dependent runtime library.

The package is format-generic. Its public files have distinct roles:

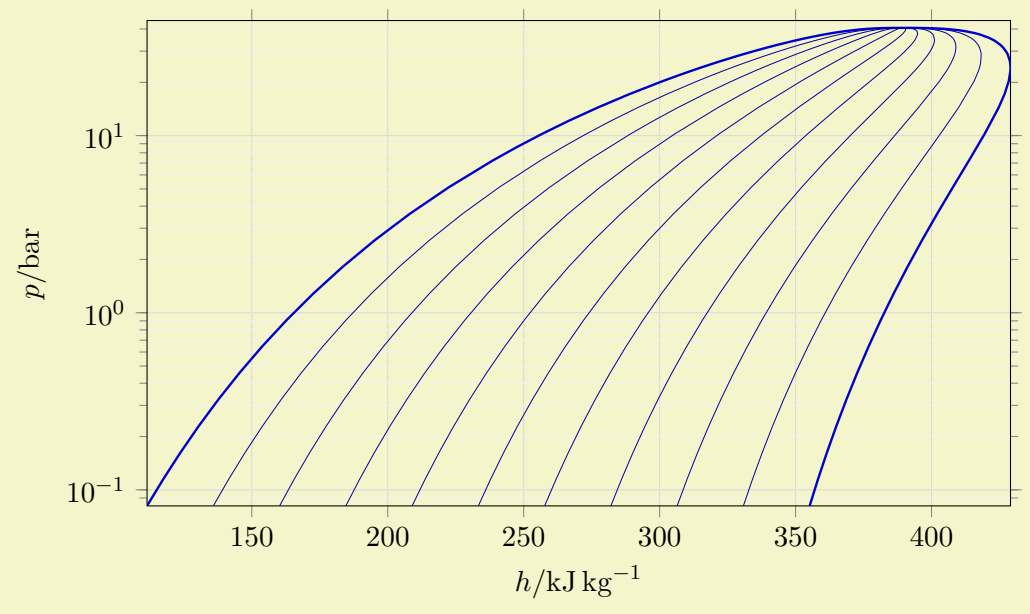
- `luacoolprop.tex` is the canonical generic TeX implementation;

- `luacoolprop.sty` is the LaTeX wrapper;
- `t-luacoolprop.tex` is the ConTeXt wrapper; and
- `p-luacoolprop.tex` is the plain TeX wrapper.

11.8 LaTeX, plain TeX, and ConTeXt

Each format uses its native wrapper and engine. A LaTeX document loads `luacoolprop.sty` and is compiled with LuaLaTeX:

LaTeX wrapper




```
\LCPPHDiagram[fluid=R134a]
```

LaTeX command line

```
lualatex -shell-escape myfile.tex
```

A plain TeX document loads `p-luacoolprop.tex` directly and is compiled with LuaTeX, not with a LaTeX executable:

Plain TeX wrapper

```
\input p-luacoolprop.tex
\pgfplotsset{compat=1.18}
\LCPPHDiagram[fluid=R134a]
\bye
```

Plain TeX command line

```
luatex -shell-escape myfile.tex
```

A ConTeXt document selects the `t-` module explicitly. This avoids an ambiguous module search in a directory that also contains the `p-` plain TeX wrapper.

ConTeXt wrapper

```
\usemodule[t][luacoolprop]
\pgfplotsset{compat=1.18}
\starttext
\LCPPHDiagram[fluid=R134a]
\stoptext
```

ConTeXt command line

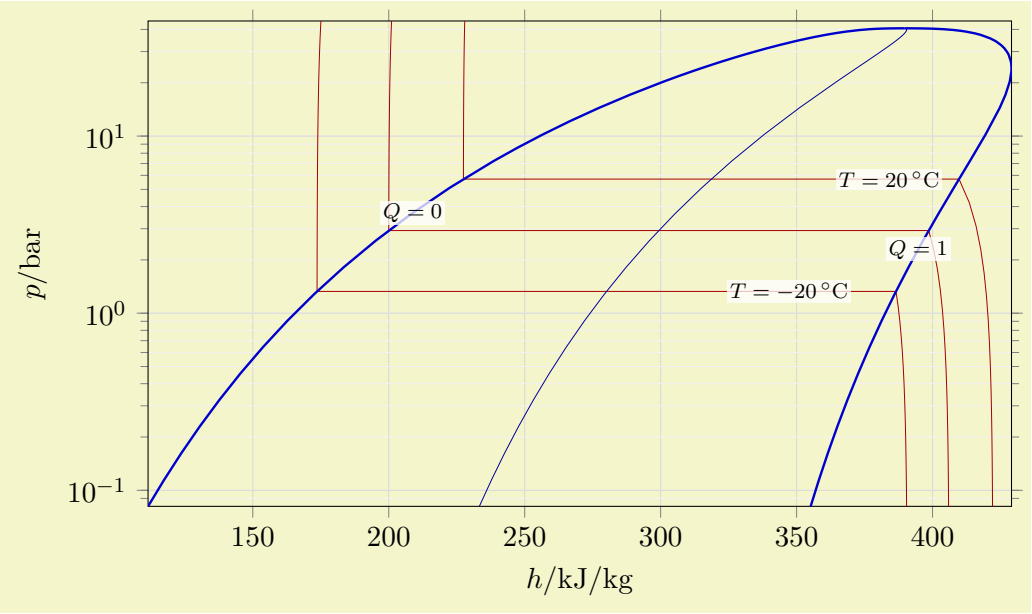
```
context -luatex -shell-escape myfile.tex
```

The `-luatex` switch selects ConTeXt MkIV. It is required because LuaCoolProp uses LuaTeX FFI; LuaMetaTeX/LMTX does not provide that module. The ConTeXt wrapper loads the ConTeXt PGFPlots frontend before the generic layer.

11.9 LaTeX number and unit formatting with `siunitx`

The LaTeX wrapper formats generated unit-bearing values with `\qty`, unit-only axis labels with `\unit`, and dimensionless numbers such as vapour quality with `\num`. If `siunitx` is not yet loaded, `luacoolprop` loads it with its standard defaults. If it is already loaded, the wrapper neither reloads it with options nor calls `\sisetup`; the document's locale, decimal marker, grouping, rounding, font, and per-mode choices are therefore preserved. The user may also call `\sisetup` after loading `luacoolprop`.

Apply the user's siunitx formatting




```
\sisetup{output-decimal-marker={,},per-mode=symbol}
\LCPPHDiagram[
    quality values={0,0.5,1},
    isotherm=true,temperature values={-20,0,20},
    labels=true]
```

The complete executable version is `examples/siunitx-formatting.tex`. Its title contains `\num{12345.6}` and its generated curve labels demonstrate that the same pre-existing decimal and unit policy is retained:

Numerical process-export macros intentionally remain plain decimal tokens so that PGFPlots and TeX arithmetic can consume them. When displaying such a dimensionless macro in LaTeX, wrap it in `\num`; for example, `\num{\CycleABToQuality}`. For a value with a unit, use `\qty`, for example `\qty{\CycleABToTemperatureK}{\kelvin}`. Plain TeX and ConTeXt keep the generic math-mode formatting and do not depend on `siunitx`.

The input grammar is deliberately independent of this output formatting. LuaCoolProp accepts only a full stop as an input decimal separator, even when `siunitx` displays a comma. Decimal floating-point and scientific notation are supported, including a directly attached unit: `pressure=1.2E5Pa`. Thus `1,2bar` is an error, not another spelling of `1.2bar`. This fixed convention

keeps the same source unambiguous under LaTeX, plain TeX, ConTeXt, and every operating-system locale.

11.10 Repository build workflow

The reproducible build script recognizes the format from each example name: `plain-*.tex` uses `luatex`, `context-*.tex` uses ConTeXt MkIV, and every other example uses `lualatex`. The manual uses three `lualatex` passes so its table of contents, cross-references, and two indexes are stable.

Build every bundled document

```
LUACOOLPROP_LIB=/path/to/libCoolProp.dylib ./scripts/build-examples.sh
```

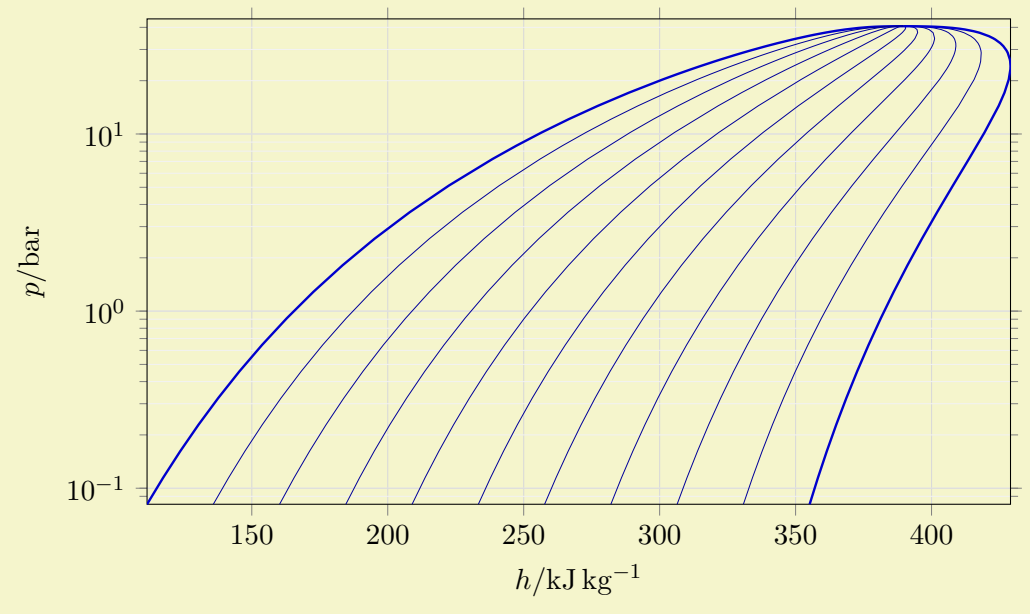
Example logs are written to `examples/logs/`; manual logs are written to `build-logs/`; each completed PDF is installed next to its source. The script stops at the first engine failure or actionable TeX diagnostic. Sandboxed builds may set `LUACOOLPROP_TEX_CACHE` to a writable directory for the LuaLaTeX and plain LuaTeX font cache.

12 Basic users: quick start

12.1 A fully automatic diagram

The high-level command creates the complete TikZ picture and the PGFPlots axis. The default units are h in $\mathrm{kJ\,kg^{-1}}$ and p in bar.

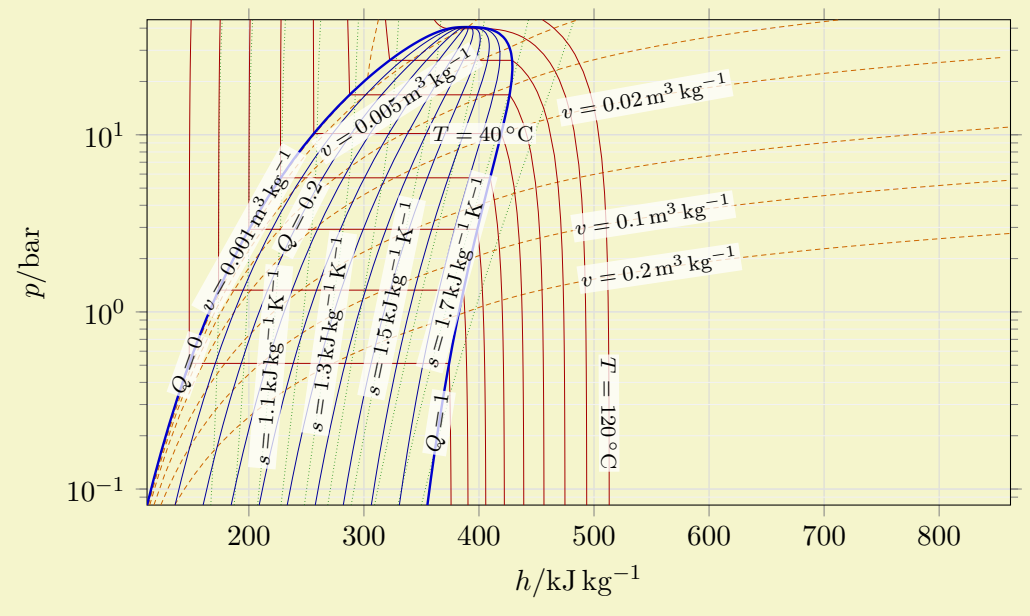
Minimal PH diagram




```
\LCPPHDiagram[fluid=R134a]
```

A more informative chart is obtained by enabling the main isoline families. Label placement is delegated to `pgfplots-autonode`; `label placement=autonode` is the default, but it is shown explicitly below.

Automatic refrigeration-style diagram

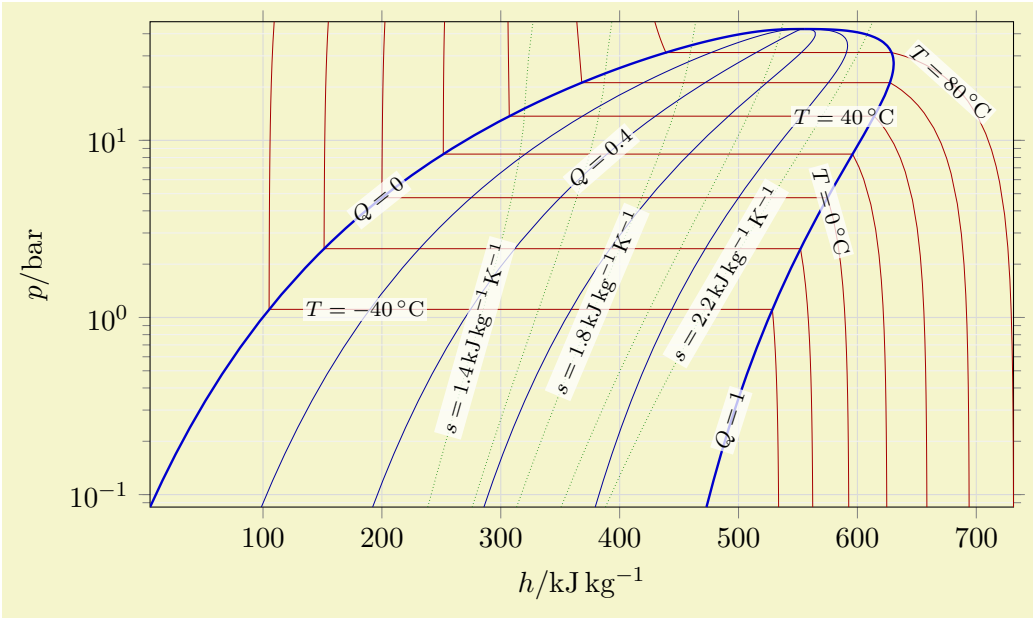



```
\LCPPHDiagram[
  fluid=R134a,
  isotherm=true,
  isentrope=true,
  isochore=true,
  quality step=0.1,
  temperature mode=preset,
  temperature preset=refrigeration,
  entropy mode=preset,
  entropy preset=refrigeration,
  specific volume mode=preset,
  specific volume preset=refrigeration,
  labels=true,
  sloped labels=true,
  label placement=autonode,
  axis options={
    auto node algorithm=repair,
    auto node bbox mode=oriented
  }
]
```

12.2 Another fluid

CoolProp fluid names can be used directly. The useful isoline range depends on the fluid and on the intended engineering application.

Propane quick diagram

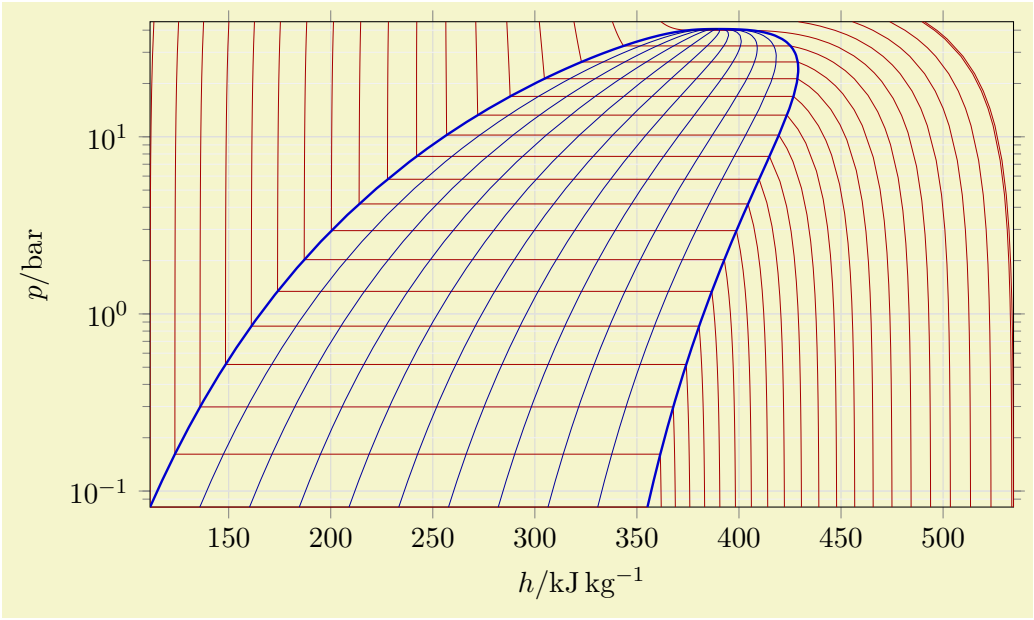



```
\LCPPHDiagram[
  fluid=Propane,
  isotherm=true,
  isentrope=true,
  quality step=0.2,
  temperature values={-40,-20,0,20,40,60,80},
  entropy values={1.4,1.6,1.8,2.0,2.2},
  labels=true,
  sloped labels=true,
  label placement=autonode
]
```

13 Diagram identifiers and stable API

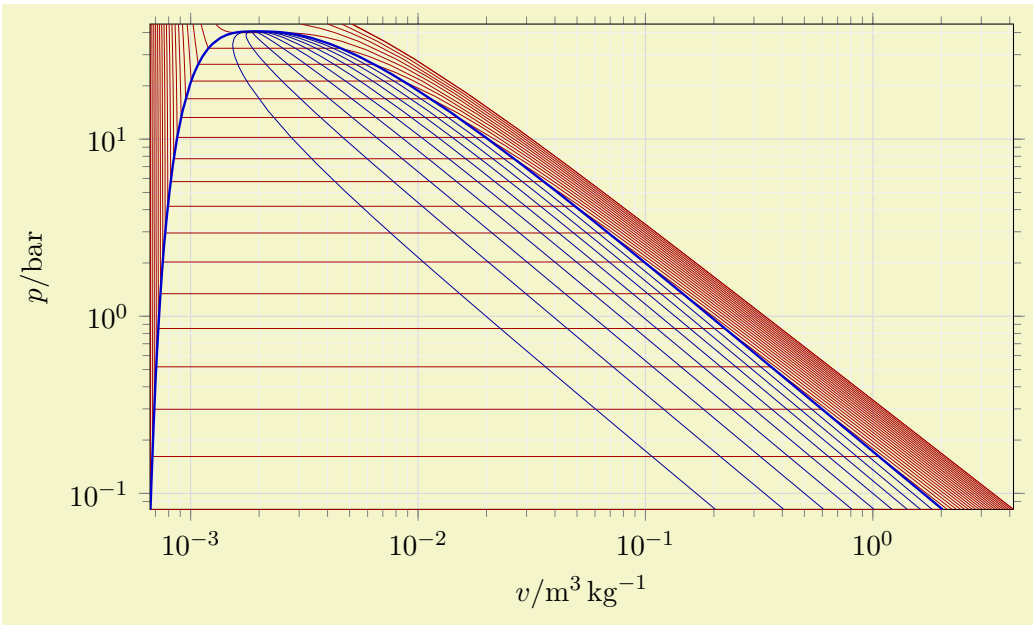
Diagram identifiers are uppercase coordinate pairs. This release implements `PH`, `PV`, `TS`, `HS`, and `PT`. The generic dispatcher keeps the calling convention independent of the coordinate pair:

Generic PH dispatcher



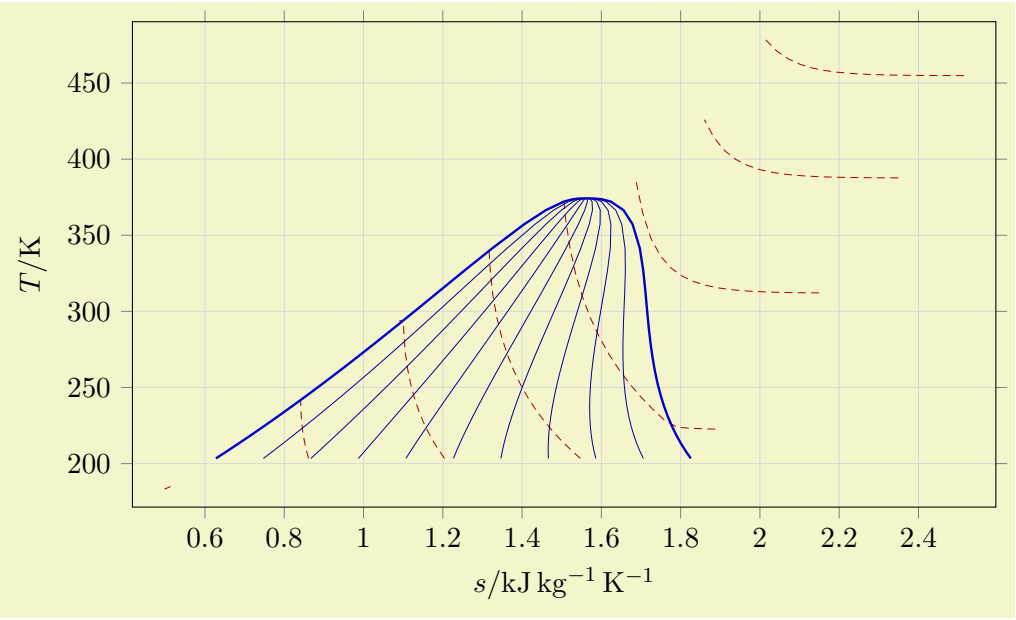

```
\LCPDiagram{PH}[fluid=R134a, isotherm=true]
```

Generic PV dispatcher




```
\LCPDiagram{PV}[fluid=R134a, isotherm=true]
```

Generic TS dispatcher




```
\LCPDiagram{TS}[fluid=R134a, isenthalp=true]
```

Inside an existing PGFPlots axis, use one of these generic forms:

- `\LCPAddDiagramPlots{PH}` or `\LCPAddDiagramPlots{PV}` or `\LCPAddDiagramPlots{TS}`;
- `\LCPAddDiagramFamily{PV}{isotherm}`; or
- `\LCPAddDiagramProcess{PV}`.

Each accepts its options in brackets after the mandatory diagram and family identifiers.

Convenience commands put the same uppercase identifier between the `LCP` prefix and the operation name. Thus `\LCPPHDiagram` and `\LCPPVDiagram`, `\LCPTSDiagram`, and `\LCPHSDiagram` are equivalent to the generic calls with `PH`, `PV`, `TS`, and `HS`. Additional implementations can add other identifiers without changing the generic dispatch commands.

The canonical family names are `quality`, `isotherm`, `isentrope`, `isochore`, `isenthalp`, and `isobar`. Each diagram exposes only the families it can calculate. Public option names use complete English property names. Examples include `quality step`, `temperature values`, and `specific volume unit`.

The canonical advanced key roots pair `/luacoolprop/diagram` and `/luacoolprop/process` with an uppercase `PH`, `PV`, `TS`, `HS`, or `PT` branch.

Accepted aliases

The public aliases include:

- `\LCPPhDiagram`, `\LCPAddPHIsoQuality`, and the mixed-case `Ph` family commands;
- the alternate `/luacoolprop/ph diagram` and `/luacoolprop/ph process` paths; and
- plural family switches and abbreviated property keys.

The canonical names are recommended. Their vocabulary is shared by all registered diagram types.

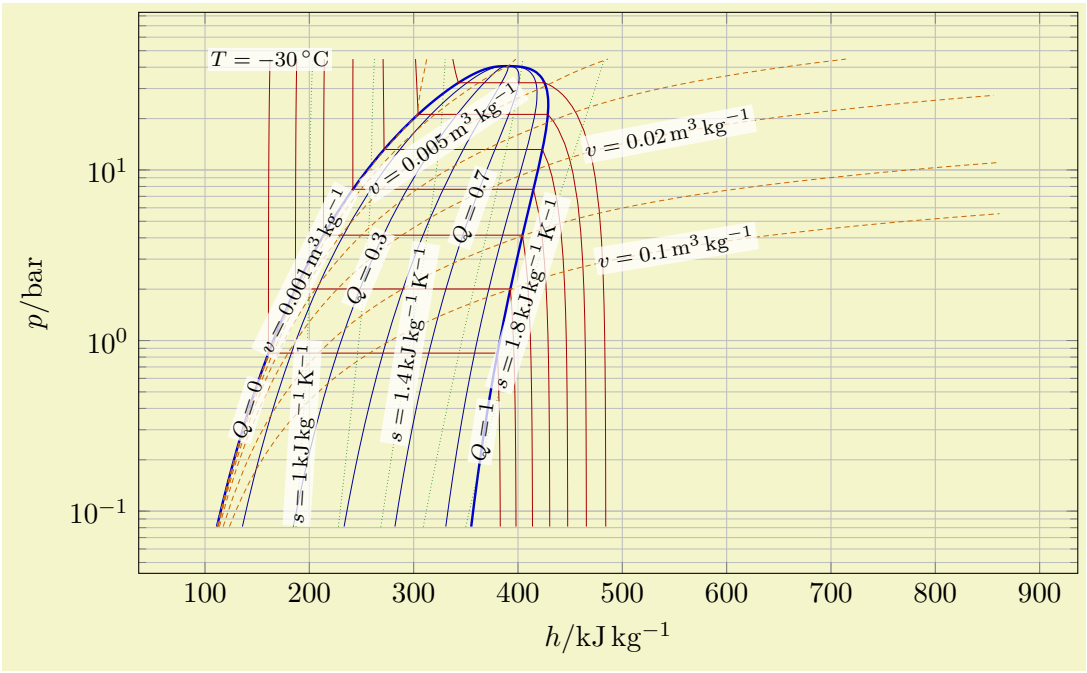
14 Advanced users: manual PGFPlots axes

Advanced users can write the PGFPlots axis by hand and add each isoline family separately. The fluid can be declared at the axis level with `lcp fluid`. Each family macro emits its `\addplot` data immediately, followed by the corresponding `\pgfplotsautonode` label command when labels are enabled. PGFPlots therefore sees the data during its survey phase and can still compute axis limits natively. Because this is a manually created axis, the `auto node placement` lifecycle key is required whenever automatic labels are requested; omitting it produces an explicit warning.

`\begin{axis}[`⟨*options*⟩`]`
⟨*environment contents*⟩
`\end{axis}`

This is PGFPlots' own environment. Place it inside a `tikzpicture` and use the `LCPAdd` commands inside it. Axis limits, logarithmic scales, ticks, legends, and clipping retain their usual PGFPlots meanings. The complete example below shows the required enclosing picture and axis.

Manual construction in an axis



```
\begin{tikzpicture}
\begin{axis}[
  lcp fluid=R134a,
  xlabel={$h/{\rm kJ\,kg^{-1}}$},
  ylabel={$p/{\rm bar}$},
  ymode=log,
  grid=both,
  auto node placement,
  width=14cm,
  height=9cm,
  auto node algorithm=repair,
  auto node bbox mode=oriented
]
  \LCPAddPHQuality[
    quality values={0,0.1,0.3,0.5,0.7,0.9,1},
    labels=true,
    sloped labels=true
  ]
  \LCPAddPHIsotherms[
    temperature values={-30,-10,10,30,50,70,90},
    labels=true,
    sloped labels=true
  ]
  \LCPAddPHIsentropes[
    entropy values={1.0,1.2,1.4,1.6,1.8},
    labels=true,
    sloped labels=true
  ]
  \LCPAddPHIsochores[
    specific volume values={0.001,0.002,0.005,0.01,0.02,0.05,0.1},
    labels=true,
    sloped labels=true
  ]
\end{axis}
\end{tikzpicture}
```

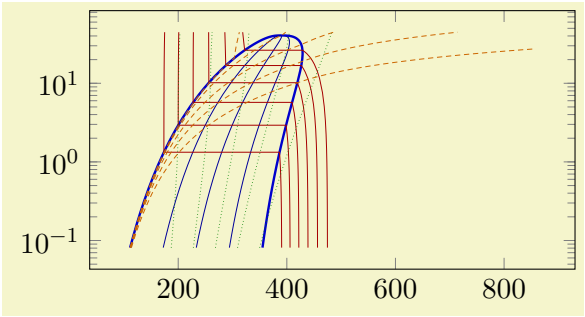
14.1 Curve-family macros

Macro	Purpose
`\LCPAddPHQuality`	add Q = constant curves
`\LCPAddPHIsotherms`	add T = constant curves
`\LCPAddPHIsentropes`	add s = constant curves
`\LCPAddPHIsochores`	add v = constant curves
`\LCPAddPHProcess`	add one thermodynamic process segment

14.2 Value grids

All isoline families accept preset, linear, logarithmic, and explicit value lists. Select a preset with both `temperature mode=preset` (or the corresponding family mode) and the desired `... preset` name. Explicit lists are often best for publication-quality diagrams and take precedence over the mode.

Explicit value lists




```
\begin{tikzpicture}
\begin{axis}[width=8cm,height=5cm,auto node placement,ymode=log]
\LCPAddPHQuality[quality values={0,0.25,0.5,0.75,1}]
\LCPAddPHIsotherms[temperature values={-20,0,20,40,60,80}]
\LCPAddPHIsentropes[entropy values={1.0,1.2,1.4,1.6,1.8}]
\LCPAddPHIsochores[specific volume values={0.001,0.002,0.005,0.01,0.02}]
\end{axis}
\end{tikzpicture}
```

Isochores use density internally because CoolProp's high-level interface accepts mass density directly. LuaCoolProp converts v to $\rho = 1/v$.

15 Automatic labels with pgfplots-autonode

`pgfplots-autonode` is LuaCoolProp's only automatic label-placement backend and is an independent required package. LuaCoolProp loads its public PGFPlots library interface automatically in LaTeX, plain LuaTeX, and ConTeXt; it does not embed a private copy or implement a competing solver. A missing `pgfplots-autonode` installation therefore stops package loading instead of silently degrading label placement.

The independent label-placement manual is the normative reference for `\pgfplotsautonode`, its candidate strategies and assignment algorithms, measured node geometry, boundary constraints, failure policies, and debug facilities. Its examples use ordinary affine PGFPlots networks and require neither LuaCoolProp nor CoolProp.

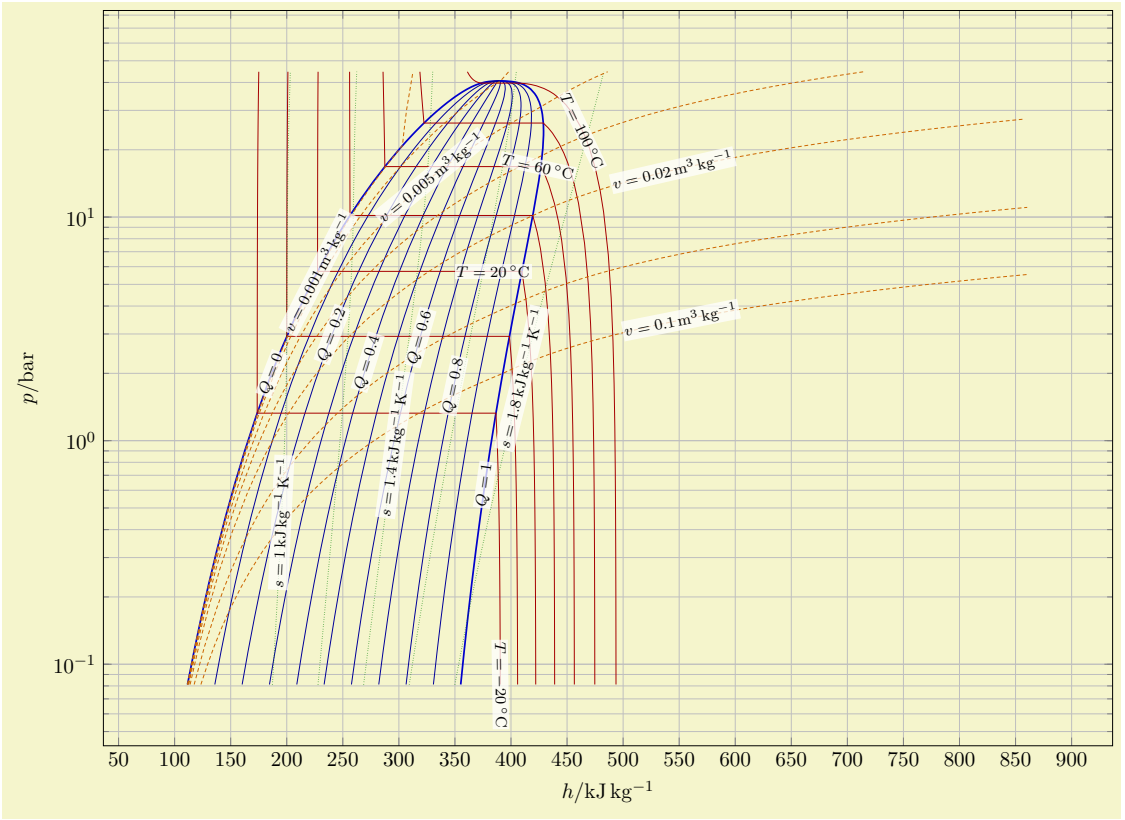
For each labelled thermodynamic curve, LuaCoolProp emits the same structure:

Generated structure

```
\addplot[<curve style>] coordinates {<points>}
  \pgfplotsautonode[<autonode keys>]{<label text>};
```

The label-placement problem is solved through the dependency's documented interface using final PGFPlots path geometry, rather than approximate thermodynamic-coordinate boxes. LuaCoolProp owns the curve and label content; `pgfplots-autonode` owns measurement and placement.

Dense autonode-labelled PH chart


p/bar
10^1
10^0
10^{-1}
50 100 150 200 250 300 350 400 450 500 550 600 650 700 750 800 850 900
h/kJ kg^{-1}
$T = 100\,^\circ$C
$T = 60\,^\circ$C
$T = 20\,^\circ$C
$T = -20\,^\circ$C
$v = 0.001\,\mathrm{m^3\,kg^{-1}}$
$v = 0.005\,\mathrm{m^3\,kg^{-1}}$
$v = 0.02\,\mathrm{m^3\,kg^{-1}}$
$v = 0.1\,\mathrm{m^3\,kg^{-1}}$
$Q = 0$
$Q = 0.2$
$Q = 0.4$
$Q = 0.6$
$Q = 0.8$
$Q = 1$
$s = 1\,\mathrm{kJ\,kg^{-1}\,K^{-1}}$
$s = 1.4\,\mathrm{kJ\,kg^{-1}\,K^{-1}}$
$s = 1.8\,\mathrm{kJ\,kg^{-1}\,K^{-1}}$

```
\begin{tikzpicture}
\begin{axis}[
  lcp fluid=R134a,
  xlabel={$h/{\rm kJ\,kg^{-1}}$},
  ylabel={$p/{\rm bar}$},
  ymode=log,
  grid=both,
  auto node placement,
  auto node algorithm=repair,
  auto node bbox mode=oriented,
  auto node failure mode=error,
  width = 20cm,
  height = 15cm,
]
  \LCPAddPHQuality[
    quality values={0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1},
    labels=true,
    sloped labels=true
  ]
  \LCPAddPHIsotherms[
    temperature values={-20,0,20,40,60,80,100},
    labels=true,
    sloped labels=true
  ]
  \LCPAddPHIsentropes[
    entropy values={1.0,1.2,1.4,1.6,1.8},
    labels=true,
    sloped labels=true
  ]
  \LCPAddPHIsochores[
    specific volume values={0.001,0.002,0.005,0.01,0.02,0.05,0.1},
    labels=true,
    sloped labels=true
  ]
\end{axis}
\end{tikzpicture}
```

15.1 LuaCoolProp keys forwarded to autonode

Key	Meaning
`autonode candidates`	number of candidate positions sampled on each plot path
`autonode candidate strategy`	distribution strategy for candidates
`autonode clearance`	additional clearance around measured labels
`autonode inner sep`	node padding used by the backend
`autonode normal shift`	shift normal to the sampled curve
`autonode preferred weight`	penalty for moving away from the preferred position
`autonode overlap weight`	multiplier for label-label overlap area during repair and local search

Native `pgfplots-autonode` axis keys such as `auto node algorithm`, `auto node bbox mode`, `auto node failure mode`, and `auto node border margin` can be passed directly to the PGF-Plots axis, or through `axis options` when using `\LCPPHDiagram`.

For a pair of overlapping labels, the solver uses the arithmetic mean of their two overlap weights. This symmetric cost can trade displacement against a residual collision in repair and local-search modes; the final failure policy still decides whether such a collision is accepted, reported, or hidden.

Sloped autonode labels are kept upright by default. Use `label allow upside down=true` only when the raw path orientation is desired.

16 Curve names and manual overrides

Each curve receives a stable name. Examples are:

Generated curve identifiers

```
lcp-ph-q-0p4
lcp-ph-T-20C
lcp-ph-s-1p4kJkgK
lcp-ph-v-0p005m3kg
```

Manual corrections are expressed as key-value overrides. With autonode these corrections are translated to preferred positions, admissible intervals, node styles, or visibility flags.

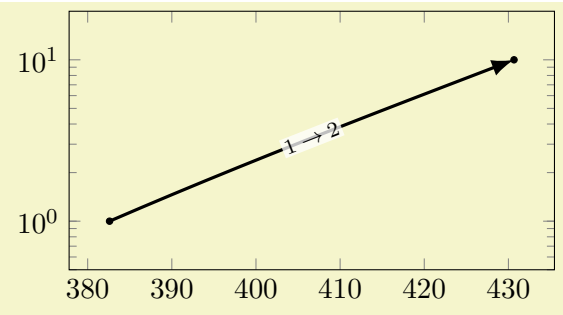
Manual label and curve overrides

```
label pos for={lcp-ph-T-0C}{0.72}
label fixed for={lcp-ph-T-0C}{true}
label style for={lcp-ph-T-0C}{fill=yellow!20,draw=yellow!50!black}
label show for={lcp-ph-v-0p005m3kg}{false}
curve style for={lcp-ph-v-0p01m3kg}{very thick,orange!90!black}
```

17 Thermodynamic process paths

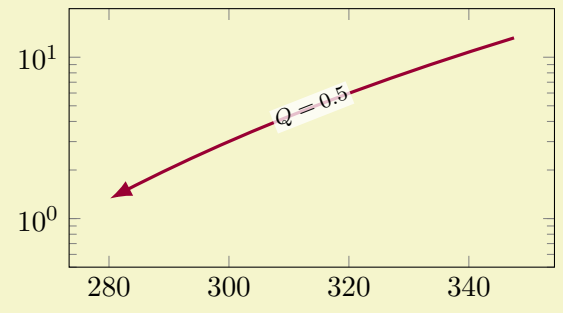
The macro `\LCPAddPHProcess` draws a process segment on top of an existing PH diagram. The type determines the conserved property and LuaCoolProp validates whether enough information has been provided. Arrow-tip styles in the following snippets require `\usetikzlibrary{arrows.meta}` in the document preamble.

Isentropic compression from saturated vapor




```
\begin{tikzpicture}
\begin{axis}[width=8cm,height=5cm,auto node placement,ymin=0.5,ymax=20,ymode=log]
\LCPAddPHProcess[
  fluid=R134a,
  type=isentrope,
  from={pressure=1bar,quality=1},
  to={pressure=10bar},
  style={very thick,-Latex},
  color=black,
  label={$1\to2$},
  mark endpoints=true
]
\end{axis}
\end{tikzpicture}
```

Constant-quality segment




```
\begin{tikzpicture}
\begin{axis}[width=8cm,height=5cm,auto node placement,ymin=0.5,ymax=20,ymode=log]
\LCPAddPHProcess[
  fluid=R134a,
  type=quality,
  quality=0.5,
  from={temperature=50C},
  to={temperature=-20C},
  style={very thick,-Latex},
  color=purple!80!black,
  label={$Q=0.5$}
]
\end{axis}
\end{tikzpicture}
```

If an endpoint is under-specified, the error message indicates the missing thermodynamic property. For instance, `from= {quality=1}` is not enough to define a saturated state; the endpoint also needs a saturation coordinate such as `pressure=1bar` or `temperature=0C`.

Part IV
Complete TeX reference

18 How to read this reference

This part is the normative TeX reference for `luacoolprop`. Every public command and key uses a native PGF manual reference entry and at least one concrete assignment. The examples in "source/result" boxes are compiled as part of this manual: the code shown on the right is exactly the code which produced the result on the left. Longer examples put the result above the source. This is the same literate, executable-documentation principle used by the PGF/TikZ and PGFPlots manuals. The reference entries and executable examples use PGF's own `command`, `key`, and `codeexample` facilities.

Complete LaTeX diagrams execute inside this manual. Complete plain-TeX, ConTeXt MkIV, and Lua programs are extracted verbatim and tested by the documentation build with their respective engines. Installation commands, paths that must be supplied by the user, and signatures containing ⟨*placeholders*⟩ are instructions or syntax templates, not runnable programs. Tutorial excerpts are explained in the context of their complete executable definitions in Appendix A.

Signatures use ⟨*argument*⟩ for a required semantic value, `{`⟨*argument*⟩`}` for a braced argument, and `[`⟨*options*⟩`]` for an optional key list. Boolean keys take `true` or `false`. A fraction along a curve is in the closed interval $[0, 1]$. Temperatures, pressures, enthalpies, entropies, and specific volumes use the units stated by the corresponding unit or scale key.

All numeric input is locale independent. Use a full stop as the decimal separator, irrespective of the document language or the output settings of `siunitx`. Integers, decimal floating-point numbers, and decimal scientific notation are accepted: `12`, `-0.25`, `.5`, and `1.2E5` are valid. A

recognized unit may immediately follow a process quantity, as in `1.2E5Pa`. Decimal commas, hexadecimal Lua literals, non-finite values, trailing characters, and unknown units are errors. Numeric lists use commas, spaces, or semicolons only to separate complete numbers. Likewise, malformed booleans, modes, presets, list members, and reversed explicit bounds are rejected instead of being replaced by defaults.

Namespaces and extensibility

The canonical root is `/luacoolprop/diagram`, with uppercase `PH`, `PV`, `TS`, `HS`, and `PT` branches. Keys are passed without that prefix in command option lists. The generic commands take an uppercase identifier and allow registered custom coordinate systems. The lowercase `ph diagram`, `pv diagram`, `ts diagram`, `hs diagram`, and `pt diagram` branches directly below `/luacoolprop` are only alternate public paths. No diagram type supplies defaults or capabilities to another type.

18.1 Scope, units, and validation contracts

Each key entry gives its full defining path, value type, default, interactions, and a configuration example. The displayed configuration snippets are applied in isolated TeX groups when this manual is built. The source/result examples in each family show complete constructions using the same handlers. A snippet that sets a library path tests configuration syntax; it does not attempt to load a library from that illustrative path.

Global scope Keys directly below `/luacoolprop` set defaults for subsequent calls within the current TeX group. They do not override a more specific option supplied to a diagram or process. Loading a native library and locking a fluid's reference state are process-lifetime effects, even when the TeX configuration is locally scoped.

Diagram scope Keys below `/luacoolprop/diagram/common` are shared controls; a type-specific namespace inherits them unless it declares an override. A diagram command initializes its own defaults, applies its options locally, and restores the surrounding settings on return. Unsupported background families are errors. Per-curve overrides take precedence over family settings.

Process scope Keys below `/luacoolprop/process/common` describe a single process call. The `PH`, `PV`, `TS`, `HS`, and `PT` namespaces add projection-specific units and numerical controls. State exports use SI properties; their plotted coordinates use the selected coordinate scales.

PGFPlots scope The fluid style is evaluated in the axis's TeX group and sets the fluid default used by its curves. Ordinary PGFPlots and TikZ options belong to their native namespaces. Pass axis options through `axis options`, and plot or node options through the documented style keys; unknown LuaCoolProp keys are never forwarded implicitly.

Grid units Sampling pressure bounds are in pascals. Temperature, entropy, specific-volume, enthalpy, and HS isobar grids follow `temperature unit`, `entropy unit`, `specific volume unit`, `enthalpy unit`, and `isobar unit`, respectively. A unit selector changes interpretation of the associated values, bounds, and increments; explicit values take precedence over generated spacing. Quality is a mass fraction between zero and one.

Process-state units Bare process pressures, temperatures, specific enthalpies, specific entropies, and specific volumes use Pa, K, J/kg, J/(kg K), and cubic metres per kilogram. Recognized suffixes select another unit. The generic `value` follows the conserved property selected by `type`. Endpoints must provide an independent state pair or enough data to complete that pair from the process invariant.

Coordinate and numerical units Raw scales multiply SI properties to obtain plot coordinates; semantic axis-unit keys select both that conversion and its label. Temperature may also have an offset. Axis bounds use plot coordinates. Adaptive tolerances and weights belong to the projection's sampling metric, described in its numerical controls; they are not absolute thermodynamic error bounds.

Other values Booleans, names, counts, depths, fractions, modes, presets, and label positions carry no physical unit. TeX text and style lists are not thermodynamic quantities. Explicit TeX dimensions and PGFPlots style values follow their native units. A key's entry states its allowed choices and any interaction with automatic values or other keys.

A misspelled key raises a TeX error in global, diagram, and process options, in LaTeX, plain TeX, and ConTeXt. For example, `presure min=1e5` is invalid; use `pressure min=1e5`. Normal documented aliases remain accepted. There is no warning-only fallback.

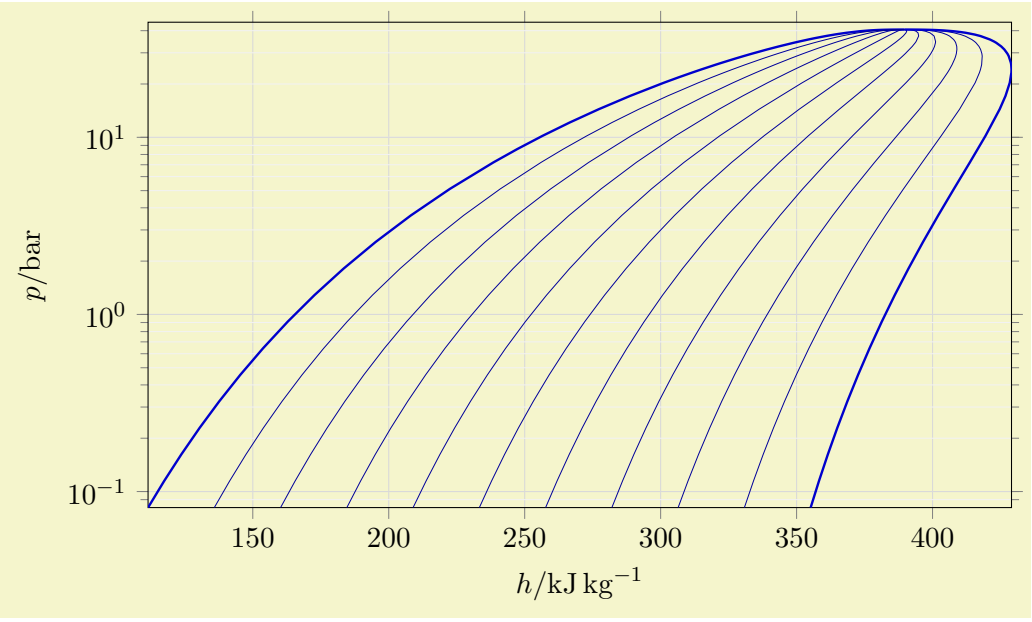
The following background switches are deliberately rejected, including their plural aliases. Their constant-property process paths can still be drawn with `\LCPAddDiagramProcess`; a missing background family does not remove a physical process from the API.

`PH`: `isenthalp,isenthalps,isobar,isobars`.
`PV`: `isenthalp,isenthalps,isobar,isobars,isochore,isochores`.
`TS`: `isentrope,isentropes,isobar,isobars,isochore,isochores,isotherm,isotherms`.
`HS`: `isenthalp,isenthalps,isentrope,isentropes`.
`PT`: `isobar,isobars,isoquality,quality,isotherm,isotherms`.

19 Loading in the three supported formats

The generic layer is `luacoolprop.tex`. The format files only arrange loading: `luacoolprop.sty` for LaTeX, `t-luacoolprop.tex` for ConTeXt, and `p-luacoolprop.tex` for plain TeX.

LaTeX




```
\LCPPHDiagram
```

Plain LuaTeX

```
\input p-luacoolprop.tex
\pgfplotsset{compat=1.18}
\LCPPHDiagram
\bye
```

ConTeXt

```
\usemodule[t][luacoolprop]
\pgfplotsset{compat=1.18}
\starttext
  \LCPPHDiagram
\stoptext
```

All three formats require full shell escape because the TeX Live engine exposes the FFI module used to load the external CoolProp shared library only in that mode. Use `lualatex -shell-escape`, `luatex -shell-escape`, or `context -luatex -shell-escape`, as appropriate. This option allows every trusted or untrusted input in the document to execute external programs; read the security warning in the installation chapter before enabling it.

20 Global configuration and fluid constants

\LCPSetup{⟨*options*⟩}

Changes persistent package defaults. The settings apply to subsequently created diagrams and process paths.

Select a fluid and plotting scales globally

```
\LCPSetup{fluid=Propane,
  enthalpy scale=0.001,
  pressure scale=0.00001}
```

\LCPSetFluid{⟨*fluid*⟩}

Sets the global CoolProp fluid identifier. It does not query CoolProp until a diagram is drawn or constants are updated.

Use propane in subsequent operations

```
\LCPSetFluid{Propane}
```

\LCPSetLibrary{⟨*path*⟩}

Selects the CoolProp shared-library path before the first native calculation. An explicit selection is authoritative; changing an already loaded library requires a fresh compiler process. Usually the environment variable `LUACOOLPROP_LIB` is preferable because the TeX source then remains portable. Leading and trailing whitespace around a braced path is ignored; spaces inside a directory or file name are preserved.

Explicit shared-library path

```
\LCPSetLibrary{/opt/coolprop/libCoolProp.dylib}
```

\LCPSetReferenceState{⟨*convention*⟩}

Selects the CoolProp enthalpy/entropy reference convention for the current global fluid. The accepted conventions are `DEF`, `IIR`, `ASHRAE`, and `NBP`. Call this command during initialization, before updating constants, creating a diagram, or evaluating a process for the fluid. The first diagram calculation locks the choice; a later attempt to change it is an error because it would mix incompatible h and s coordinates in one run. Repeating the same convention is harmless.

Select the IIR reference during initialization

```
\LCPSetFluid{R134a}
\LCPSetReferenceState{IIR}
```

\LCPUpdateFluidConstants

Queries CoolProp for the current fluid and refreshes all constant macros listed below. Call it after changing the fluid with `\LCPSetFluid`.

Refresh the constants of the current fluid

```
\LCPSetFluid{R134a}
\LCPUpdateFluidConstants
```

\LCPDeclareFluid{⟨*fluid*⟩}

Combines `\LCPSetFluid` and `\LCPUpdateFluidConstants`.

Query and typeset fluid constants

Fluid: R134a$T_{\mathrm c}=101.061966585\,^{\circ}\mathrm C$ $p_{\mathrm c}=40.5927637379\,\mathrm{bar}$

```
\LCPDeclareFluid{R134a}
Fluid: \LCPFluidName\par
$T_{\mathrm c}=\LCPFluidTcritC\,^{\circ}\mathrm C$\par
$p_{\mathrm c}=\LCPFluidPcritBar\,\mathrm{bar}$
```

The update command defines the following expandable result macros. Empty values mean that no update has run yet.

\Command	Meaning and unit
`\LCPFluidName`	CoolProp fluid identifier.
`\LCPFluidReferenceState`	Active enthalpy/entropy reference convention.
`\LCPFluidPcritSI`	Critical pressure in pascals.
`\LCPFluidPcritBar`	Critical pressure in bar.
`\LCPFluidPcritPlot`	Critical pressure after the global pressure scale.
`\LCPFluidPtripleSI`	Triple-point pressure in pascals.
`\LCPFluidPtripleBar`	Triple-point pressure in bar.
`\LCPFluidTcritK`	Critical temperature in kelvin.
`\LCPFluidTcritC`	Critical temperature in degrees Celsius.
`\LCPFluidTtripleK`	Triple-point temperature in kelvin.
`\LCPFluidTtripleC`	Triple-point temperature in degrees Celsius.
`\LCPFluidRhocritSI`	Critical mass density in kilograms per cubic metre.

20.1 Global keys: /luacoolprop

/luacoolprop/fluid=⟨*CoolProp identifier*⟩ (no default)

Default: R134a.

Sets the default fluid and has the same effect as \LCPSetFluid.

Example: fluid=Propane.

/luacoolprop/library=⟨*file path*⟩ (no default)

Default: empty.

Overrides discovery of the external shared library before the first native calculation. A bad explicit selection raises an error without fallback. The loaded library is fixed for the compiler process, regardless of TeX grouping. The environment variable LUACOOLPROP_LIB is an alternative.

Example: library=/opt/coolprop/libCoolProp.dylib.

/luacoolprop/cache=⟨*off, read-write, refresh, or read-only*⟩ (no default)

Default: off.

Sets the default numerical cache policy for every diagram and process. Scoped by TeX grouping. See Section 21.

Example: cache=read-write.

/luacoolprop/cache directory=⟨*directory path*⟩ (no default)

Default: build/luacoolprop-cache.

Sets the default cache directory, relative to the compiler working directory unless absolute. Spaces are allowed. The directory is created only when writing; it must be writable for read-write and refresh.

Example: cache directory=build/luacoolprop-cache.

/luacoolprop/reference state=⟨*DEF, IIR, ASHRAE, or NBP*⟩ (no default)

Default: DEF.

Selects the persistent reference convention used for both enthalpy and entropy. Set it before the first calculation for each fluid.

Example: reference state=IIR.

/luacoolprop/enthalpy axis unit=⟨*jkg or kjkg*⟩ (no default)

Default: kjkg.

Selects both the enthalpy coordinate conversion and its generated axis label. Prefer this semantic key to a raw scale.

Example: enthalpy axis unit=jkg.

/luacoolprop/pressure axis unit=⟨*pa, kpa, mpa, or bar*⟩ (no default)

Default: bar.

Selects both the pressure coordinate conversion and its generated axis label.

Example: pressure axis unit=mpa.

/luacoolprop/specific volume axis unit=⟨*m3kg or lkg*⟩ (no default)

Default: m3kg.

Selects both the specific-volume coordinate conversion and its generated axis label.

Example: specific volume axis unit=lkg.

`/luacoolprop/entropy axis unit=`⟨*jkgk or kjkgk*⟩ (no default)

Default: `kjkgk`.

Selects both the entropy coordinate conversion and its generated axis label.

Example: `entropy axis unit=jkgk`.

`/luacoolprop/temperature axis unit=`⟨*kelvin or celsius*⟩ (no default)

Default: `kelvin`.

Selects the TS ordinate and label together. Celsius applies the additive offset $T_{\mathrm{plot}} = T_{\mathrm{SI}} - 273.15$, not merely a multiplier.

Example: `temperature axis unit=celsius`.

`/luacoolprop/enthalpy scale=`⟨*number*⟩ (no default)

Default: `0.001`.

Multiplies CoolProp enthalpies in SI units before plotting. The default converts `J/kg` to `kJ/kg`. This advanced raw interface deliberately replaces the named-unit label by an explicit scaled-SI formula; use `enthalpy axis unit` for an ordinary unit label.

Example: `enthalpy scale=0.001`.

`/luacoolprop/pressure scale=`⟨*number*⟩ (no default)

Default: `0.00001`.

Multiplies CoolProp pressures in pascals before plotting. The default converts pascals to bar. A raw value produces a scaled-SI axis label.

Example: `pressure scale=0.00001`.

`/luacoolprop/specific volume scale=`⟨*number*⟩ (no default)

Default: `1`.

Multiplies CoolProp mass-specific volumes in SI units before plotting a PV diagram. The default retains `m3/kg`; a raw value produces a scaled-SI axis label.

Example: `specific volume scale=1`.

`/luacoolprop/entropy scale=`⟨*number*⟩ (no default)

Default: `0.001`.

Multiplies CoolProp mass-specific entropy in `J/(kg K)` before plotting a TS or HS diagram. The default gives `kJ/(kg K)`; a raw value produces a scaled-SI axis label.

Example: `entropy scale=0.001`.

`/luacoolprop/temperature scale=`⟨*number*⟩ (no default)

Default: `1`.

Multiplies CoolProp absolute temperatures before plotting a TS diagram. A raw scale has zero offset and produces a scaled-SI label.

Example: `temperature scale=1`.

Accepted alias: `h scale` forwards to `enthalpy scale`.

Accepted alias: `p scale` forwards to `pressure scale`.

Accepted alias: `v scale` forwards to `specific volume scale`.

Accepted alias: `s scale` forwards to `entropy scale`.

Accepted alias: `t scale` forwards to `temperature scale`.

21 Reusing numerical curves and processes

Numerical caching is optional and disabled by default. It applies equally to PH, PV, TS, HS and PT, including individual family commands and process paths. It saves computed points and completed process endpoints; it does not save a rendered picture. PGFPlots still draws the paths, and the independent `pgfplots-autonode` library still measures and places the current labels. Changes to colors, label text or fonts can therefore reuse numerical data. Adding a curve calculates that curve without discarding other compatible entries.

`off`	Compute normally; do not read or write the cache.
`read-write`	Reuse a compatible memory/disk entry; compute and save a miss.
`refresh`	Recompute requested numerical entries and replace their stored values.
`read-only`	Require compatible data; a missing or invalid entry is an error.

A process-local cache retains at most 128 entries and 16 MiB of encoded data; older entries are evicted without deleting their disk files. Each returned Lua table is independent of earlier callers. A disk entry is limited to 32 MiB. Neither policy limits the total disk-directory size: removing that project's cache directory between compilations is safe when recalculation is allowed. Memory entries remain valid until eviction or `lcp.cache.clear_memory()`.

`/luacoolprop/diagram/common/cache`=⟨*off, read-write, refresh, or read-only*⟩ (no default)

Default: `inherited`.

Overrides the global policy for this diagram or family request. All five diagram branches inherit this key. The default comes from `/luacoolprop/cache`.

Example: `cache=read-write`.

`/luacoolprop/diagram/common/cache directory`=⟨*directory path*⟩ (no default)

Default: `inherited`.

Overrides the global directory for this diagram or family request. Numerical signatures are independent of the directory and rendering style.

Example: `cache directory=build/luacoolprop-cache`.

`/luacoolprop/process/common/cache`=⟨*off, read-write, refresh, or read-only*⟩ (no default)

Default: `inherited`.

Overrides the global policy for this process. A hit still defines the current endpoint export macros and emits requested coordinate log messages. Process names and rendering styles are applied to the reused physical states.

Example: `cache=read-write`.

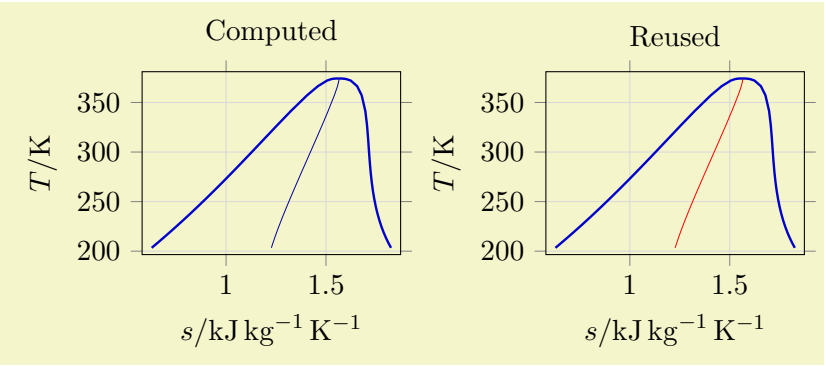
`/luacoolprop/process/common/cache directory`=⟨*directory path*⟩ (no default)

Default: `inherited`.

Overrides the global directory for this process, in all five projections.

Example: `cache directory=build/luacoolprop-cache`.

This example first refreshes three TS quality curves and then requires those same data for a differently styled rendering. The code is compiled as part of this manual; the second diagram must not need numerical curve generation.




```
\begingroup
  \LCPSetup{cache directory={build/manual-numerical-cache}}
  \LCPTSDiagram[
    fluid=R134a,quality values={0,0.5,1},labels=false,
    cache=refresh,
    axis options={width=5cm,height=4cm,title={Computed}}
  ]
  \LCPTSDiagram[
    fluid=R134a,quality values={0,0.5,1},labels=false,
    cache=read-only,quality color=red,
    axis options={width=5cm,height=4cm,title={Reused}}
  ]
\endgroup
```

The signature includes the selected runtime's source fingerprints, the actual CoolProp binary fingerprint, version, revision, adapter, ABI and capabilities, the fluid and reference state, supported native configuration changes, and the numerical options and input constraints. The source/binary fingerprints are computed once per Lua process. Coordinate scaling and domain bounds are part of the signature when passed to the numerical generator: they affect adaptive sampling. Input spellings and reference-initialization order can conservatively produce different entries. Cache files preserve binary64 values, critical endpoints, disconnected segments and warning messages, not merely rounded coordinates intended for typesetting.

Caching supports pure fluids with CoolProp's default HEOS backend (or an explicit `HEOS::` prefix). Other backends require `cache=off`, because their external data have no complete fingerprint in this implementation. The loader resolves the actual readable shared-library file even when selection uses a dynamic-loader name. Use an explicit `library` path when several installations coexist. Native configuration changes must go through LuaCoolProp's `set_config_*` functions. Direct mutations through the raw FFI handle, replacement computational functions, or a second native binding are outside the cache contract; disable caching for such documents. Reference-state checks and locks remain active on hits. Read-only mode still requires CoolProp and shell escape; it is not an offline import mode.

An invalid entry is recomputed in read-write mode and rejected in read-only mode. Permission errors and failed writes are explicit errors. Publication uses a temporary file and atomic rename within the cache directory; incomplete files are never read as entries. On systems that cannot atomically replace an existing file, a concurrent identical entry is accepted; other replacement failures are reported rather than silently deleting the old entry. Do not change the selected runtime or native library while a compiler process is running. Cache files contain data only and are never executed as Lua, but checksums are not an authenticity guarantee for files obtained from strangers.

The practical benefit depends on the figure: numerical reuse does not remove TeX's label-composition or collision-resolution costs. For unchanged complete LaTeX figures, TikZ externalization can additionally reuse the rendered PDF. That is a separate mechanism; custom exported state macros and native-library changes need their own dependency handling.

22 Externalizing complete pictures with TikZ

For a finished figure, the TikZ `external` library can save the rendered PDF and include it on later runs. This avoids both thermodynamic curve calculation and label measurement/placement. Numerical caching instead keeps the figure editable: PGFPlots and `pgfplots-autonode` still render it with current styles. The two mechanisms can be combined.

The following complete LaTeX document is compiled independently from its exact listed source during the manual build. TikZ's documented externalization workflow is LaTeX-specific; use explicit tables or separately compiled figures for plain TeX and ConTeXt. Consult the TikZ manual's chapter *Externalization Library* for its own keys and dependency-management facilities.

```
\documentclass{article}
\usepackage{luacoolprop}
\usetikzlibrary{external}
\pgfplotsset{compat=1.18}
\directlua{lfs.mkdir("build"); lfs.mkdir("build/external-tikz")}
\tikzexternalize[prefix=build/external-tikz/]
\tikzset{external/system call={lualatex
  \tikzexternalcheckshellescape -halt-on-error
  -interaction=batchmode -jobname "\image" "\texsource"}}
\LCPSetup{cache=read-write}
\begin{document}
\tikzsetnextfilename{water-ts}
\begin{tikzpicture}
  \begin{axis}[
    width=8cm,height=6cm,auto node placement,
    xlabel={$s\,[\mathrm{kJ\,kg^{-1}\,K^{-1}}]$},
    ylabel={$T\,[\mathrm{K}]$}
  ]
    \LCPAddTSQuality[
      fluid=Water,quality values={0,0.5,1},labels=true,
      quality label every=1
    ]
  \end{axis}
\end{tikzpicture}
\end{document}
```

Compile with `lualatex -shell-escape main.tex`. Both the main process and its child compiler must resolve the same package and CoolProp library. A `library` key in the preamble or an inherited `LUACOOLPROP_LIB` environment variable supplies the native path. The example creates its output directories with Lua, not a platform-specific shell command. Its explicit axis keeps all property calculations inside the picture; the fully automatic diagram commands may prepare axis limits before TikZ intercepts the picture, and therefore still need CoolProp on a PDF reuse. Choose a unique explicit filename for each externalized figure.

TikZ's picture checksum does not identify every dependency of a thermodynamic figure. In particular, changes to CoolProp, fluid reference conventions, LuaCoolProp, autonode, preamble macros, fonts or global setup may require an explicit rebuild even when the picture's literal source is unchanged. Use `\tikzset{external/force remake}` for a rebuilding run, or remove the affected external figure's generated files. Merely refreshing the numerical cache cannot update a PDF which TikZ decides to reuse.

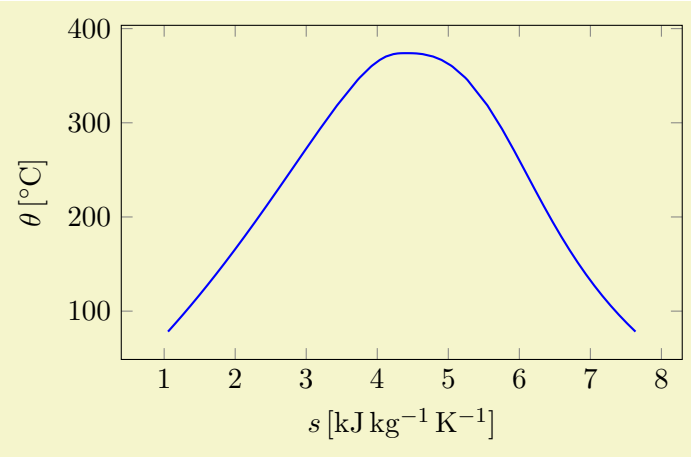
Process endpoint macros defined *inside* a picture must not be relied upon outside it: reusing the PDF does not execute the picture body, and a child compiler has its own TeX/Lua state. Calculate values needed by surrounding text outside the externalized picture, or deliberately export and consume the required data. Ordinary PGFPlots labels inside the saved PDF are already rendered; their positions will change only when that PDF is rebuilt.

23 Exporting numerical tables for PGFPlots

An explicit export is a portable scientific artifact, separate from the private numerical cache. A single text file contains a JSON manifest in its first comment line and a PGFPlots-compatible table. It can be inspected, archived and plotted without loading LuaCoolProp or CoolProp. Generating it still requires the native library and shell escape; supply a readable native file so the manifest can record its fingerprint. Numerical cache options apply to the export calculation just as they do to a drawing command.

`\LCPExportDiagramTable{`⟨*type*⟩`}[`⟨*options*⟩`]{`⟨*filename*⟩`}`

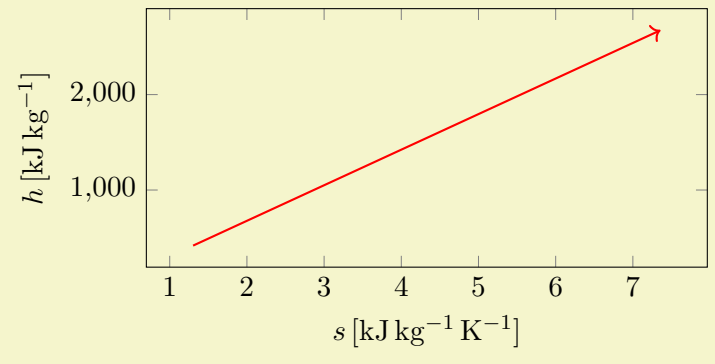
Export the selected curve network for PH, PV, TS, HS or PT. Options, defaults, units, curve selection, bounds and sampling are those of `/luacoolprop/diagram/`⟨*type*⟩, exactly as for `\LCPDiagram`. No surrounding axis is required. Drawing styles and label options do not add labels to the table. The following example exports the water saturation branches and draws the resulting file using ordinary PGFPlots; the source and result are both produced during this manual build.




```
\LCPExportDiagramTable{TS}[
  fluid=Water,quality values={0,1},labels=false,
  temperature axis unit=celsius
]{build/manual-tables/water-ts.dat}
\begin{tikzpicture}
  \begin{axis}[
    width=9cm,height=6cm,
    xlabel={$s\,[\mathrm{kJ\,kg^{-1}\,K^{-1}}]$},
    ylabel={$\theta\,[{}^\circ\mathrm{C}]$},
    unbounded coords=jump
  ]
    \addplot[blue,thick] table[x=x,y=y]
      {build/manual-tables/water-ts.dat};
  \end{axis}
\end{tikzpicture}
```

`\LCPExportProcessTable{`⟨*type*⟩`}[`⟨*options*⟩`]{`⟨*filename*⟩`}`

Export one process path with the keys and defaults of `/luacoolprop/process/`⟨*type*⟩. Its manifest also contains the completed `from` and `to` states in SI, the conserved quantity, reference state and process identifier. The command writes data only: it does not draw a path, define endpoint macros or emit coordinate log messages, even if rendering-only keys such as `export coordinates` or `log coordinates` are supplied. Use the ordinary process drawing command when those TeX side effects are wanted.




```
\LCPExportProcessTable{HS}[
  fluid=Water,type=isobar,
  from={pressure=1bar,quality=0},to={quality=1}
]{build/manual-tables/water-vaporization.dat}
\begin{tikzpicture}
  \begin{axis}[
    width=9cm,height=5cm,
    xlabel={$s\,[\mathrm{kJ\,kg^{-1}\,K^{-1}}]$},
    ylabel={$h\,[\mathrm{kJ\,kg^{-1}}]$},
    unbounded coords=jump
  ]
    \addplot[red,thick,->] table[x=x,y=y]
      {build/manual-tables/water-vaporization.dat};
  \end{axis}
\end{tikzpicture}
```

23.1 Columns, units and disconnected domains

The table columns are `curve`, `segment`, `x`, `y`, `x_si`, `y_si`, `p_Pa`, `T_K`, `h_Jkg`, `s_JkgK`, `v_m3kg` and `Q`. Curve indices and segment indices start at one; their correspondence to family names and values is recorded in the manifest. Each constant curve value has an explicit SI `value_unit`; process metadata likewise supplies `constant_unit`. The `x,y` columns use the requested coordinate scaling, including temperature offsets. They contain physical coordinates, never their logarithms: choose `xmode` and `ymode` in PGFPlots as usual. Default PH and PT plots have logarithmic pressure, PV has logarithmic volume and pressure, and TS/HS are linear. The `x_si,y_si` columns carry the same axis quantities in SI.

Additional state-property columns contain only values already available from the sampler; unavailable properties are `nan`. Export does not perform extra property evaluations to populate them. In particular, quality has no meaning outside the two-phase region or at the critical point, and a missing quality is not replaced by zero. A numerical value in a table is not an uncertainty estimate.

At a domain break, and between separate curves, a row with indices zero and `nan` coordinates prevents an artificial connecting segment. Always use `unbounded coords=jump`. Keeping only finite rows would destroy this information. To style each curve separately, select its `curve` index with PGFPlots' `restrict expr to domain` and retain the jump policy. The exported network contains the same computed points as the drawing commands; PGFPlots may still clip some of them to the final visible axis.

All finite numbers use 17 significant decimal digits for binary64 round trips, independently of `coord digits`, which controls drawing/macro formatting. For process paths the manifest's endpoint states preserve the numerical values before that presentation rounding.

23.2 Provenance and file ownership

The first line begins with `# luacoolprop-manifest` followed by JSON. Its schema is `luacoolprop-table-1`. It records the diagram, fluid, reference convention, coordinate transformations and SI units, curve indices, segment counts and omitted-sample diagnostics, supplied options, selected Lua source/native binary fingerprints, CoolProp version/revision, and managed native configuration/reference changes. The transform convention is plot = (SI − offset) scale. Machine-specific library and cache paths are omitted. Environment-based CoolProp settings are included, so review the manifest before sharing it. Custom raw-FFI mutations and external backend data are not certified by these fingerprints; record such dependencies independently.

Filenames are explicit, relative to the compiler's working directory unless absolute. Missing parent directories are created. Repeated calls replace the named file, so choose a generated-output directory and avoid source filenames. All computations and serialization finish before publication. A temporary file in the same directory is renamed atomically; on systems where replacement is unavailable, identical existing content is accepted, otherwise an error leaves the old file intact. Concurrent writers must not use one filename for different figures. No table is automatically imported back into LuaCoolProp, refreshed, or removed. Regenerate exports deliberately when inputs change.

For an offline document, retain only the PGFPlots drawing code and the table; do not call either export command or load LuaCoolProp. Such a consumer needs neither CoolProp nor shell escape. Axis labels, styles, annotations and numerical calculations for surrounding text remain the consumer's responsibility.

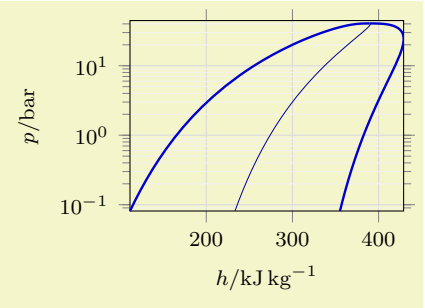
24 The generic diagram API

Use this API in libraries or in documents which may later switch diagram coordinates. The diagram identifier is case-sensitive and uppercase.

`\LCPDiagram{`⟨*type*⟩`}[`⟨*options*⟩`]`

Creates a complete TikZ picture and PGFPlots axis for ⟨*type*⟩.

The generic PH renderer

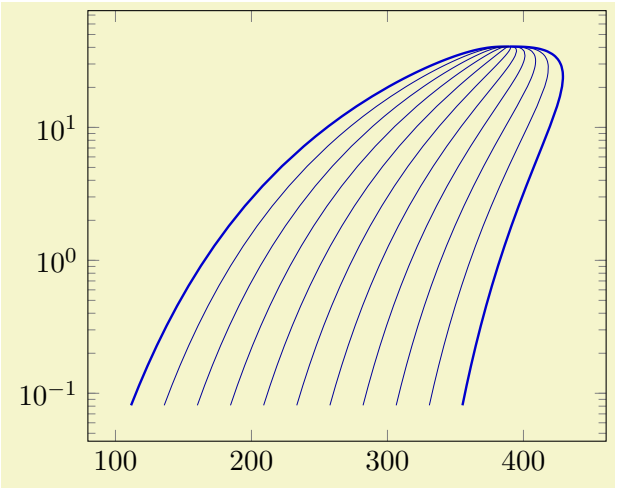



```
\LCPDiagram{PH}[
  fluid=R134a,
  quality values={0,0.5,1},
  axis options={width=5.2cm,height=4.1cm,
    font=\scriptsize}
]
```

`\LCPAddDiagramPlots{`⟨*type*⟩`}[`⟨*options*⟩`]`

Adds all enabled curve families to an existing axis. It does not create the axis, so the caller controls its coordinates and styling.

Add PH plots to an existing axis

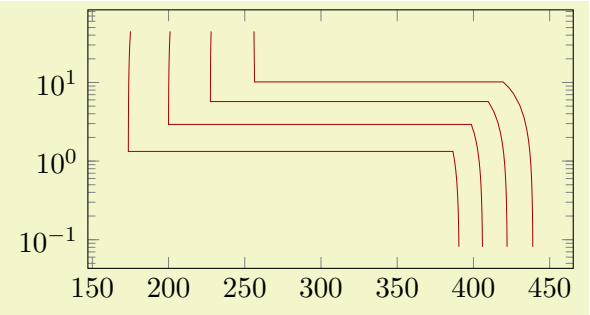



```
\begin{tikzpicture}
\begin{axis}[ymode=log]
  \LCPAddDiagramPlots{PH}[fluid=R134a]
\end{axis}
\end{tikzpicture}
```

\LCPAddDiagramFamily{⟨*type*⟩}{⟨*family*⟩}[⟨*options*⟩]

Adds exactly one family to an existing axis. PH registers `quality`, `isotherm`, `isentrope`, and `isochore`; PV registers `quality`, `isotherm`, and `isentrope`; TS registers `quality` and `isenthalp`; HS registers `quality`, `isochore`, `isotherm`, and `isobar`; PT registers `phase_envelope`, `isentrope`, `isenthalp`, and `isochore`.

Add only PH isotherms

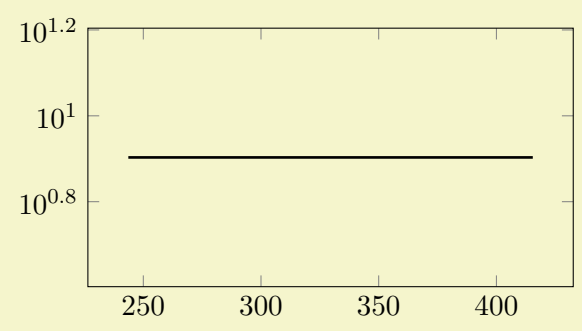



```
\begin{tikzpicture}
\begin{axis}[width=8cm,height=5cm,auto node placement,ymode=log]
\LCPAddDiagramFamily{PH}{isotherm}[
  temperature values={-20,0,20,40}]
\end{axis}
\end{tikzpicture}
```

\LCPAddDiagramProcess{⟨*type*⟩}[⟨*options*⟩]

Adds one thermodynamic process path to an existing axis.

Add an isobaric PH process




```
\begin{tikzpicture}
\begin{axis}[width=8cm,height=5cm,auto node placement,ymin=4,ymax=16,ymode=log]
\LCPAddDiagramProcess{PH}[
  type=isobar,pressure=8bar,
  from={quality=0},to={quality=1}]
\end{axis}
\end{tikzpicture}
```

24.1 A symmetric capability contract

Each registered type declares its coordinate properties and background families. The dispatcher validates those declarations before it performs any CoolProp calculation; PH is one implementation in this registry, not the base class of PV, TS, HS, or PT.

Type	x	y	quality	other background families
PH	h	p	yes	isotherm, isentrope, isochore
PV	v	p	yes	isotherm, isentrope
TS	s	T	yes	isenthalp
HS	s	h	yes	isochore, isotherm, isobar
PT	T	p	no	phase envelope, isentrope, isenthalp, isochore

An explicit switch for a family absent from this table is a hard error, not a silently ignored option. For example, `isochore=true` is invalid for a PV background, `isotherm=true` is invalid for TS, `isentrope=true` is invalid for HS, and `quality=true` is invalid for PT because quality is degenerate in PT coordinates. A conserved-property process is a different object: all six process kinds may be projected on every diagram even when the corresponding background family is unavailable.

24.2 Pure-fluid and reference-state contracts

The diagram API accepts only identifiers for which CoolProp reports `pure=true`. Explicit mixtures, predefined `.mix` files, and blend names such as `R407C` are rejected before sampling. Diagram construction assumes one triple point, one critical point, and one saturation dome; mixtures need phase-envelope and flash logic which this release does not claim to implement. Lua's low-level CoolProp wrappers remain available for deliberate mixture calculations. See CoolProp's mixture restrictions at https://coolprop.org/fluid_properties/Mixtures.html.

Managed diagrams and reference conventions use CoolProp's HEOS backend. An unprefixed fluid name and its explicit `HEOS::` form select the same model. Other backends remain accessible through the low-level Lua interface, but are rejected by the managed diagram and reference-state interfaces: changing a HEOS reference would not establish the reference of another model.

Specific enthalpy and entropy are relative properties. Their numerical origins depend on `reference state`; pressure, temperature, density, specific volume, and enthalpy differences

are unaffected by choosing another origin. Set one convention during initialization and use it throughout every diagram, exported state, table, and calculation which is compared. Energy balances should normally use differences such as $h_2 - h_1$, never infer a physical heat or work transfer from an isolated absolute value of h. The conventions and initialization rule follow https://coolprop.org/coolprop/HighLevelAPI.html#reference-states.

The convention is locked for the native fluid, including all its aliases. For example, `R134a`, `R134A`, and `HEOS::R134a` share one selection: after any diagram or process has used that fluid, requesting a different convention through another spelling raises an error before any native reference is changed. Repeating the selected convention is harmless. Exported process and table metadata report this same selection, whether the values were freshly calculated or read from the numerical cache. This lock lasts for the compiler process; TeX grouping does not release it.

`/luacoolprop/diagram/common/reference state`=⟨*DEF, IIR, ASHRAE, or NBP*⟩(no default)

Default: `DEF`.

Selects and then locks the CoolProp enthalpy/entropy origin for the diagram fluid. A per-diagram value must agree with every earlier calculation for that fluid.

Example: `reference state=IIR`.

24.3 Disconnected thermodynamic domains

Adaptive sampling never joins two valid regions across a state rejected by CoolProp. Every connected component is serialized separately and PGFPlots receives `unbounded coords=jump`; labels use one connected component instead of interpolating through the gap. This matters near equation-of-state limits and for conserved-property curves which leave and later re-enter a valid domain.

`/luacoolprop/diagram/common/domain policy`=⟨*ignore, warning, or error*⟩ (no default)

Default: `ignore`.

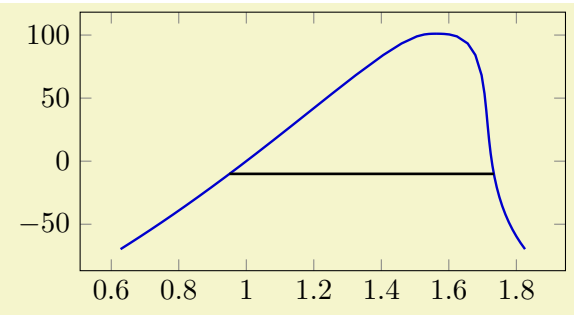
Controls the diagnostic when samples are omitted. `ignore` keeps valid components silently, `warning` reports the number omitted and retained, and `error` stops the build. `silent` and `warn` are accepted Lua compatibility spellings.

Example: `domain policy=warning`.

Accepted alias: `discontinuity policy` forwards to `domain policy`.

The same `reference state` and `domain policy` keys belong to every process namespace. Its canonical path is `/luacoolprop/process/TYPE`. Process metadata records the active reference convention; endpoint exports always retain raw SI properties independently of the plotted coordinate units. Each process namespace also accepts the semantic axis-unit keys belonging to its projection, so a manual axis can apply one choice consistently to both backgrounds and paths. For example:

A Celsius TS background and process



```
\begin{tikzpicture}
\begin{axis}[width=8cm,height=5cm,auto node placement,]
\LCPSetup{temperature axis unit=celsius}
\LCPAddTSQuality[quality values={0,1}]
\LCPAddTSProcess[type=isobar,pressure=2bar,
  from={quality=0},to={quality=1}]
\end{axis}
\end{tikzpicture}
```

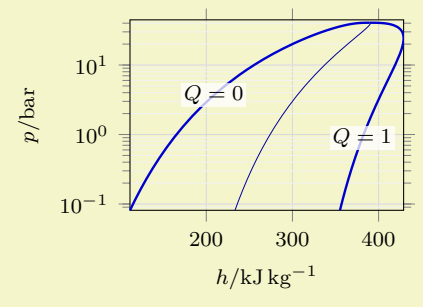
25 PH convenience commands

The following commands are stable conveniences for the current PH diagram. Their options use `/luacoolprop/diagram/PH`; process options use `/luacoolprop/process/PH`.

`\LCPPHDiagram`[⟨*options*⟩]

Equivalent to `\LCPDiagram{PH}`. It creates the picture, axis, automatic PH ranges, and all enabled families.

A compact labelled PH diagram

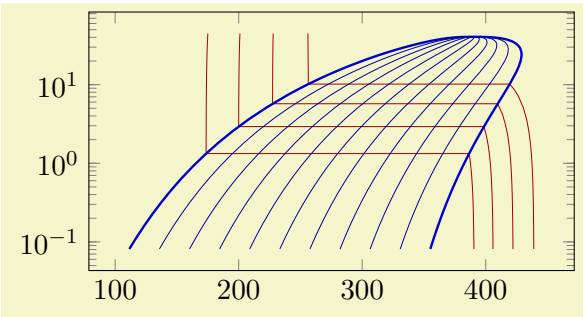



```
\LCPPHDiagram[
  fluid=R134a,
  quality values={0,0.5,1},
  labels=true,
  axis options={width=5.2cm,height=4.1cm,
    font=\scriptsize}
]
```

`\LCPAddPHPlots`[⟨*options*⟩]

Adds all enabled PH families inside an existing axis.

Quality dome plus isotherms

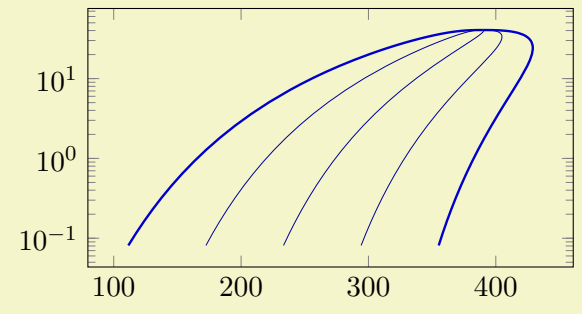



```
\begin{tikzpicture}
\begin{axis}[width=8cm,height=5cm,auto node placement,ymode=log]
\LCPAddPHPlots[quality=true,isotherm=true,
  temperature values={-20,0,20,40}]
\end{axis}
\end{tikzpicture}
```

`\LCPAddPHQuality`[⟨*options*⟩]

Adds only the quality family, including the saturated-liquid and saturated-vapour boundaries.

Five quality curves

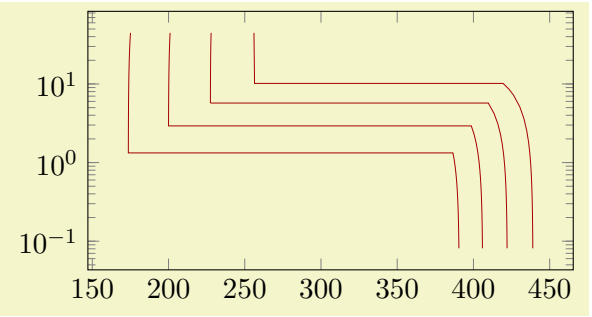



```
\begin{tikzpicture}
\begin{axis}[width=8cm,height=5cm,auto node placement,ymode=log]
\LCPAddPHQuality[quality values={0,0.25,0.5,0.75,1}]
\end{axis}
\end{tikzpicture}
```

\LCPAddPHIsotherms[⟨*options*⟩]

Adds only constant-temperature curves.

Explicit Celsius temperatures

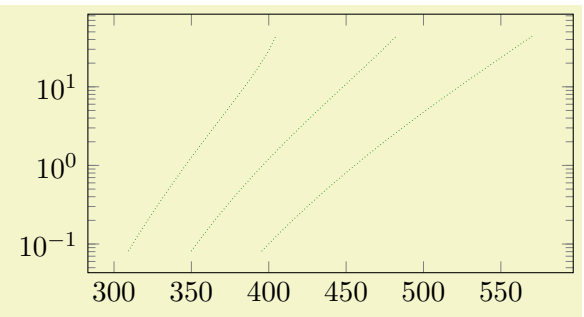



```
\begin{tikzpicture}
\begin{axis}[width=8cm,height=5cm,auto node placement,ymode=log]
\LCPAddPHIsotherms[temperature values={-20,0,20,40}]
\end{axis}
\end{tikzpicture}
```

\LCPAddPHIsentropes[⟨*options*⟩]

Adds only constant-specific-entropy curves.

Explicit entropy values

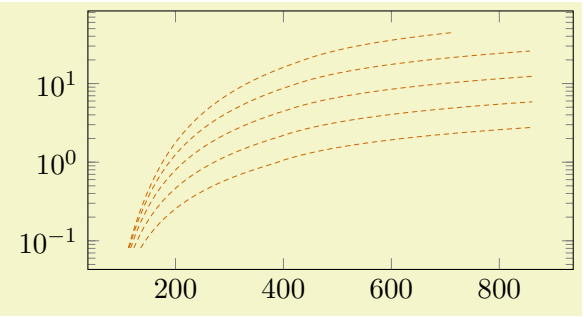


```
\begin{tikzpicture}
\begin{axis}[width=8cm,height=5cm,auto node placement,ymode=log]
\LCPAddPHIsentropes[entropy values={1.6,1.8,2.0}]
\end{axis}
\end{tikzpicture}
```

`\LCPAddPHIsochores`[⟨*options*⟩]

Adds only constant-specific-volume curves.

Logarithmic specific-volume grid

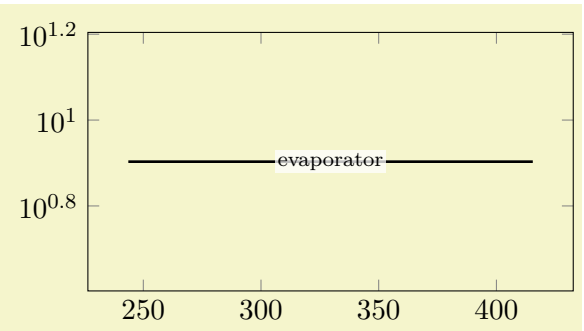



```
\begin{tikzpicture}
\begin{axis}[width=8cm,height=5cm,auto node placement,ymode=log]
\LCPAddPHIsochores[specific volume min=0.01,
  specific volume max=0.2,specific volume count=5]
\end{axis}
\end{tikzpicture}
```

`\LCPAddPHProcess`[⟨*options*⟩]

Adds a PH process path. See the complete process-key reference below.

Named evaporation process




```
\begin{tikzpicture}
\begin{axis}[width=8cm,height=5cm,auto node placement,ymin=4,ymax=16,ymode=log]
\LCPAddPHProcess[type=isobar,pressure=8bar,
  from={quality=0},to={quality=1},
  name=evaporation,label={evaporator}]
\end{axis}
\end{tikzpicture}
```

25.1 Alternate command names

The uppercase PH spellings above are canonical.

Accepted alias: `\LCPPhDiagram`.
Target: `\LCPPHDiagram`.
Example: `\LCPPhDiagram[fluid=R134a]`.

Accepted alias: `\LCPAddPHIsoQuality`.
Target: `\LCPAddPHQuality`.
Example: `\LCPAddPHIsoQuality[fluid=R134a]`.

Accepted alias: `\LCPAddPhIsoQuality`.
Target: `\LCPAddPHQuality`.
Example: `\LCPAddPhIsoQuality[fluid=R134a]`.

Accepted alias: `\LCPAddPhIsotherms`.
Target: `\LCPAddPHIsotherms`.
Example: `\LCPAddPhIsotherms[fluid=R134a]`.

Accepted alias: `\LCPAddPhIsentropes`.
Target: `\LCPAddPHIsentropes`.
Example: `\LCPAddPhIsentropes[fluid=R134a]`.

Accepted alias: `\LCPAddPhIsochores`.
Target: `\LCPAddPHIsochores`.
Example: `\LCPAddPhIsochores[fluid=R134a]`.

Accepted alias: `\LCPAddPhPlots`.
Target: `\LCPAddPHPlots`.
Example: `\LCPAddPhPlots[fluid=R134a]`.

Accepted alias: `\LCPAddPhProcess`.
Target: `\LCPAddPHProcess`.
Example: `\LCPAddPhProcess[fluid=R134a]`.

Accepted alias: `\LCPPhIsoQualityDiagram`.
Target: `\LCPPHDiagram`.
Example: `\LCPPhIsoQualityDiagram[fluid=R134a]`.

26 Shared diagram controls and PH axis units

Shared keys are defined below `/luacoolprop/diagram/common`; the PH axis-unit choices are defined below `/luacoolprop/diagram/PH`. Each reference entry shows its defining path. In a command option list, write only the leaf name, for example `fluid=Propane`. Other projections inherit shared controls and document their specific overrides below.

26.1 Fluid, families, ranges, and axis

`/luacoolprop/diagram/common/fluid`=⟨*CoolProp identifier*⟩ (no default)

Default: `global fluid`.

Selects the fluid for this diagram only.

Example: `fluid=Propane`.

`/luacoolprop/diagram/common/library`=⟨*file path*⟩ (no default)

Default: `global library`.

Selects the library for this request before the first native load. It cannot switch an already loaded library; use a fresh compiler process.

Example: `library=/opt/coolprop/libCoolProp.dylib`.

/luacoolprop/diagram/common/quality=⟨*boolean*⟩ (no default)

Default: `true`.

Enables or disables the quality family.

Example: `quality=false`.

/luacoolprop/diagram/common/isotherm=⟨*boolean*⟩ (no default)

Default: `false`.

Enables or disables the isotherm family.

Example: `isotherm=true`.

/luacoolprop/diagram/common/isentrope=⟨*boolean*⟩ (no default)

Default: `false`.

Enables or disables the isentrope family.

Example: `isentrope=true`.

/luacoolprop/diagram/common/isochore=⟨*boolean*⟩ (no default)

Default: `false`.

Enables or disables the isochore family.

Example: `isochore=true`.

/luacoolprop/diagram/common/pressure min=⟨*number or auto*⟩ (no default)

Default: `auto`.

Sets the lower sampling pressure in pascals.

Example: `pressure min=50000`.

/luacoolprop/diagram/common/pressure max=⟨*number or auto*⟩ (no default)

Default: `auto`.

Sets the upper sampling pressure in pascals.

Example: `pressure max=5E6`.

/luacoolprop/diagram/common/pressure max factor=⟨*positive number*⟩ (no default)

Default: `1.10`.

When the maximum is automatic, multiplies the critical pressure to leave headroom above the critical point.

Example: `pressure max factor=1.05`.

/luacoolprop/diagram/common/enthalpy scale=⟨*number*⟩ (no default)

Default: `global scale`.

Advanced raw multiplier for SI enthalpy. It selects the explicit scaled-SI label rather than claiming a named unit.

Example: `enthalpy scale=0.002`.

/luacoolprop/diagram/common/pressure scale=⟨*number*⟩ (no default)

Default: `global scale`.

Advanced raw multiplier for SI pressure. It selects the explicit scaled-SI label rather than claiming a named unit.

Example: `pressure scale=0.00002`.

`/luacoolprop/diagram/PH/enthalpy axis unit`=⟨*jkg or kjkg*⟩ (no default)

Default: `kjkg`.

Atomically selects the enthalpy multiplier and horizontal-axis unit.

Example: `enthalpy axis unit=jkg`.

`/luacoolprop/diagram/PH/pressure axis unit`=⟨*pa, kpa, mpa, or bar*⟩ (no default)

Default: `bar`.

Atomically selects the pressure multiplier and vertical-axis unit.

Example: `pressure axis unit=mpa`.

`/luacoolprop/diagram/common/enthalpy symbol`=⟨*TeX math material*⟩ (no default)

Default: `h`.

Sets the enthalpy symbol in a generated enthalpy axis. Omit surrounding math delimiters.

Example: `enthalpy symbol={\mathcal H}`.

`/luacoolprop/diagram/common/pressure symbol`=⟨*TeX math material*⟩ (no default)

Default: `p`.

Sets the pressure symbol in a generated pressure axis. Omit surrounding math delimiters.

Example: `pressure symbol={\pi }`.

`/luacoolprop/diagram/common/axis options`=⟨*PGFPlots options*⟩ (no default)

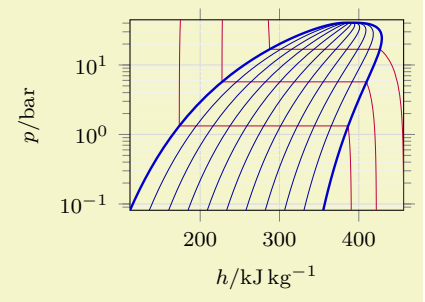
Default: `empty`.

Appends options to the axis created by `\LCPDiagram`. Width, height, labels, limits, legend placement, and fonts may all be overridden.

Example: `axis options={width=10cm,height=7cm,title={R134a}}`.

Accepted alias: `iso quality` forwards to `quality`.
Accepted alias: `isoquality` forwards to `quality`.
Accepted alias: `quality curves` forwards to `quality`.
Accepted alias: `isotherms` forwards to `isotherm`.
Accepted alias: `isentropes` forwards to `isentrope`.
Accepted alias: `isochores` forwards to `isochore`.
Accepted alias: `p min` forwards to `pressure min`.
Accepted alias: `p max` forwards to `pressure max`.
Accepted alias: `p max factor` forwards to `pressure max factor`.
Accepted alias: `h scale` forwards to `enthalpy scale`.
Accepted alias: `p scale` forwards to `pressure scale`.

Choosing families and appearance




```
\LCPPHDiagram[
  quality=true,isotherm=true,
  isentrope=false,isochore=false,
  temperature values={-20,20,60},
  isotherm color=purple,
  axis options={width=5.2cm,height=4.1cm,
    font=\scriptsize}
]
```

26.2 Quality grid

Quality is the vapour mass fraction: $q = 0$ is saturated liquid and $q = 1$ is saturated vapour. Use either an explicit list, a preset, a count, or a minimum/maximum/step description. An explicit list has highest practical clarity in archival documents.

`/luacoolprop/diagram/common/quality min=`⟨*number*⟩ (no default)

Default: `0`.

Lower quality in a generated grid.

Example: `quality min=0.1`.

`/luacoolprop/diagram/common/quality max=`⟨*number*⟩ (no default)

Default: `1`.

Upper quality in a generated grid.

Example: `quality max=0.9`.

`/luacoolprop/diagram/common/quality step=`⟨*positive number*⟩ (no default)

Default: `0.1`.

Increment used by a linear grid.

Example: `quality step=0.2`.

`/luacoolprop/diagram/common/quality count=`⟨*integer or auto*⟩ (no default)

Default: `auto`.

Requests an evenly distributed number of values.

Example: `quality count=6`.

`/luacoolprop/diagram/common/quality values=`⟨*comma list*⟩ (no default)

Default: `empty`.

Supplies the exact qualities and overrides generated-grid controls.

Example: `quality values={0,0.25,0.5,0.75,1}`.

`/luacoolprop/diagram/common/quality mode=`⟨*choice*⟩ (no default)

Default: `linear`.

Selects how grid values are obtained.

`linear` Uniform spacing between the bounds.

`log` Logarithmic spacing; meaningful only for positive bounds.

`logarithmic` Synonym for `log`.

`list` Use `quality values`.

`explicit` Synonym for `list`.

`preset` Use the selected preset.

`predefined` Synonym for `preset`.

`auto` Use a known preset, otherwise fall back to generated spacing.

Example: `quality mode=linear`.

`/luacoolprop/diagram/common/quality preset=`⟨*name*⟩ (no default)

Default: `default`.

Selects a named grid when `quality mode` is `preset`, `predefined`, or `auto`.

`default` Qualities from 0 to 1 in steps of 0.1.

`dense` The same eleven-curve grid as `default`.

`sparse` The five values 0, 0.25, 0.5, 0.75, and 1.

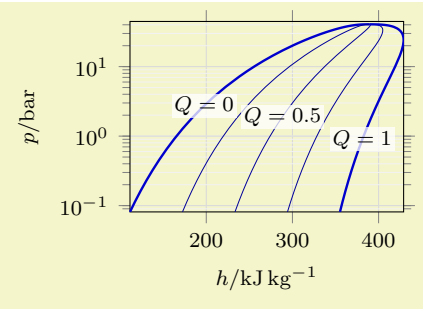
`boundary` Only saturated liquid and saturated vapour.

`boundaries` Synonym for `boundary`.

Example: `quality mode=preset,quality preset=sparse`.

Accepted alias: `q min` forwards to `quality min`.
Accepted alias: `q max` forwards to `quality max`.
Accepted alias: `q step` forwards to `quality step`.
Accepted alias: `q mode` forwards to `quality mode`.
Accepted alias: `q count` forwards to `quality count`.
Accepted alias: `q values` forwards to `quality values`.
Accepted alias: `q preset` forwards to `quality preset`.

An explicit quality grid




```
\LCPPHDiagram[
  quality values={0,0.25,0.5,0.75,1},
  labels=true,quality label every=2,
  axis options={width=5.2cm,height=4.1cm,
    font=\scriptsize}
]
```

26.3 Temperature, entropy, and specific-volume grids

All three grid families share the same pattern: `min`, `max`, `step`, `mode`, `count`, `preset`, and `values`. Explicit `values` take precedence.

26.3.1 Temperature and isotherms

`/luacoolprop/diagram/common/temperature unit=`⟨*unit*⟩ (no default)

Default: `celsius`.

Interprets all temperature-grid values. Use `celsius` (aliases `c` and `degc`) or `kelvin` (alias `k`).

Example: `temperature unit=kelvin`.

`/luacoolprop/diagram/common/temperature min=`⟨*number or auto*⟩ (no default)

Default: `auto`.

Lower temperature for the generated grid.

Example: `temperature min=-40`.

`/luacoolprop/diagram/common/temperature max=`⟨*number or auto*⟩ (no default)

Default: `auto`.

Upper temperature for the generated grid.

Example: `temperature max=80`.

/luacoolprop/diagram/common/temperature step=⟨*positive number*⟩ (no default)

Default: `10`.

Linear-grid increment in the selected unit.

Example: `temperature step=20`.

/luacoolprop/diagram/common/temperature count=⟨*integer or auto*⟩ (no default)

Default: `auto`.

Requests a fixed number of temperatures.

Example: `temperature count=7`.

/luacoolprop/diagram/common/temperature values=⟨*comma list*⟩ (no default)

Default: `empty`.

Uses exactly the listed temperatures.

Example: `temperature values={-40,-20,0,20,40,60}`.

/luacoolprop/diagram/common/temperature mode=⟨*choice*⟩ (no default)

Default: `linear`.

Selects `linear`, `log` (or `logarithmic`), `list` (or `explicit`), `preset` (or `predefined`), or `auto`. Explicit values always force list mode. Presets are used only by `preset`, `predefined`, or `auto`.

Example: `temperature mode=preset`.

/luacoolprop/diagram/common/temperature preset=⟨*name*⟩ (no default)

Default: `default`.

Available names are `default`, `refrigeration`, `sparse`, and `dense`. Refrigeration supplies Celsius values from -40 to 120 in 20-degree steps; sparse uses $-40, 0, 40, 80, 120$; dense uses 10-degree steps.

Example: `temperature mode=preset,temperature preset=refrigeration`.

Accepted alias: `t min` forwards to `temperature min`.
Accepted alias: `t max` forwards to `temperature max`.
Accepted alias: `t step` forwards to `temperature step`.
Accepted alias: `t mode` forwards to `temperature mode`.
Accepted alias: `t count` forwards to `temperature count`.
Accepted alias: `t preset` forwards to `temperature preset`.
Accepted alias: `t values` forwards to `temperature values`.

26.3.2 Specific entropy and isentropes

/luacoolprop/diagram/common/entropy unit=⟨*unit*⟩ (no default)

Default: `kjkgk`.

Interprets entropy values. Use `kjkgk` for $\mathrm{kJ\,kg^{-1}\,K^{-1}}$, or `si`, `jkgk`, or `j/kg/k` for $\mathrm{J\,kg^{-1}\,K^{-1}}$.

Example: `entropy unit=si`.

/luacoolprop/diagram/common/entropy min=⟨*number or auto*⟩ (no default)

Default: `auto`.

Lower entropy for the generated grid.

Example: `entropy min=1.4`.

/luacoolprop/diagram/common/entropy max=⟨*number or auto*⟩ (no default)

Default: `auto`.

Upper entropy for the generated grid.

Example: `entropy max=2.2`.

/luacoolprop/diagram/common/entropy step=⟨*positive number*⟩ (no default)

Default: `0.1`.

Linear-grid entropy increment.

Example: `entropy step=0.2`.

/luacoolprop/diagram/common/entropy count=⟨*integer or auto*⟩ (no default)

Default: `auto`.

Requests a fixed number of entropy values.

Example: `entropy count=6`.

/luacoolprop/diagram/common/entropy values=⟨*comma list*⟩ (no default)

Default: `empty`.

Uses exactly the listed entropy values.

Example: `entropy values={1.4,1.6,1.8,2.0,2.2}`.

/luacoolprop/diagram/common/entropy mode=⟨*choice*⟩ (no default)

Default: `linear`.

Supports `linear`, `log`/`logarithmic`, `list`/`explicit`, `preset`/`predefined`, and `auto`, with the same precedence rules as the temperature grid.

Example: `entropy mode=linear`.

/luacoolprop/diagram/common/entropy preset=⟨*name*⟩ (no default)

Default: `default`.

Available names are `default`, `refrigeration`, `sparse`, and `dense`; values are converted according to `entropy unit`.

Example: `entropy mode=preset,entropy preset=dense`.

Accepted alias: `s min` forwards to `entropy min`.
Accepted alias: `s max` forwards to `entropy max`.
Accepted alias: `s step` forwards to `entropy step`.
Accepted alias: `s mode` forwards to `entropy mode`.
Accepted alias: `s count` forwards to `entropy count`.
Accepted alias: `s preset` forwards to `entropy preset`.
Accepted alias: `s values` forwards to `entropy values`.

26.3.3 Specific volume and isochores

/luacoolprop/diagram/common/specific volume unit=⟨*unit*⟩ (no default)

Default: `m3kg`.

Interprets volume values. Use `m3kg` for $\mathrm{m^3\,kg^{-1}}$; `lkg`, `l/kg`, `dm3kg`, and `dm3/kg` select litres per kilogram.

Example: `specific volume unit=lkg`.

`/luacoolprop/diagram/common/specific volume min=`⟨*number or auto*⟩ (no default)

Default: `auto`.

Lower specific volume for the generated grid.

Example: `specific volume min=0.01`.

`/luacoolprop/diagram/common/specific volume max=`⟨*number or auto*⟩ (no default)

Default: `auto`.

Upper specific volume for the generated grid.

Example: `specific volume max=0.2`.

`/luacoolprop/diagram/common/specific volume step=`⟨*number or auto*⟩ (no default)

Default: `auto`.

Increment for a linear grid.

Example: `specific volume step=0.02`.

`/luacoolprop/diagram/common/specific volume count=`⟨*integer*⟩ (no default)

Default: `8`.

Requests a fixed number of volume values.

Example: `specific volume count=6`.

`/luacoolprop/diagram/common/specific volume values=`⟨*comma list*⟩ (no default)

Default: `empty`.

Uses exactly the listed volumes.

Example: `specific volume values={0.01,0.02,0.05,0.1,0.2}`.

`/luacoolprop/diagram/common/specific volume mode=`⟨*choice*⟩ (no default)

Default: `log`.

Supports `linear`, `log`/`logarithmic`, `list`/`explicit`, `preset`/`predefined`, and `auto`. Logarithmic spacing is natural over wide volume ranges.

Example: `specific volume mode=log`.

`/luacoolprop/diagram/common/specific volume preset=`⟨*name*⟩ (no default)

Default: `default`.

Available names are `default`, `refrigeration`, `sparse`, and `dense`; values follow `specific volume unit`. Default and refrigeration use the same grid.

Example: `specific volume mode=preset,specific volume preset=sparse`.

Accepted alias: `volume unit` forwards to `specific volume unit`.

Accepted alias: `v unit` forwards to `specific volume unit`.

Accepted alias: `v min` forwards to `specific volume min`.

Accepted alias: `v max` forwards to `specific volume max`.

Accepted alias: `v step` forwards to `specific volume step`.

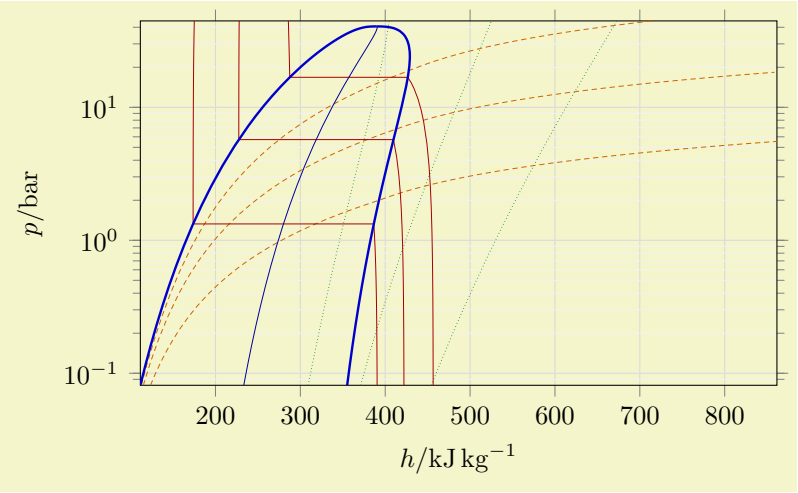
Accepted alias: `v mode` forwards to `specific volume mode`.

Accepted alias: `v count` forwards to `specific volume count`.

Accepted alias: `v preset` forwards to `specific volume preset`.

Accepted alias: `v values` forwards to `specific volume values`.

Combining all thermodynamic families




```
\LCPPHDiagram[
  quality values={0,0.5,1},
  isotherm=true,temperature values={-20,20,60},
  isentrope=true,entropy values={1.6,1.9,2.2},
  isochore=true,specific volume values={0.01,0.03,0.1},
  axis options={width=10cm,height=6.4cm,font=\small}
]
```

26.4 Adaptive sampling and numerical output

LuaCoolProp adaptively subdivides isolines. The general controls apply to the quality family; family-specific controls let expensive curves be tuned independently. Raise a tolerance or lower a maximum depth for faster drafts; lower a tolerance only when the visual improvement justifies the extra CoolProp calls.

Quality curves receive additional, automatic refinement close to the critical pressure. Saturated liquid ($Q = 0$), saturated vapour ($Q = 1$), and all intermediate qualities are evaluated at a geometric sequence of subcritical pressures. Because the CoolProp P, Q input pair is singular at the critical state, LuaCoolProp then appends one canonical limiting point evaluated from the fluid's T_c and ρ_c. Its metadata states `critical_limit=true` and `quality_defined=false`; no quality is assigned at the critical state. Consequently every subcritical quality curve approaches exactly the same plotted coordinate when the requested pressure range reaches p_c. This is a graphical limiting construction, not a claim that every quality exists at the critical state. These safeguards require no sampling key. The two saturation branches do *not* meet at the triple point: saturated liquid and saturated vapour are distinct there and meet only at the critical point.

`/luacoolprop/diagram/common/initial intervals`=⟨*positive integer*⟩ (no default)

Default: `18`.

Initial subdivision count for quality curves.

Example: `initial intervals=24`.

`/luacoolprop/diagram/common/max depth`=⟨*nonnegative integer*⟩ (no default)

Default: `8`.

Maximum recursive subdivision depth for quality curves.

Example: `max depth=10`.

/luacoolprop/diagram/common/tolerance=⟨*positive number*⟩ (no default)

Default: `0.25`.

Geometric refinement tolerance for quality curves.

Example: `tolerance=0.15`.

/luacoolprop/diagram/common/log weight=⟨*nonnegative number*⟩ (no default)

Default: `30`.

Weights pressure changes in logarithmic space during refinement.

Example: `log weight=40`.

/luacoolprop/diagram/common/coord digits=⟨*positive integer*⟩ (no default)

Default: `6`.

Decimal significant precision emitted to PGFPlots coordinates.

Example: `coord digits=8`.

/luacoolprop/diagram/common/isotherm initial intervals=⟨*positive integer*⟩ (no default)

Default: `14`.

Initial subdivisions for each isotherm.

Example: `isotherm initial intervals=18`.

/luacoolprop/diagram/common/isotherm max depth=⟨*nonnegative integer*⟩ (no default)

Default: `7`.

Maximum adaptive depth for isotherms.

Example: `isotherm max depth=9`.

/luacoolprop/diagram/common/isotherm tolerance=⟨*positive number*⟩ (no default)

Default: `0.35`.

Adaptive tolerance for isotherms.

Example: `isotherm tolerance=0.2`.

/luacoolprop/diagram/common/isentrope initial intervals=⟨*positive integer*⟩ (no default)

Default: `14`.

Initial subdivisions for each isentrope.

Example: `isentrope initial intervals=18`.

/luacoolprop/diagram/common/isentrope max depth=⟨*nonnegative integer*⟩ (no default)

Default: `7`.

Maximum adaptive depth for isentropes.

Example: `isentrope max depth=9`.

/luacoolprop/diagram/common/isentrope tolerance=⟨*positive number*⟩ (no default)

Default: `0.30`.

Adaptive tolerance for isentropes.

Example: `isentrope tolerance=0.2`.

/luacoolprop/diagram/common/isochore initial intervals=⟨*positive integer*⟩ (no default)

Default: `16`.

Initial subdivisions for each isochore.

Example: `isochore initial intervals=20`.

/luacoolprop/diagram/common/isochore max depth=⟨*nonnegative integer*⟩ (no default)

Default: `8`.

Maximum adaptive depth for isochores.

Example: `isochore max depth=10`.

/luacoolprop/diagram/common/isochore tolerance=⟨*positive number*⟩ (no default)

Default: `0.28`.

Adaptive tolerance for isochores.

Example: `isochore tolerance=0.18`.

/luacoolprop/diagram/common/saturation pressure epsilon=⟨*positive number*⟩(no default)

Default: `0.00001`.

Offsets evaluations from the exact saturation boundary to avoid ambiguous two-phase states. Change this only when diagnosing a backend edge case.

Example: `saturation pressure epsilon=0.00002`.

26.5 Curve styles, colours, and legend participation

Styles are ordinary TikZ/PGFPlots option lists. A family colour is combined with its family style; `curve style` is an additional common style.

/luacoolprop/diagram/common/curve style=⟨*TikZ style*⟩ (no default)

Default: `empty`.

Adds common drawing options to all emitted thermodynamic curves.

Example: `curve style={opacity=.8}`.

/luacoolprop/diagram/common/quality color=⟨*colour*⟩ (no default)

Default: `blue!60!black`.

Sets interior quality-curve colour.

Example: `quality color=cyan!60!black`.

/luacoolprop/diagram/common/quality boundary color=⟨*colour*⟩ (no default)

Default: `blue!80!black`.

Sets the saturated-liquid and saturated-vapour boundary colour.

Example: `quality boundary color=navy`.

/luacoolprop/diagram/common/isotherm color=⟨*colour*⟩ (no default)

Default: `red!65!black`.

Sets isotherm colour.

Example: `isotherm color=magenta!70!black`.

`/luacoolprop/diagram/common/isentrope color=`⟨*colour*⟩ (no default)

Default: `green!50!black`.

Sets isentrope colour.

Example: `isentrope color=green!50!black`.

`/luacoolprop/diagram/common/isochore color=`⟨*colour*⟩ (no default)

Default: `orange!80!black`.

Sets isochore colour.

Example: `isochore color=brown`.

`/luacoolprop/diagram/common/interior style=`⟨*TikZ style*⟩ (no default)

Default: `line width=0.25pt`.

Styles interior quality curves.

Example: `interior style={line width=.4pt,densely dashed}`.

`/luacoolprop/diagram/common/boundary style=`⟨*TikZ style*⟩ (no default)

Default: `line width=0.9pt`.

Styles the quality-envelope boundaries.

Example: `boundary style={line width=1.2pt}`.

`/luacoolprop/diagram/common/isotherm style=`⟨*TikZ style*⟩ (no default)

Default: `line width=0.25pt`.

Styles all isotherms.

Example: `isotherm style={line width=.35pt,dashed}`.

`/luacoolprop/diagram/common/isentrope style=`⟨*TikZ style*⟩ (no default)

Default: `dotted thin line`.

Styles all isentropes.

Example: `isentrope style={line width=.35pt,densely dotted}`.

`/luacoolprop/diagram/common/isochore style=`⟨*TikZ style*⟩ (no default)

Default: `short dashed thin line`.

Styles all isochores.

Example: `isochore style={line width=.35pt,dash dot}`.

`/luacoolprop/diagram/common/legend=`⟨*boolean*⟩ (no default)

Default: `false`.

Adds emitted curves to the PGFPlots legend when true.

Example: `legend=true`.

`/luacoolprop/diagram/common/forget plot=`⟨*boolean*⟩ (no default)

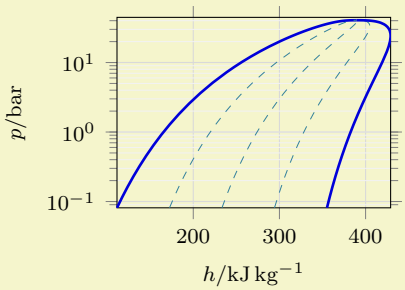
Default: `true`.

Applies PGFPlots `forget plot`; set false when constructing a custom legend.

Example: `forget plot=false`.

Accepted alias: `boundary color` forwards to `quality boundary color`.

A publication-oriented style




```
\LCPPHDiagram[
  quality values={0,0.25,0.5,0.75,1},
  quality color=cyan!55!black,
  quality boundary color=blue!85!black,
  interior style={line width=.35pt,dashed},
  boundary style={line width=1pt},
  axis options={width=5.2cm,height=4.1cm,
    font=\scriptsize}
]
```

26.6 Labels and family defaults

Setting `labels=true` requests labels from LuaCoolProp. Placement is then delegated to the required external `pgfplots-autonode` library. High-level diagram commands enable its hooks locally. A manually created axis using low-level family or process commands must include `auto node placement`; LuaCoolProp never appends that style to unrelated PGFPlots axes. The shared keys in this subsection prepare label text, visibility, preferred positions, and node styles; the autonomous library performs geometry and collision avoidance.

Symbols are arbitrary TeX math material and are independent of the physical quantity passed to CoolProp. Do not include dollar signs: LuaCoolProp supplies the surrounding math mode. Symbol settings affect the whole family, while `label text for` remains the more specific override for one named curve. They do not change stable curve identifiers: an isotherm still uses an `lcp-ph-T-...` name after displaying `temperature symbol=\theta`.

`/luacoolprop/diagram/common/quality symbol=`⟨*TeX math material*⟩ (no default)

Default: `Q`.

Sets the symbol preceding every generated quality value.

Example: `quality symbol=x`.

`/luacoolprop/diagram/common/temperature symbol=`⟨*TeX math material*⟩ (no default)

Default: `T`.

Sets the symbol preceding every generated isotherm value.

Example: `temperature symbol={\theta }`.

`/luacoolprop/diagram/common/entropy symbol=`⟨*TeX math material*⟩ (no default)

Default: `s`.

Sets the symbol preceding every generated isentrope value.

Example: `entropy symbol={\sigma }`.

`/luacoolprop/diagram/common/specific volume symbol=`⟨*TeX math material*⟩ (no default)

Default: `v`.

Sets the symbol preceding every generated isochore value.

Example: `specific volume symbol={\nu }`.

`/luacoolprop/diagram/common/labels=`⟨*boolean*⟩ (no default)

Default: `false`.

Enables labels for eligible curve families.

Example: `labels=true`.

`/luacoolprop/diagram/common/label placement`=⟨*autonode*⟩ (no default)

Default: `autonode`.

Selects autonomous placement. `autonode` is the supported placement engine.

Example: `label placement=autonode`.

`/luacoolprop/diagram/common/quality labels`=⟨*boolean or auto*⟩ (no default)

Default: `auto`.

Overrides global label visibility for quality curves.

Example: `quality labels=true`.

`/luacoolprop/diagram/common/isotherm labels`=⟨*boolean or auto*⟩ (no default)

Default: `auto`.

Overrides global label visibility for isotherms.

Example: `isotherm labels=true`.

`/luacoolprop/diagram/common/isentrope labels`=⟨*boolean or auto*⟩ (no default)

Default: `auto`.

Overrides global label visibility for isentropes.

Example: `isentrope labels=false`.

`/luacoolprop/diagram/common/isochore labels`=⟨*boolean or auto*⟩ (no default)

Default: `auto`.

Overrides global label visibility for isochores.

Example: `isochore labels=true`.

`/luacoolprop/diagram/common/label pos`=⟨*fraction*⟩ (no default)

Default: `0.35`.

Sets the common preferred curve position.

Example: `label pos=0.5`.

`/luacoolprop/diagram/common/quality label pos`=⟨*fraction*⟩ (no default)

Default: `0.35`.

Sets the quality-family preferred position.

Example: `quality label pos=0.3`.

`/luacoolprop/diagram/common/isotherm label pos`=⟨*fraction*⟩ (no default)

Default: `0.20`.

Sets the isotherm-family preferred position.

Example: `isotherm label pos=0.25`.

`/luacoolprop/diagram/common/isentrope label pos`=⟨*fraction*⟩ (no default)

Default: `0.45`.

Sets the isentrope-family preferred position.

Example: `isentrope label pos=0.5`.

`/luacoolprop/diagram/common/isochore label pos=`⟨*fraction*⟩ (no default)

Default: `0.58`.

Sets the isochore-family preferred position.

Example: `isochore label pos=0.6`.

`/luacoolprop/diagram/common/label every=`⟨*positive integer*⟩ (no default)

Default: `2`.

Labels one curve out of every n by default.

Example: `label every=3`.

`/luacoolprop/diagram/common/quality label every=`⟨*positive integer*⟩ (no default)

Default: `2`.

Sets the quality-family label stride.

Example: `quality label every=1`.

`/luacoolprop/diagram/common/isotherm label every=`⟨*positive integer*⟩ (no default)

Default: `2`.

Sets the isotherm-family label stride.

Example: `isotherm label every=3`.

`/luacoolprop/diagram/common/isentrope label every=`⟨*positive integer*⟩ (no default)

Default: `2`.

Sets the isentrope-family label stride.

Example: `isentrope label every=2`.

`/luacoolprop/diagram/common/isochore label every=`⟨*positive integer*⟩ (no default)

Default: `2`.

Sets the isochore-family label stride.

Example: `isochore label every=2`.

`/luacoolprop/diagram/common/label sloped=`⟨*boolean*⟩ (no default)

Default: `false`.

Sets the common label-rotation policy.

Example: `label sloped=true`.

`/luacoolprop/diagram/common/quality label sloped=`⟨*boolean or auto*⟩ (no default)

Default: `auto`.

Overrides rotation for quality labels.

Example: `quality label sloped=false`.

`/luacoolprop/diagram/common/isotherm label sloped=`⟨*boolean or auto*⟩ (no default)

Default: `auto`.

Overrides rotation for isotherm labels.

Example: `isotherm label sloped=true`.

/luacoolprop/diagram/common/isentrope label sloped=⟨*boolean or auto*⟩ (no default)

Default: `auto`.

Overrides rotation for isentrope labels.

Example: `isentrope label sloped=true`.

/luacoolprop/diagram/common/isochore label sloped=⟨*boolean or auto*⟩ (no default)

Default: `auto`.

Overrides rotation for isochore labels.

Example: `isochore label sloped=true`.

/luacoolprop/diagram/common/label allow upside down=⟨*boolean*⟩ (no default)

Default: `false`.

Sets the common PGFPlots orientation policy.

Example: `label allow upside down=true`.

/luacoolprop/diagram/common/quality label allow upside down=⟨*boolean or auto*⟩ (no default)

Default: `auto`.

Overrides upside-down orientation for quality labels.

Example: `quality label allow upside down=false`.

/luacoolprop/diagram/common/isotherm label allow upside down=⟨*boolean or auto*⟩ (no default)

Default: `auto`.

Overrides upside-down orientation for isotherm labels.

Example: `isotherm label allow upside down=false`.

/luacoolprop/diagram/common/isentrope label allow upside down=⟨*boolean or auto*⟩ (no default)

Default: `auto`.

Overrides upside-down orientation for isentrope labels.

Example: `isentrope label allow upside down=true`.

/luacoolprop/diagram/common/isochore label allow upside down=⟨*boolean or auto*⟩ (no default)

Default: `auto`.

Overrides upside-down orientation for isochore labels.

Example: `isochore label allow upside down=false`.

/luacoolprop/diagram/common/label style=⟨*TikZ style name*⟩ (no default)

Default: `luacoolprop label node`.

Sets the common label-node style name.

Example: `label style=luacoolprop label node`.

/luacoolprop/diagram/common/quality label style=⟨*TikZ style name*⟩ (no default)

Default: `luacoolprop quality label node`.

Sets the quality-label style name.

Example: `quality label style=luacoolprop quality label node`.

/luacoolprop/diagram/common/isotherm label style=⟨*TikZ style name*⟩ (no default)

Default: `luacoolprop isotherm label node`.

Sets the isotherm-label style name.

Example: `isotherm label style=luacoolprop isotherm label node`.

/luacoolprop/diagram/common/isentrope label style=⟨*TikZ style name*⟩ (no default)

Default: `luacoolprop isentrope label node`.

Sets the isentrope-label style name.

Example: `isentrope label style=luacoolprop isentrope label node`.

/luacoolprop/diagram/common/isochore label style=⟨*TikZ style name*⟩ (no default)

Default: `luacoolprop isochore label node`.

Sets the isochore-label style name.

Example: `isochore label style=luacoolprop isochore label node`.

The five `... label node style` keys redefine the corresponding TikZ style directly. They accept a complete node-option list.

/luacoolprop/diagram/common/label node style=⟨*TikZ node options*⟩ (no default)

Default: `package style`.

Redefines `luacoolprop label node`.

Example: `label node style={fill=yellow!15,inner sep=2pt}`.

/luacoolprop/diagram/common/quality label node style=⟨*TikZ node options*⟩ (no default)

Default: `inherits common style`.

Redefines the quality label node.

Example: `quality label node style={fill=blue!8,text=blue!70!black}`.

/luacoolprop/diagram/common/isotherm label node style=⟨*TikZ node options*⟩(no default)

Default: `inherits common style`.

Redefines the isotherm label node.

Example: `isotherm label node style={fill=red!8,text=red!70!black}`.

/luacoolprop/diagram/common/isentrope label node style=⟨*TikZ node options*⟩ (no default)

Default: `inherits common style`.

Redefines the isentrope label node.

Example: `isentrope label node style={fill=green!8,text=green!40!black}`.

/luacoolprop/diagram/common/isochore label node style=⟨*TikZ node options*⟩(no default)

Default: `inherits common style`.

Redefines the isochore label node.

Example: `isochore label node style={fill=orange!8,text=orange!70!black}`.

Accepted alias: `sloped labels` forwards to `label sloped`.

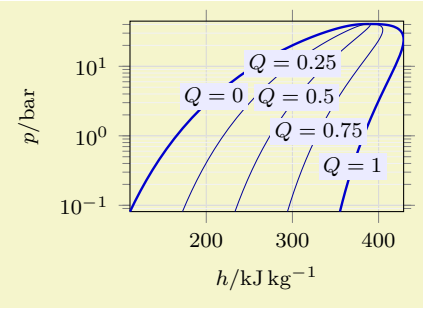
Accepted alias: `smart labels` forwards to `label placement=autonode`.

Accepted alias: `automatic labels` forwards to `label placement=autonode`.

Accepted alias: `autonode labels` forwards to `label placement=autonode`.

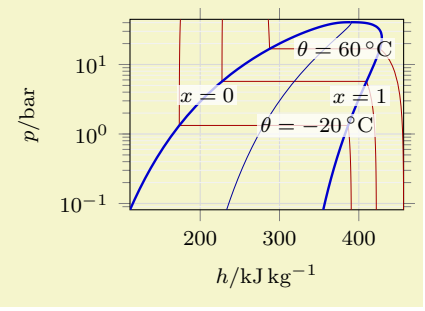
Accepted alias: `pgfplots autonode labels` forwards to `label placement=autonode`.

Family label controls




```
\LCPPHDiagram[
  quality values={0,0.25,0.5,0.75,1},
  labels=true,quality label every=1,
  quality label pos=.45,
  quality label node style={fill=blue!8,
    inner sep=1.5pt,font=\scriptsize},
  axis options={width=5.2cm,height=4.1cm,
    font=\scriptsize}
]
```

Alternative thermodynamic notation




```
\LCPPHDiagram[
  quality values={0,0.5,1},quality symbol=x,
  isotherm=true,temperature values={-20,20,60},
  temperature symbol={\theta},labels=true,
  label every=1,
  axis options={width=5.2cm,height=4.1cm,
    font=\scriptsize}
]
```

26.7 Per-curve overrides

Each override takes two braced arguments inside the key list: a curve selector and its value. Selectors are the stable curve names emitted by LuaCoolProp, for example `lcp-ph-q-0p5`, `lcp-ph-T-20C`, or a process `name`. Repeating a key is allowed and builds a list of overrides.

`/luacoolprop/diagram/common/label pos for=`⟨*selector and fraction*⟩ (no default)

Default: `none`.

Overrides one label's preferred position.

Example: `label pos for={lcp-ph-q-0p5}{0.62}`.

`/luacoolprop/diagram/common/label style for=`⟨*selector and style*⟩ (no default)

Default: `none`.

Overrides one label's TikZ style name or options.

Example: `label style for={lcp-ph-q-0p5}{fill=yellow}`.

`/luacoolprop/diagram/common/label text for=`⟨*selector and TeX text*⟩ (no default)

Default: `none`.

Replaces one automatically generated label.

Example: `label text for={lcp-ph-q-0p5}{$q=\qty {50}{\percent }$}`.

/luacoolprop/diagram/common/label show for=⟨*selector and boolean*⟩ (no default)

Default: `none`.

Forces one label to be shown or hidden.

Example: `label show for={lcp-ph-q-0p5}{false}`.

/luacoolprop/diagram/common/label fixed for=⟨*selector and boolean*⟩ (no default)

Default: `none`.

Marks one label position as fixed rather than preferred.

Example: `label fixed for={lcp-ph-q-0p5}{true}`.

/luacoolprop/diagram/common/label sloped for=⟨*selector and boolean*⟩ (no default)

Default: `none`.

Overrides rotation for one label.

Example: `label sloped for={lcp-ph-q-0p5}{true}`.

/luacoolprop/diagram/common/label allow upside down for=⟨*selector and boolean*⟩ (no default)

Default: `none`.

Overrides the orientation policy for one label.

Example: `label allow upside down for={lcp-ph-q-0p5}{false}`.

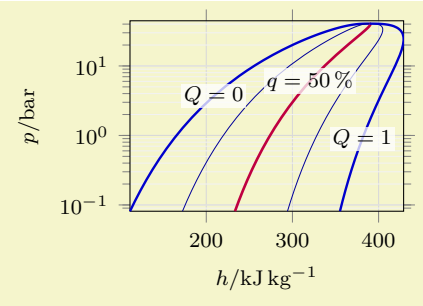
/luacoolprop/diagram/common/curve style for=⟨*selector and TikZ style*⟩ (no default)

Default: `none`.

Overrides the drawing style for one curve.

Example: `curve style for={lcp-ph-q-0p5}{purple,line width=1pt}`.

Highlight one named curve




```
\LCPPHDiagram[
  quality values={0,0.25,0.5,0.75,1},labels=true,
  label text for={lcp-ph-q-0p5}{$q=\qty{50}{\percent}$},
  label pos for={lcp-ph-q-0p5}{0.55},
  curve style for={lcp-ph-q-0p5}{purple,
    line width=1pt},
  axis options={width=5.2cm,height=4.1cm,
    font=\scriptsize}
]
```

26.8 Autonode options forwarded by PH diagrams

These keys configure the candidate set attached to labels emitted by LuaCoolProp. The next chapter documents the autonomous library itself and its axis-wide solver controls.

/luacoolprop/diagram/common/autonode candidates=⟨*positive integer*⟩ (no default)

Default: `51`.

Sets the number of sampled positions for each PH label.

Example: `autonode candidates=31`.

`/luacoolprop/diagram/common/autonode candidate strategy=`⟨*strategy*⟩ (no default)

Default: `around-preferred`.

Uses `uniform`, `around-preferred`, or `adaptive` candidate ordering.

Example: `autonode candidate strategy=adaptive`.

`/luacoolprop/diagram/common/autonode clearance=`⟨*TeX dimension*⟩ (no default)

Default: `1pt`.

Adds collision clearance around PH labels.

Example: `autonode clearance=2pt`.

`/luacoolprop/diagram/common/autonode inner sep=`⟨*TeX dimension*⟩ (no default)

Default: `1.5pt`.

Sets the label node's inner separation for both rendering and measurement; a later inner sep in the node style takes precedence.

Example: `autonode inner sep=2pt`.

`/luacoolprop/diagram/common/autonode normal shift=`⟨*TeX dimension*⟩ (no default)

Default: `0pt`.

Offsets PH labels along the local curve normal.

Example: `autonode normal shift=4pt`.

`/luacoolprop/diagram/common/autonode preferred weight=`⟨*number*⟩ (no default)

Default: `8`.

Weights distance from the preferred position.

Example: `autonode preferred weight=12`.

`/luacoolprop/diagram/common/autonode overlap weight=`⟨*number*⟩ (no default)

Default: `1000`.

Weights label overlap relative to preferred-position distance.

Example: `autonode overlap weight=2000`.

`/luacoolprop/diagram/common/label min pos=`⟨*fraction*⟩ (no default)

Default: `0`.

Sets the earliest candidate position along each PH curve.

Example: `label min pos=0.12`.

`/luacoolprop/diagram/common/label max pos=`⟨*fraction*⟩ (no default)

Default: `1`.

Sets the latest candidate position along each PH curve.

Example: `label max pos=0.88`.

26.9 Alternate label-control keys

The following accepted names do *not* configure the autonomous solver. The effective controls live under `/pgfplots` or `/pgfplots/autonode` and are documented in the next chapter.

/luacoolprop/diagram/common/label collision policy=⟨*value*⟩ (no default)

Default: `keep`.

Accepted without solver effect; use `auto node failure mode`.

Example: `label collision policy=keep`.

Accepted alias: `label collision` forwards to `label collision policy`.

/luacoolprop/diagram/common/label hide overlap threshold=⟨*number*⟩ (no default)

Default: `0.06`.

Accepted without solver effect; use the autonode failure policy.

Example: `label hide overlap threshold=0.08`.

/luacoolprop/diagram/common/label hide edge threshold=⟨*number*⟩ (no default)

Default: `0.025`.

Accepted without solver effect; use `auto node border margin`.

Example: `label hide edge threshold=0.03`.

/luacoolprop/diagram/common/label candidate count=⟨*integer*⟩ (no default)

Default: `21`.

Accepted without solver effect; use `autonode candidates`.

Example: `label candidate count=31`.

/luacoolprop/diagram/common/label separation=⟨*number*⟩ (no default)

Default: `0.018`.

Accepted without solver effect; use `autonode clearance`.

Example: `label separation=0.02`.

/luacoolprop/diagram/common/label overlap penalty=⟨*number*⟩ (no default)

Default: `1000`.

Accepted without solver effect; use `autonode overlap weight`.

Example: `label overlap penalty=1500`.

/luacoolprop/diagram/common/label distance penalty=⟨*number*⟩ (no default)

Default: `1`.

Accepted without solver effect; use `autonode preferred weight`.

Example: `label distance penalty=2`.

/luacoolprop/diagram/common/label edge penalty=⟨*number*⟩ (no default)

Default: `5`.

Accepted without solver effect; use `auto node border margin`.

Example: `label edge penalty=8`.

/luacoolprop/diagram/common/label box base width=⟨*number*⟩ (no default)

Default: `0.045`.

Accepted without solver effect; autonode measures the real TeX node.

Example: `label box base width=0.05`.

/luacoolprop/diagram/common/label box char width=⟨*number*⟩ (no default)

Default: `0.0060`.

Accepted without solver effect; autonode measures the real TeX node.

Example: `label box char width=0.007`.

/luacoolprop/diagram/common/label box height=⟨*number*⟩ (no default)

Default: `0.060`.

Accepted without solver effect; autonode measures the real TeX node.

Example: `label box height=0.07`.

/luacoolprop/diagram/common/label box sloped multiplier=⟨*number*⟩ (no default)

Default: `1.35`.

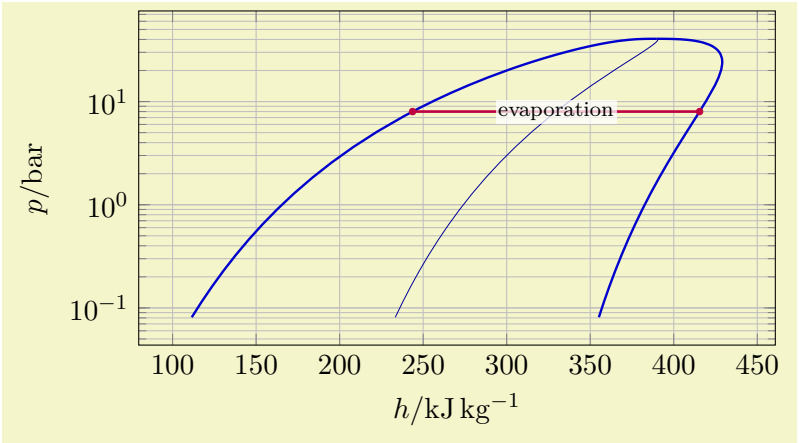
Accepted without solver effect; use `auto node bbox mode=oriented`.

Example: `label box sloped multiplier=1.4`.

27 Shared process paths and the PH projection

A process is drawn inside an existing PH axis. Its key namespace is `/luacoolprop/process/PH`. State specifications in `from` and `to` are braced key lists interpreted by the thermodynamic layer.

A labelled isobaric process over the quality dome




```
\begin{tikzpicture}
\begin{axis}[width=10cm,height=6cm,ymode=log,auto node placement,
  xlabel={$h/\mathrm{kJ\,kg^{-1}}$},ylabel={$p/\mathrm{bar}$},
  grid=both]
  \LCPAddPHQuality[quality values={0,0.5,1}]
  \LCPAddPHProcess[type=isobar,pressure=8bar,
    from={quality=0},to={quality=1},
    color=purple,label={evaporation},mark endpoints=true]
\end{axis}
\end{tikzpicture}
```

27.1 Process reference, diagnostics, and axis units

/luacoolprop/process/common/reference state=⟨*DEF, IIR, ASHRAE, or NBP*⟩ (no default)

Default: `global convention`.

Must agree with the reference state already locked for this fluid.

Example: `reference state=DEF`.

`/luacoolprop/process/common/domain policy`=⟨*ignore, warning, or error*⟩ (no default)

Default: `ignore`.

Controls diagnostics for disconnected valid components; it never joins across an invalid region.

Example: `domain policy=warning`.

Accepted alias: `discontinuity policy` forwards to `domain policy`.

`/luacoolprop/process/PH/enthalpy axis unit`=⟨*jkg or kjkg*⟩ (no default)

Default: `global unit`.

Selects the process coordinate conversion; use the same unit for the background and axis.

Example: `enthalpy axis unit=jkg`.

`/luacoolprop/process/PH/pressure axis unit`=⟨*pa, kpa, mpa, or bar*⟩ (no default)

Default: `global unit`.

Selects the process coordinate conversion; use the same unit for the background and axis.

Example: `pressure axis unit=kpa`.

`/luacoolprop/process/PV/specific volume axis unit`=⟨*m3kg or lkg*⟩ (no default)

Default: `global unit`.

Selects the process coordinate conversion; use the same unit for the background and axis.

Example: `specific volume axis unit=lkg`.

`/luacoolprop/process/PV/pressure axis unit`=⟨*pa, kpa, mpa, or bar*⟩ (no default)

Default: `global unit`.

Selects the process coordinate conversion; use the same unit for the background and axis.

Example: `pressure axis unit=kpa`.

`/luacoolprop/process/TS/entropy axis unit`=⟨*jkgk or kjkgk*⟩ (no default)

Default: `global unit`.

Selects the process coordinate conversion; use the same unit for the background and axis.

Example: `entropy axis unit=jkgk`.

`/luacoolprop/process/TS/temperature axis unit`=⟨*kelvin or celsius*⟩ (no default)

Default: `global unit`.

Selects the process coordinate conversion; use the same unit for the background and axis.

Example: `temperature axis unit=celsius`.

`/luacoolprop/process/HS/entropy axis unit`=⟨*jkgk or kjkgk*⟩ (no default)

Default: `global unit`.

Selects the process coordinate conversion; use the same unit for the background and axis.

Example: `entropy axis unit=jkgk`.

`/luacoolprop/process/HS/enthalpy axis unit`=⟨*jkg or kjkg*⟩ (no default)

Default: `global unit`.

Selects the process coordinate conversion; use the same unit for the background and axis.

Example: `enthalpy axis unit=jkg`.

27.2 Thermodynamic definition

`/luacoolprop/process/common/fluid`=⟨*CoolProp identifier*⟩ (no default)

Default: `global fluid`.

Selects the process fluid.

Example: `fluid=R134a`.

`/luacoolprop/process/common/library`=⟨*file path*⟩ (no default)

Default: `global library`.

Selects the library for this request before the first native load. It cannot switch an already loaded library; use a fresh compiler process.

Example: `library=/opt/coolprop/libCoolProp.dylib`.

`/luacoolprop/process/common/type`=⟨*process kind*⟩ (no default)

Default: `isobar`.

Selects the property held constant along the path.

`isobar` Constant pressure.

`isotherm` Constant temperature.

`isenthalp` Constant specific enthalpy.

`isentrope` Constant specific entropy.

`isochore` Constant specific volume.

`quality` Constant vapour quality.

Example: `type=isentrope`.

The descriptive aliases `isobaric`, `pressure`, `isothermal`, `temperature`, `isentropic`, `entropy`, `isenthalpic`, `enthalpy`, `throttle`, `isoquality`, `isochoric`, and `volume`, as well as the one-letter forms `p`, `t`, `s`, `h`, `q`, and `v`, are accepted as aliases. The canonical vocabulary is shown in the list above.

Example: `type=isobar`.

`/luacoolprop/process/common/from`=⟨*state key list*⟩ (no default)

Default: `empty`.

Defines the initial state on the process constraint.

Example: `from={quality=0}`.

`/luacoolprop/process/common/to`=⟨*state key list*⟩ (no default)

Default: `empty`.

Defines the final state on the process constraint.

Example: `to={quality=1}`.

`/luacoolprop/process/common/value`=⟨*quantity*⟩ (no default)

Default: `empty`.

Supplies the conserved property when its named key is not used. Unit parsing follows the selected process type.

Example: `value=8bar`.

/luacoolprop/process/common/pressure=⟨*pressure*⟩ (no default)

Default: `empty`.

Sets an isobar. A bare number is in pascals; accepted suffixes are `Pa`, `kPa`, `MPa`, `bar`, and `mbar`.

Example: `pressure=8bar`.

/luacoolprop/process/common/temperature=⟨*temperature*⟩ (no default)

Default: `empty`.

Sets an isotherm. A bare number is in kelvin; suffix `C`, `degC`, or `celsius` selects degrees Celsius.

Example: `temperature=20C`.

/luacoolprop/process/common/enthalpy=⟨*specific enthalpy*⟩ (no default)

Default: `empty`.

Sets an isenthalpic path. A bare number is in `J/kg`; `kJkg` selects `kJ/kg`.

Example: `enthalpy=250kJkg`.

/luacoolprop/process/common/entropy=⟨*specific entropy*⟩ (no default)

Default: `empty`.

Sets an isentrope. A bare number is in `J/(kg K)`; `kJkgK` selects `kJ/(kg K)`.

Example: `entropy=1.8kJkgK`.

/luacoolprop/process/common/quality=⟨*number*⟩ (no default)

Default: `empty`.

Sets constant vapour quality for a quality path.

Example: `quality=0.5`.

/luacoolprop/process/common/specific volume=⟨*specific volume*⟩ (no default)

Default: `empty`.

Sets an isochore in cubic metres per kilogram.

Example: `specific volume=0.05`.

The nested `from` and `to` lists accept the state properties below. A state normally needs two independent properties; the conserved process property may supply one of them. A colon may replace the equals sign, and a semicolon may replace a comma.

State key	Example	Meaning
`p, pressure`	`p=8bar`	Pressure; the suffix rules are the same as above.
`T, temperature`	`T=20C`	Temperature in kelvin unless a Celsius suffix is present.
`h, enthalpy`	`h=250kJkg`	Specific enthalpy.
`s, entropy`	`s=1.8kJkgK`	Specific entropy.
`q, Q, quality, x`	`quality=0.5`	Vapour quality from 0 to 1.
`v, volume, specific volume`	`v=0.05m3/kg`	Specific volume; bare values use $\mathrm{m^3\,kg^{-1}}$, and `L/kg` or `dm3/kg` is also accepted.
`rho, density, rhomass`	`rho=25kg/m3`	Mass density; bare values use $\mathrm{kg\,m^{-3}}$ and are converted to specific volume.

For example, `from={p=8bar,quality=0}` is fully specified by itself, whereas `from={quality=0}` becomes complete when the surrounding isobar also supplies `pressure=8bar`. LuaCoolProp reports

incomplete, non-physical, and out-of-range states with suggestions rather than silently drawing a misleading process.

When more than two properties are supplied, LuaCoolProp freezes the first independent pair in the order documented by the Lua API, asks CoolProp for the complete canonical state, and verifies every additional explicit property. It also verifies the conserved property against both completed process endpoints. A contradiction is a hard error whose diagnostic gives the supplied value, the CoolProp value, their absolute difference, and the applied tolerance. Comparisons use the larger of an absolute floor and a relative tolerance of 5×10^{-7}. The absolute floors in SI units are 0.1 for pressure, 10^{-5} for temperature, 10^{-3} for enthalpy, 10^{-5} for entropy, 10^{-9} for quality, 10^{-12} for specific volume, and 10^{-8} for density. These tolerances absorb flash-solver roundoff but do not conceal scientifically distinct states.

Accepted alias: `kind` forwards to `type`.

Accepted alias: `process` forwards to `type`.

Accepted alias: `start` forwards to `from`.

Accepted alias: `stop` forwards to `to`.

Accepted alias: `p` forwards to `pressure`.

Accepted alias: `T` forwards to `temperature`.

Accepted alias: `t` forwards to `temperature`.

Accepted alias: `h` forwards to `enthalpy`.

Accepted alias: `s` forwards to `entropy`.

Accepted alias: `q` forwards to `quality`.

Accepted alias: `Q` forwards to `quality`.

Accepted alias: `v` forwards to `specific volume`.

27.3 Identity, appearance, and labels

`/luacoolprop/process/common/name`=⟨*identifier*⟩ (no default)

Default: `empty`.

Assigns the stable TikZ `name path` selector used by labels, intersections, and higher-level documents.

Example: `name=evaporation`.

`/luacoolprop/process/common/color`=⟨*colour*⟩ (no default)

Default: `black`.

Sets the process-curve colour.

Example: `color=purple`.

`/luacoolprop/process/common/style`=⟨*TikZ style*⟩ (no default)

Default: `line width=1pt`.

Sets process drawing options.

Example: `style={line width=1.4pt,-{Stealth}}`.

Arrow-tip styles such as `-{Stealth}` require the document to load TikZ's `arrows.meta` library explicitly. LuaCoolProp loads only the `intersections` library required by its stable named-path contract.

`/luacoolprop/process/common/label`=⟨*TeX text*⟩ (no default)

Default: `empty`.

Adds a label managed by autonode.

Example: `label={$1\to 2$}`.

`/luacoolprop/process/common/label pos`=⟨*fraction*⟩ (no default)

Default: `0.5`.

Sets the preferred label position along the process.

Example: `label pos=0.6`.

`/luacoolprop/process/common/label style`=⟨*TikZ style name*⟩ (no default)

Default: `luacoolprop process label node`.

Selects the process-label node style.

Example: `label style=luacoolprop process label node`.

`/luacoolprop/process/common/label node style`=⟨*TikZ node options*⟩ (no default)

Default: `package style`.

Redefines `luacoolprop process label node`.

Example: `label node style={fill=purple!8,text=purple}`.

`/luacoolprop/process/common/label sloped`=⟨*boolean*⟩ (no default)

Default: `true`.

Rotates the process label with the local tangent.

Example: `label sloped=false`.

`/luacoolprop/process/common/label allow upside down`=⟨*boolean*⟩ (no default)

Default: `false`.

Allows the label to retain an upside-down tangent orientation.

Example: `label allow upside down=true`.

`/luacoolprop/process/common/mark endpoints`=⟨*boolean*⟩ (no default)

Default: `false`.

Draws markers at the start and end states.

Example: `mark endpoints=true`.

`/luacoolprop/process/common/marker style`=⟨*TikZ plot style*⟩ (no default)

Default: `only marks, mark=*`.

Styles endpoint markers.

Example: `marker style={only marks,mark=square*,mark size=1.5pt}`.

Accepted alias: `id` forwards to `name`.
Accepted alias: `process color` forwards to `color`.
Accepted alias: `process style` forwards to `style`.
Accepted alias: `label text` forwards to `label`.
Accepted alias: `sloped label` forwards to `label sloped`.
Accepted alias: `markers` forwards to `mark endpoints`.

27.4 Named intersections and numeric endpoint access

`/luacoolprop/process/common/export coordinates=`⟨*control-sequence prefix*⟩ (no default)

Default: `empty`.

Globally defines four displayed-coordinate macros and all thermodynamic endpoint macros that CoolProp can resolve. The coordinate macros are \⟨*prefix*⟩`FromX`, \⟨*prefix*⟩`FromY`, \⟨*prefix*⟩`ToX`, and \⟨*prefix*⟩`ToY`. Give the prefix without a leading backslash. Values use the displayed coordinate scales and the precision selected by `coord digits`.

Example: `export coordinates=CycleAB`.

`/luacoolprop/process/common/log coordinates=`⟨*boolean*⟩ (no default)

Default: `false`.

Writes the process name and its two displayed coordinate pairs to both the terminal and the TeX log.

Example: `log coordinates=true`.

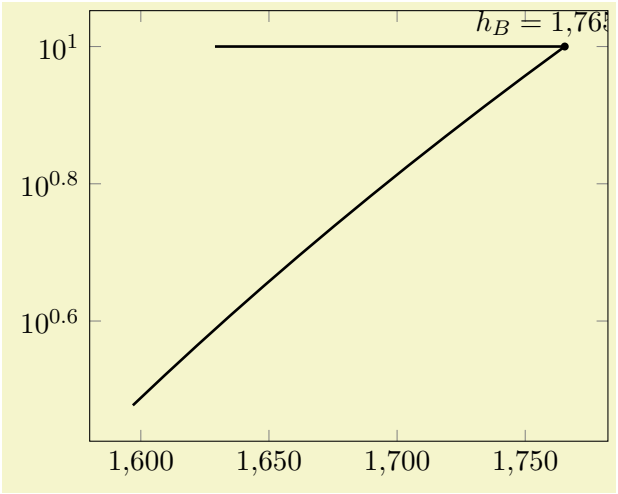
The exported definitions are global because a process can be evaluated inside a PGFPlots group while its numerical results are needed later in the document. A previously exported numeric macro may be used directly inside a later `from` or `to` specification. LuaCoolProp expands state data before passing it to Lua, so constructs such as `to={pressure=\PreviousToPressureSI Pa}` form a genuine calculation chain. TeX-rich process labels and styles follow their separate, protected serialization path and are not expanded as numerical state data. A prefix must begin with a letter, `@`, colon, or underscore; subsequent characters may also be digits. For example, `export coordinates=CycleAB` defines `\CycleABFromX`, `\CycleABFromY`, `\CycleABToX`, and `\CycleABToY`. The values are plain decimal text, without units, and reproduce exactly the rounded coordinates sent to PGFPlots. Thus the meaning of X and Y follows the diagram type; for a PH diagram with the default scales they are respectively $\mathrm{kJ\,kg^{-1}}$ and bar.

For each endpoint, the bridge also defines every available macro in the table below. Replace ⟨*side*⟩ with `From` or `To`. A property that CoolProp cannot resolve for that state is intentionally left undefined.

`Suffix after prefix and side`	Unit	Meaning
`PressureSI`	Pa	Absolute pressure.
`TemperatureK`	K	Absolute temperature.
`EnthalpySI`	$\mathrm{J\,kg^{-1}}$	Mass-specific enthalpy.
`EntropySI`	$\mathrm{J\,kg^{-1}\,K^{-1}}$	Mass-specific entropy.
`SpecificVolumeSI`	$\mathrm{m^3\,kg^{-1}}$	Mass-specific volume.
`Quality`	1	Vapour quality, defined only in the two-phase region or on its boundary.

For example, `\CycleABToTemperatureK` is the final absolute temperature, while `\CycleABToEnthalpySI` retains the SI value independently of the horizontal plot scale. These property exports use the same decimal precision setting as the coordinate exports.

Intersecting named processes and reusing their numbers




```
\usetikzlibrary{intersections}
\begin{tikzpicture}
\begin{axis}[ymode=log,auto node placement]
  \LCPAddPHProcess[
    fluid=R717,
    type=isentrope,
    from={pressure=3bar,quality=1},to={pressure=10bar},
    name=cycle-ab,export coordinates=CycleAB,
    log coordinates=true]
  \LCPAddPHProcess[
    fluid=R717,type=isobar,pressure=10bar,
    from={entropy=6.2276889kJkgK},to={quality=1},
    name=cycle-bc]
  \path[name intersections={of=cycle-ab and cycle-bc,by=CycleB}];
  \fill (CycleB) circle[radius=1.5pt];
  \node[above] at (CycleB)
    {$h_B=\pgfmathprintnumber{\CycleABToX}$};
\end{axis}
\end{tikzpicture}
```

The geometry above comes from TikZ's `intersections` library; no state coordinate is repeated manually. Independently, a document can write the same numeric endpoint to its own log with `\typeout{B=(\CycleABToX,\CycleABToY)}` or use the macros in calculations, tables, and annotations.

27.5 Process scaling and sampling

`/luacoolprop/process/common/enthalpy scale`=⟨*number*⟩ (no default)

Default: `global scale`.

Multiplies process enthalpies before plotting.

Example: `enthalpy scale=0.001`.

`/luacoolprop/process/common/pressure scale`=⟨*number*⟩ (no default)

Default: `global scale`.

Multiplies process pressures before plotting.

Example: `pressure scale=0.00001`.

`/luacoolprop/process/common/initial intervals`=⟨*positive integer*⟩ (no default)

Default: `12`.

Sets initial process-path subdivisions.

Example: `initial intervals=18`.

`/luacoolprop/process/common/max depth`=⟨*nonnegative integer*⟩ (no default)

Default: `7`.

Sets maximum adaptive process depth.

Example: `max depth=9`.

`/luacoolprop/process/common/tolerance`=⟨*positive number*⟩ (no default)

Default: `0.25`.

Sets geometric process refinement tolerance.

Example: `tolerance=0.15`.

`/luacoolprop/process/common/log weight`=⟨*nonnegative number*⟩ (no default)

Default: `30`.

Weights logarithmic pressure differences during refinement.

Example: `log weight=40`.

`/luacoolprop/process/common/coord digits`=⟨*positive integer*⟩ (no default)

Default: `6`.

Sets decimal precision emitted for process coordinates.

Example: `coord digits=8`.

Accepted alias: `h scale` forwards to `enthalpy scale`.

Accepted alias: `p scale` forwards to `pressure scale`.

28 PV pressure–specific-volume diagrams

A PV diagram plots absolute pressure p as a function of mass-specific volume $v = 1/\rho$. With the default scales its coordinates are v in $\mathrm{m^3\,kg^{-1}}$ and p in bar. Both axes are logarithmic by default: the liquid region, two-phase dome, vapour region, and several pressure decades therefore remain legible in one plot. Every emitted point also retains its raw SI pressure, specific volume, and density in the Lua record.

PV is a first-class registered diagram type, not a graphical transformation of PH. CoolProp evaluates density directly for each thermodynamic pair. Adaptive sampling measures curvature in $(\log v, \log p)$ display coordinates; this is especially important close to the saturated-liquid boundary and the critical point.

Supported PV background families

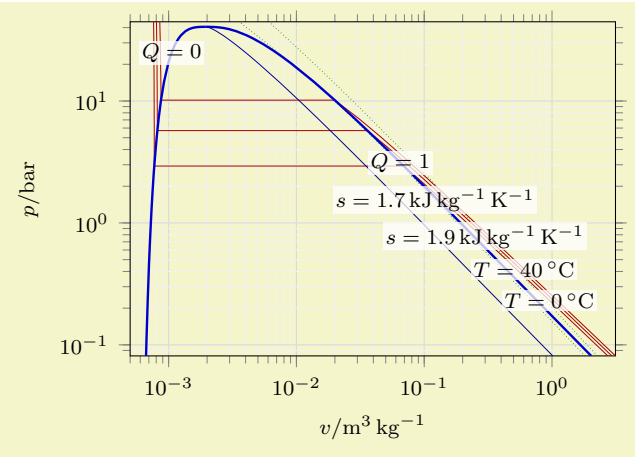
The public PV background families are `quality`, `isotherm`, and `isentrope`. An isochore would be only a vertical line in these coordinates, so it is not a PV background family. Isochores remain available as PH families and as PV process paths. PV processes additionally support isobars, isenthalps, and the other process types documented below.

28.1 PV convenience commands

`\LCPPVDiagram`[⟨*options*⟩]

Creates a complete PV picture and a log–log PGFPlots axis. Quality curves are enabled by default; isotherms and isentropes are opt-in.

A complete labelled PV diagram



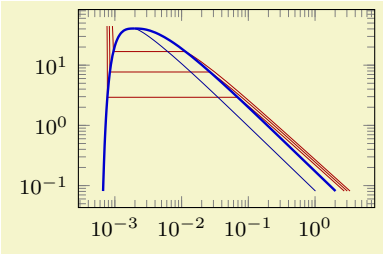

```
\LCPPVDiagram[
  fluid=R134a,
  isotherm=true,
  isentrope=true,
  quality values={0,0.5,1},
  temperature values={0,20,40},
  entropy values={1.7,1.9},
  labels=true,
  axis options={xmin=0.0005,width=8cm,
    height=6cm,font=\scriptsize}
]
```

It is exactly equivalent to `\LCPDiagram{PV}[`⟨*options*⟩`]`. The extra space on the liquid side leaves room for the saturated-liquid label while retaining the automatic inside-axis constraint.

`\LCPAddPVPlots[`⟨*options*⟩`]`

Adds every enabled PV family to an existing axis. The caller must select positive logarithmic coordinates when reproducing the standard appearance.

All selected families in a manual axis

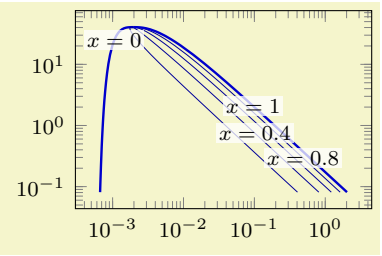



```
\begin{tikzpicture}
\begin{loglogaxis}[width=5.5cm,auto node placement,
  height=4.2cm,font=\scriptsize]
  \LCPAddPVPlots[
    quality values={0,0.5,1},
    isotherm=true,
    temperature values={0,30,60}]
\end{loglogaxis}
\end{tikzpicture}
```

`\LCPAddPVQuality[`⟨*options*⟩`]`

Adds only the saturation boundaries and selected interior qualities. Every subcritical quality curve approaches the same quality-independent critical limit; quality is undefined at the critical state itself.

PV quality curves

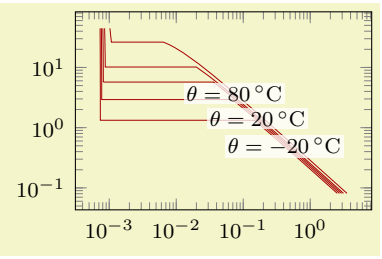



```
\begin{tikzpicture}
\begin{loglogaxis}[width=5.5cm,auto node placement,
  height=4.2cm,font=\scriptsize]
  \LCPAddPVQuality[
    quality step=0.2,
    quality symbol=x,
    labels=true]
\end{loglogaxis}
\end{tikzpicture}
```

`\LCPAddPVIsotherms`[⟨*options*⟩]

Adds only isotherms. A subcritical isotherm contains its superheated-vapour branch, the exact horizontal segment between saturated vapour and saturated liquid, and its compressed-liquid branch.

PV isotherms and their plateaux

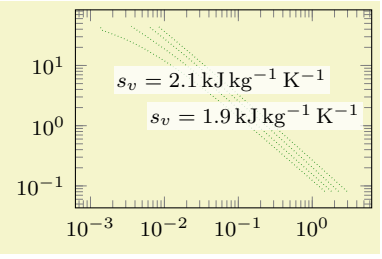



```
\begin{tikzpicture}
\begin{loglogaxis}[width=5.5cm,auto node placement,
  height=4.2cm,font=\scriptsize]
  \LCPAddPVIsotherms[
    temperature values={-20,0,20,40,80},
    temperature symbol={\theta},
    labels=true]
\end{loglogaxis}
\end{tikzpicture}
```

`\LCPAddPVIsentropes`[⟨*options*⟩]

Adds only constant-specific-entropy curves. Entropy inputs follow `entropy unit`; the default is $\mathrm{kJ\,kg^{-1}\,K^{-1}}$.

PV isentropes

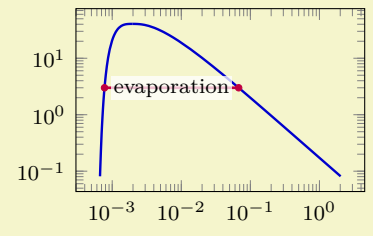



```
\begin{tikzpicture}
\begin{loglogaxis}[width=5.5cm,auto node placement,
  height=4.2cm,font=\scriptsize]
  \LCPAddPVIsentropes[
    entropy values={1.5,1.7,1.9,2.1},
    entropy symbol={s_v},
    labels=true]
\end{loglogaxis}
\end{tikzpicture}
```

`\LCPAddPVProcess`[⟨*options*⟩]

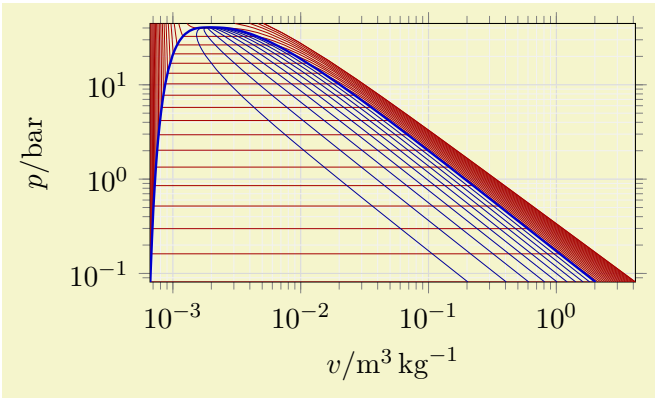
Adds one process in PV coordinates. State specifications, conserved-property inference, named paths, endpoint exports, styles, and diagnostics are identical to `\LCPAddPHProcess`; X exports are specific volumes instead of enthalpies.

A named PV evaporation process



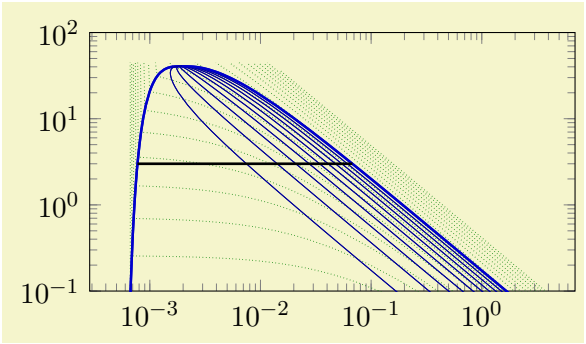

```
\begin{tikzpicture}
\begin{loglogaxis}[width=5.4cm,auto node placement,
  height=4cm,font=\scriptsize]
  \LCPAddPVQuality[
    quality values={0,1}]
  \LCPAddPVProcess[
    type=isobar,pressure=3bar,
    from={quality=0},to={quality=1},
    name=pv-evaporation,
    color=purple,label={evaporation},
    mark endpoints=true]
\end{loglogaxis}
\end{tikzpicture}
```

The generic spellings are also fully equivalent:




```
\LCPDiagram{PV}[isotherm=true,
  axis options={width=8cm,height=5cm}]
```

PV through the generic registry



```
\begin{tikzpicture}
\begin{axis}[width=8cm,height=5cm,auto node placement,
  xmode=log,ymode=log,ymin=0.1,ymax=100]
\LCPAddDiagramPlots{PV}[isentrope=true]
\LCPAddDiagramFamily{PV}{quality}[quality step=0.1]
\LCPAddDiagramProcess{PV}[
  type=isobar,pressure=3bar,
  from={quality=0},to={quality=1}]
\end{axis}
\end{tikzpicture}
```

28.2 PV keys and defaults

Options are read from `/luacoolprop/diagram/PV`. PV intentionally combines the format-independent controls with its own declared capabilities. The following table states precisely which shared groups apply; their definitions appear in Section 26.

Group	Keys accepted without change in PV
Source and selection	`fluid`, `library`, `quality`, `isotherm`, `isentrope`, and their documented aliases.
Pressure	`pressure min`, `pressure max`, `pressure max factor`, `pressure scale`, and aliases.
Quality grid	Every `quality ...` grid key and every `q ...` alias: mode, values, minimum, maximum, step, count, and preset.
Temperature grid	Every `temperature ...` and `t ...` grid key, including unit, mode, values, limits, step, count, and preset.
Entropy grid	Every `entropy ...` and `s ...` grid key, including unit, mode, values, limits, step, count, and preset.
Symbols	`pressure symbol`, `specific volume symbol`, `quality symbol`, `temperature symbol`, and `entropy symbol`.
Appearance	General curve style, quality boundary/interior styles, family colours and styles, legend, forget plot, and `axis options`.
Labels	General and family-specific quality, isotherm, and isentrope label keys; all autonode forwarding and per-curve override keys.
Sampling	`initial intervals`, `max depth`, `tolerance`, `log weight`, family-specific refinements, `saturation pressure epsilon`, and `coord digits`.

PV has no isochore background family: an explicit `isochore` switch is an error. To draw an isochoric transformation, use `\LCPAddPVProcess` with `type=isochore`.

`/luacoolprop/diagram/PV/specific volume scale`=⟨*positive number*⟩ (no default)

Default: `global scale, initially 1`.

Advanced raw multiplier for SI specific volume. It produces the explicit scaled-SI axis label.

Example: `specific volume scale=500`.

Accepted alias: `v scale` forwards to `specific volume scale`.

`/luacoolprop/diagram/PV/specific volume axis unit`=⟨*m3kg or lkg*⟩ (no default)

Default: `m3kg`.

Atomically selects the specific-volume conversion and horizontal label.

Example: `specific volume axis unit=lkg`.

`/luacoolprop/diagram/PV/pressure axis unit`=⟨*pa, kpa, mpa, or bar*⟩ (no default)

Default: `bar`.

Atomically selects the pressure conversion and vertical label.

Example: `pressure axis unit=kpa`.

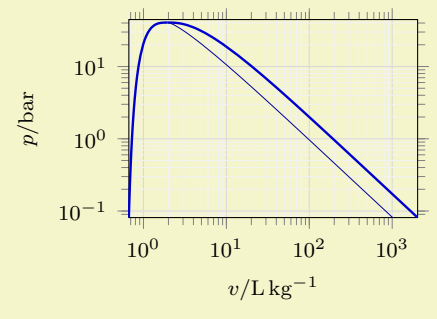
`/luacoolprop/diagram/PV/log x weight`=⟨*nonnegative number*⟩ (no default)

Default: `30`.

Weights deviations in $\log_{10} v$ during adaptive refinement. Increasing it produces more points where a curve bends sharply on the logarithmic volume axis.

Example: `log x weight=45`.

Scaling the PV abscissa to litres per kilogram




```
\LCPPVDiagram[
  specific volume axis unit=lkg,
  quality values={0,0.5,1},
  axis options={width=5.4cm,
    height=4.2cm,font=\scriptsize}
]
```

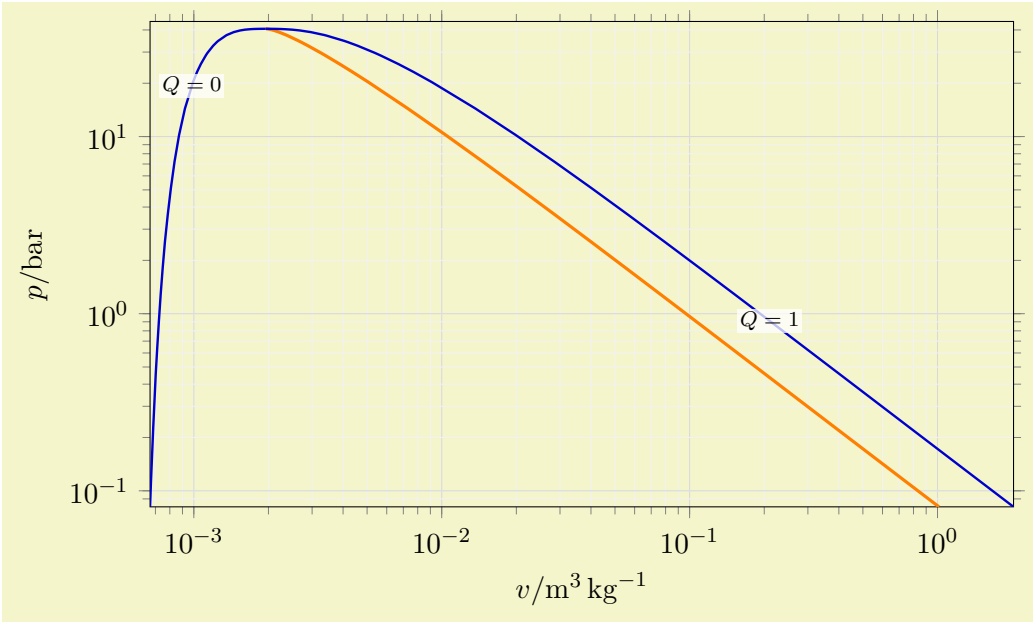
28.3 PV curve names, intersections, and overrides

Stable PV path names are separate from PH names:

Family	Pattern	Example
quality	`lcp-pv-q-`⟨*value*⟩	`lcp-pv-q-0p5`
isotherm	`lcp-pv-T-`⟨*value*⟩`-C`	`lcp-pv-T-20-C`
isentrope	`lcp-pv-s-`⟨*value*⟩`-kJkgK`	`lcp-pv-s-1p8-kJkgK`
process	`lcp-pv-process-`⟨*type*⟩`-...`	customisable with `name`

Consequently the common per-curve keys target PV curves by their PV name:

A PV-specific curve override




```
\LCPPVDiagram[
  quality values={0,0.5,1},labels=true,
  curve style for={lcp-pv-q-0p5}{very thick,orange},
  label text for={lcp-pv-q-0p5}{$x=\qty{50}{\percent}$},
  label fixed for={lcp-pv-q-0p5}{true},
  label pos for={lcp-pv-q-0p5}{0.55}]
```

Names can be consumed by TikZ's `intersections` library exactly as PH names can. Automatic label placement remains entirely delegated to the required external `pgfplots-autonode` library.

28.4 PV process keys and numerical exports

PV process options use the `PV` branch of the `/luacoolprop/process` namespace. It uses the shared state definition, process selection, style, marker, label, naming, sampling, and coordinate-export keys. All six process types are supported:

`isobar` exact horizontal segment at constant pressure;

`isochore` exact vertical segment at constant specific volume;

`isotherm` adaptively sampled constant-temperature path, including its two-phase plateau where applicable;

`isentropic` adaptively sampled constant-entropy path;

`isenthalpic` adaptively sampled throttling path; and

`quality` adaptively sampled constant-quality path, with `isoquality` accepted as an alias.

`/luacoolprop/process/PV/specific volume scale`=⟨*positive number*⟩ (no default)

Default: `global scale, initially 1`.

Multiplies endpoint and sampled specific volumes before plotting. It also determines the unit scale of exported `FromX` and `ToX`.

Example: `specific volume scale=1000`.

Accepted alias: `v scale` forwards to `specific volume scale`.

`/luacoolprop/process/PV/log x weight`=⟨*nonnegative number*⟩ (no default)

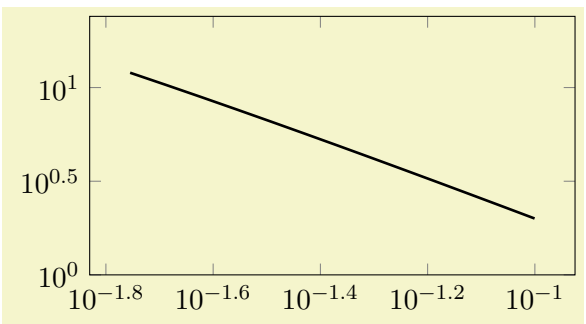
Default: `30`.

Weights logarithmic-volume curvature while refining a non-straight PV process path.

Example: `log x weight=45`.

With `export coordinates=StateAB`, the generic property exports retain the SI values documented in the shared process reference. In addition, the display-coordinate meanings are now `\StateABFromX` $= v_{\mathrm{from}}$ times `specific volume scale` and `\StateABFromY` $= p_{\mathrm{from}}$ times `pressure scale`, with analogous `ToX` and `ToY` definitions. This distinction makes numerical tables independent of plot units: use `FromSpecificVolumeSI` for SI calculations and `FromX` for an annotation aligned with the axis.

Exporting PV endpoint coordinates and SI properties



```
\begin{tikzpicture}
\begin{axis}[width=8cm,height=5cm,auto node placement,ymin=1,ymax=24,xmode=log,ymode=log]
\LCPAddPVProcess[
  type=isentropic,
  from={p=2bar,quality=1},to={p=12bar},
  export coordinates=Compression,
  log coordinates=true]
% Plot coordinates: \CompressionFromX, \CompressionToX.
% Raw SI properties: \CompressionFromSpecificVolumeSI,
%                    \CompressionToPressureSI.
\end{axis}
\end{tikzpicture}
```

28.5 Plain TeX and ConTeXt

The implementation is in the generic layer; neither wrapper has a reduced PV feature set.

Plain LuaTeX

```
\input p-luacoolprop.tex
\pgfplotsset{compat=1.18}
\LCPPVDiagram[isotherm=true,isentrope=true]
\bye
```

ConTeXt MkIV

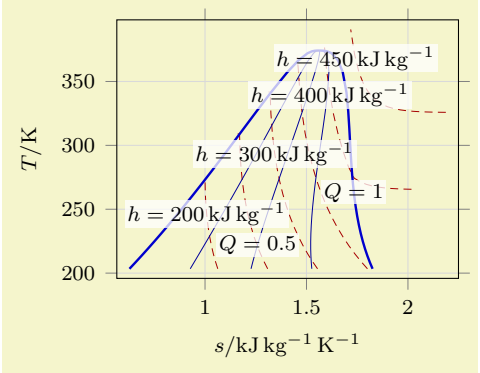
```
\usemodule[t][luacoolprop]
\pgfplotsset{compat=1.18}
\starttext
  \LCPPVDiagram[isotherm=true,isentrope=true]
\stoptext
```

29 TS temperature–entropy diagrams

A TS diagram places mass-specific entropy on the horizontal axis and absolute temperature on the vertical axis. LuaCoolProp keeps CoolProp values in SI units internally and applies `entropy scale=0.001` and `temperature scale=1` at the plotting boundary. The resulting default axes therefore read $\mathrm{kJ\,kg^{-1}\,K^{-1}}$ and K. Both axes are linear. Unlike a PH or PV diagram, pressure is a sampling variable rather than a plotted coordinate.

The saturation family contains the two boundaries $Q = 0$ and $Q = 1$, optional interior qualities, and one shared quality-independent critical limit obtained from the unique critical state. Isenthalps are sampled adaptively from constant mass-specific enthalpy and pressure. This construction is fluid-independent: no R134a coordinates or refrigerant-specific critical correction are stored in the package.

A complete TS diagram



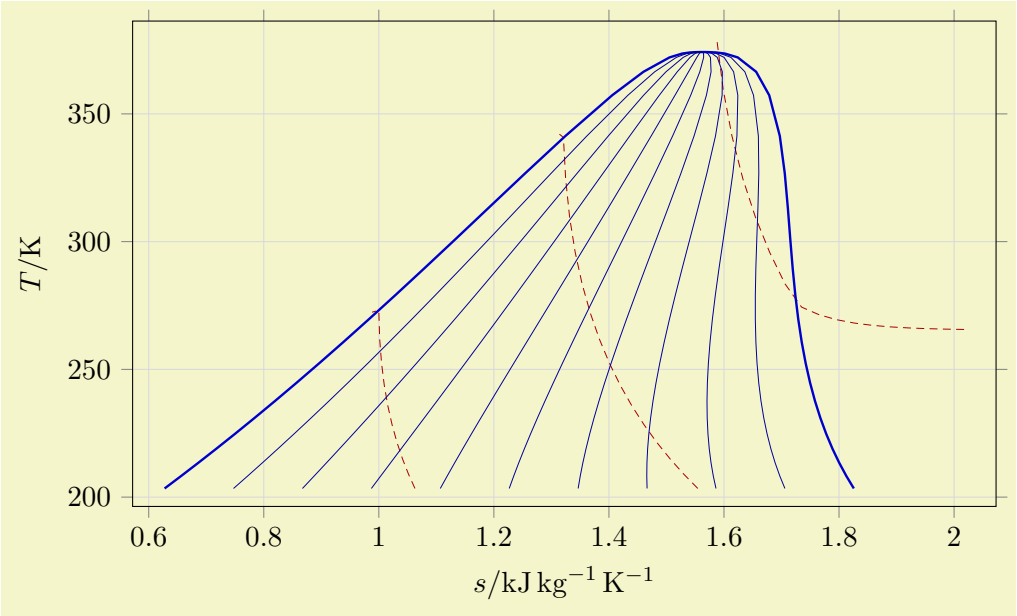
```
\LCPTSDiagram[
  fluid=R134a,
  isenthalp=true,
  quality values={0,0.25,0.5,0.75,1},
  enthalpy values={200,250,300,350,400,450},
  labels=true,
  axis options={width=6.1cm,height=5.0cm,
    font=\scriptsize}
]
```

29.1 TS commands

\LCPTSDiagram[⟨*options*⟩]

Creates a complete linear TS axis, computes data-dependent bounds, and draws all enabled families. It is equivalent to `\LCPDiagram{TS}`.

A saturation dome and selected isenthalps

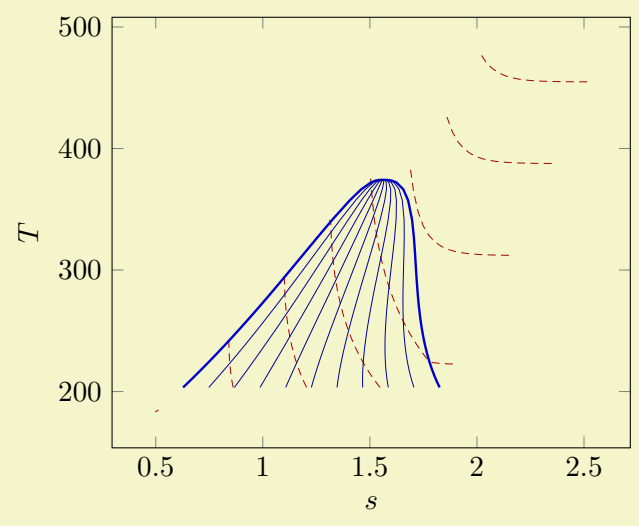



```
\LCPTSDiagram[fluid=R134a,isenthalp=true,
  enthalpy values={200,300,400}]
```

\LCPAddTSPlots[⟨*options*⟩]

Adds every enabled TS family to the current PGFPlots axis. Use this command when the surrounding axis, layers, annotations, and cycle are controlled by the document.

Low-level TS background

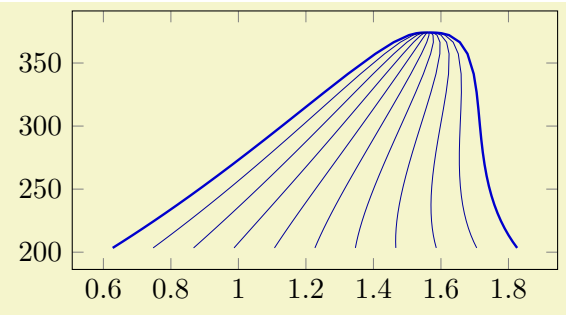



```
\begin{tikzpicture}
\begin{axis}[xlabel={$s$},ylabel={$T$}]
  \LCPAddTSPlots[fluid=R134a,isenthalp=true]
\end{axis}
\end{tikzpicture}
```

\LCPAddTSQuality[⟨*options*⟩]

Adds only the constant-quality family to the current axis. The stable path names have the form `lcp-ts-q-0p5`; they may be used with TikZ's `intersections` library. Near the critical state, sampling is adaptive in $\log_{10}((p_c - p)/p_c)$ as well as in the main log-pressure coordinate; the shared `tolerance` and `max depth` keys control both passes.

Only the saturation boundaries

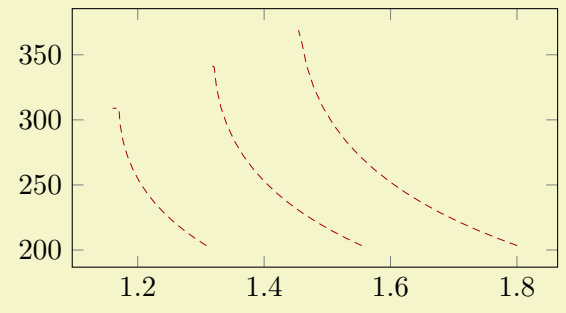



```
\begin{tikzpicture}
\begin{axis}[width=8cm,height=5cm,auto node placement,]
\LCPAddTSQuality[fluid=R134a,
  quality preset=boundaries]
\end{axis}
\end{tikzpicture}
```

\LCPAddTSIsenthalps[⟨*options*⟩]

Adds only constant mass-specific-enthalpy curves. Input values follow `enthalpy unit`; the default is `kjkg`.

Three explicit isenthalps

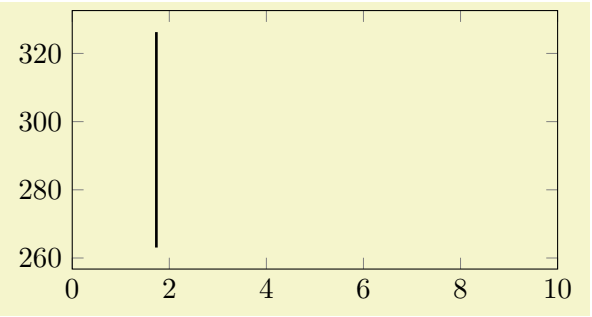



```
\begin{tikzpicture}
\begin{axis}[width=8cm,height=5cm,auto node placement,]
\LCPAddTSIsenthalps[fluid=R134a,
  enthalpy values={250,300,350}]
\end{axis}
\end{tikzpicture}
```

`\LCPAddTSProcess`[⟨*options*⟩]

Adds one process to a current TS axis. It accepts the same state language and all six shared process types. Isotherms are horizontal, isentropes vertical, and the remaining process types are thermodynamically sampled.

An isentropic compression

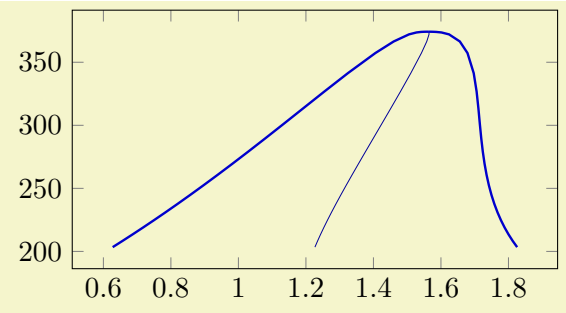



```
\begin{tikzpicture}
\begin{axis}[width=8cm,height=5cm,auto node placement,xmin=0,xmax=10,]
\LCPAddTSProcess[fluid=R134a,type=isentrope,
  from={p=2bar,Q=1},to={p=12bar}]
\end{axis}
\end{tikzpicture}
```

`\LCPAddTSIsoQuality`[⟨*options*⟩]

Alternate spelling for `\LCPAddTSQuality`; the latter is canonical.

Alternate spelling



```
\begin{tikzpicture}
\begin{axis}[width=8cm,height=5cm,auto node placement,]
\LCPAddTSIsoQuality[quality values={0,0.5,1}]
\end{axis}
\end{tikzpicture}
```

The generic forms are fully equivalent:

Generic TS family and process dispatch

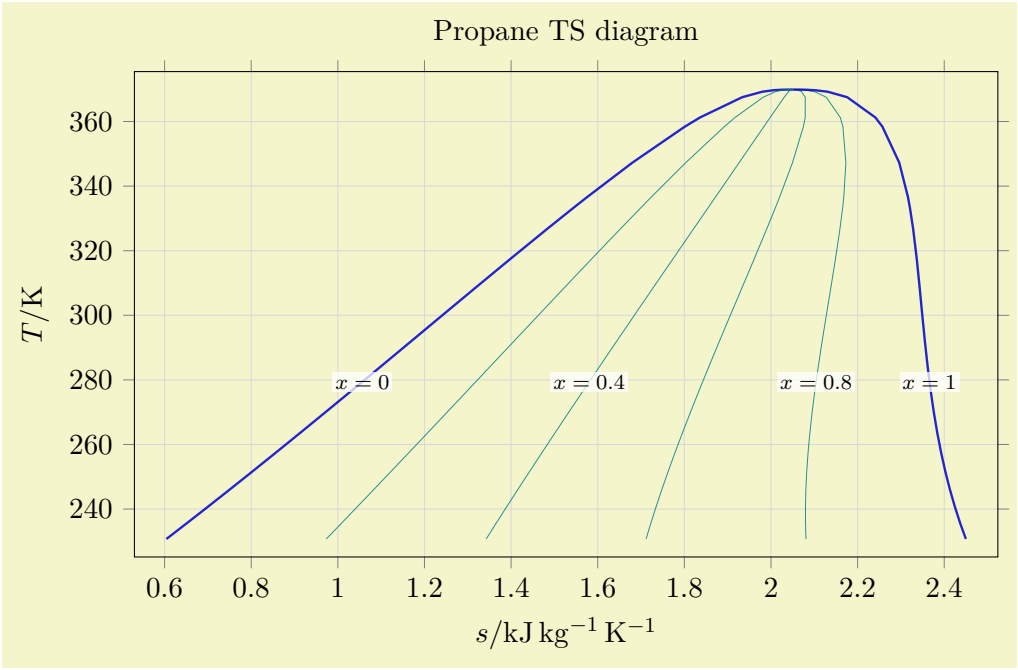
```
\LCPAddDiagramFamily{TS}{quality}[<options>]
\LCPAddDiagramFamily{TS}{isenthalp}[<options>]
\LCPAddDiagramProcess{TS}[<options>]
```

A generic client can thus select PH, PV, TS, or HS without changing its dispatch structure.

29.2 Shared diagram vocabulary

The TS namespace is `/luacoolprop/diagram/TS`. It deliberately reuses the shared keys for the fluid and library, pressure limits, quality grid, symbols, common sampling controls, curve overrides, autonode controls, styles, and labels. These controls have one definition used symmetrically by every applicable diagram type. For example:

Shared keys used by TS




```
\LCPTSDiagram[
  fluid=Propane,
  pressure min=100000,
  pressure max factor=1.05,
  quality symbol=x,
  quality step=0.2,
  labels=true,
  label placement=autonode,
  quality color=teal,
  curve style={opacity=.85},
  axis options={title={Propane TS diagram}}
]
```

TS does not register isotherm, isentrope, isochore, or isobar backgrounds, and their family switches therefore fail explicitly. Use the TS-specific `temperature axis min/max` and `entropy axis min/max` keys to constrain the plotted coordinate ranges.

29.3 TS family, scale, and grid keys

`/luacoolprop/diagram/TS/isenthalp`=⟨*boolean*⟩ (no default)

Default: `false`.

Enables the constant-mass-specific-enthalpy background family.

Example: `isenthalp=true`.

Accepted alias: `isenthalps` forwards to `isenthalp`.

Accepted alias: `enthalpy curves` forwards to `isenthalp`.

`/luacoolprop/diagram/TS/entropy scale`=⟨*number*⟩ (no default)

Default: `0.001`.

Advanced raw multiplier for SI entropy; it produces an explicit scaled-SI axis label.

Example: `entropy scale=0.002`.

Accepted alias: `s scale` forwards to `entropy scale`.

`/luacoolprop/diagram/TS/temperature scale`=⟨*number*⟩ (no default)

Default: `1`.

Multiplies vertical absolute temperatures. This is a multiplicative advanced interface with zero offset and a scaled-SI label.

Example: `temperature scale=0.5`.

Accepted alias: `t scale` forwards to `temperature scale`.

`/luacoolprop/diagram/TS/entropy axis unit`=⟨*jkgk or kjkgk*⟩ (no default)

Default: `kjkgk`.

Atomically selects the entropy conversion and horizontal label.

Example: `entropy axis unit=jkgk`.

`/luacoolprop/diagram/TS/temperature axis unit`=⟨*kelvin or celsius*⟩ (no default)

Default: `kelvin`.

Atomically selects the vertical label and conversion. Celsius subtracts 273.15 before plotting and is supported because TS uses a linear ordinate.

Example: `temperature axis unit=celsius`.

`/luacoolprop/diagram/TS/enthalpy unit`=⟨*unit*⟩ (no default)

Default: `kjkg`.

Controls explicit enthalpy-grid input and generated labels. Use `kjkg` for $\mathrm{kJ\,kg^{-1}}$ or `si` for $\mathrm{J\,kg^{-1}}$; CoolProp always receives SI values.

Example: `enthalpy unit=si`.

Accepted alias: `h unit` forwards to `enthalpy unit`.

`/luacoolprop/diagram/TS/enthalpy min`=⟨*number or auto*⟩ (no default)

Default: `auto`.

Lower value used by an automatically generated isenthalp grid.

Example: `enthalpy min=200`.

`/luacoolprop/diagram/TS/enthalpy max`=⟨*number or auto*⟩ (no default)

Default: `auto`.

Upper value used by an automatically generated isenthalp grid.

Example: `enthalpy max=450`.

`/luacoolprop/diagram/TS/enthalpy step`=⟨*positive number or auto*⟩ (no default)

Default: `auto`.

Linear spacing. With `auto`, `enthalpy count` controls the number of curves.

Example: `enthalpy step=50`.

`/luacoolprop/diagram/TS/enthalpy mode`=⟨*choice*⟩ (no default)

Default: `linear`.

Selects `linear` automatic generation or `list` interpretation of `enthalpy values`. Supplying a nonempty explicit list selects list mode automatically.

Example: `enthalpy mode=list`.

`/luacoolprop/diagram/TS/enthalpy count`=⟨*positive integer*⟩ (no default)

Default: `8`.

Number of linearly spaced automatic values when no step is supplied.

Example: `enthalpy count=6`.

`/luacoolprop/diagram/TS/enthalpy values`=⟨*comma list*⟩ (no default)

Default: `empty`.

Explicit enthalpies in the selected display unit.

Example: `enthalpy values={200,250,300,350,400}`.

Accepted alias: `h min` forwards to `enthalpy min`.
Accepted alias: `h max` forwards to `enthalpy max`.
Accepted alias: `h step` forwards to `enthalpy step`.
Accepted alias: `h mode` forwards to `enthalpy mode`.
Accepted alias: `h count` forwards to `enthalpy count`.
Accepted alias: `h values` forwards to `enthalpy values`.

29.4 TS isenthalp appearance and labels

`/luacoolprop/diagram/TS/isenthalp labels`=⟨*boolean or auto*⟩ (no default)

Default: `auto`.

Overrides the common `labels` switch for isenthalps.

Example: `isenthalp labels=true`.

/luacoolprop/diagram/TS/isenthalp label pos=⟨*fraction*⟩ (no default)

Default: `0.65`.

Preferred arc-length position forwarded to pgfplots-autonode.

Example: `isenthalp label pos=0.55`.

/luacoolprop/diagram/TS/isenthalp label every=⟨*positive integer*⟩ (no default)

Default: `2`.

Labels every nth isenthalp and always considers the last curve.

Example: `isenthalp label every=1`.

/luacoolprop/diagram/TS/isenthalp label sloped=⟨*boolean or auto*⟩ (no default)

Default: `auto`.

Requests a label aligned with its local isenthalp tangent.

Example: `isenthalp label sloped=true`.

/luacoolprop/diagram/TS/isenthalp label allow upside down=⟨*boolean or auto*⟩ (no default)

Default: `auto`.

Allows the automatic node to retain an upside-down tangent direction.

Example: `isenthalp label allow upside down=false`.

/luacoolprop/diagram/TS/isenthalp label style=⟨*TikZ style name*⟩ (no default)

Default: `luacoolprop isenthalp label node`.

Selects the TikZ node style used for generated enthalpy labels.

Example: `isenthalp label style=luacoolprop label node`.

/luacoolprop/diagram/TS/isenthalp label node style=⟨*TikZ options*⟩ (no default)

Default: `package default`.

Redefines the dedicated `luacoolprop isenthalp label node` style.

Example: `isenthalp label node style={fill=yellow!15,inner sep=1pt}`.

/luacoolprop/diagram/TS/isenthalp color=⟨*colour*⟩ (no default)

Default: `red!65!black`.

Sets the base colour of constant-enthalpy curves.

Example: `isenthalp color=orange!80!black`.

/luacoolprop/diagram/TS/isenthalp style=⟨*TikZ plot options*⟩ (no default)

Default: `thin dashed line`.

Appends family-specific drawing options after the colour.

Example: `isenthalp style={line width=.35pt,densely dotted}`.

The generated stable name includes the displayed enthalpy and unit; with the defaults, `enthalpy values={300}` creates `lcp-ts-h-300-kJkg`. All per-curve override keys operate on that name.

29.5 TS sampling and axis bounds

`/luacoolprop/diagram/TS/isenthalp initial intervals`=⟨*positive integer*⟩ (no default)

Default: `16`.

Initial logarithmic-pressure intervals before adaptive refinement.

Example: `isenthalp initial intervals=24`.

`/luacoolprop/diagram/TS/isenthalp max depth`=⟨*nonnegative integer*⟩ (no default)

Default: `8`.

Maximum recursive bisection depth for an isenthalp.

Example: `isenthalp max depth=10`.

`/luacoolprop/diagram/TS/isenthalp tolerance`=⟨*positive number*⟩ (no default)

Default: `0.10`.

Maximum scaled midpoint error accepted by isenthalp sampling.

Example: `isenthalp tolerance=0.06`.

`/luacoolprop/diagram/TS/temperature weight`=⟨*positive number*⟩ (no default)

Default: `0.01`.

Converts vertical deviations in kelvin to the scaled midpoint error used on the linear TS axis. Larger values refine temperature curvature more aggressively.

Example: `temperature weight=0.02`.

`/luacoolprop/diagram/TS/entropy axis min`=⟨*number or auto*⟩ (no default)

Default: `auto`.

Overrides the automatically padded horizontal lower bound in plotted entropy units.

Example: `entropy axis min=0.8`.

`/luacoolprop/diagram/TS/entropy axis max`=⟨*number or auto*⟩ (no default)

Default: `auto`.

Overrides the automatically padded horizontal upper bound.

Example: `entropy axis max=2.2`.

`/luacoolprop/diagram/TS/temperature axis min`=⟨*number or auto*⟩ (no default)

Default: `auto`.

Overrides the vertical lower bound after `temperature scale`.

Example: `temperature axis min=220`.

`/luacoolprop/diagram/TS/temperature axis max`=⟨*number or auto*⟩ (no default)

Default: `auto`.

Overrides the vertical upper bound after `temperature scale`.

Example: `temperature axis max=420`.

The general `initial intervals`, `max depth`, and `tolerance` keys remain fallbacks for both TS families. Quality curves use the same coordinate-independent critical-neighbourhood refinement contract as every other diagram. Tight tolerances increase both CoolProp calls and generated TeX coordinates; they should be changed only after inspecting the plotted result.

29.6 TS process keys and exported values

The process namespace is `/luacoolprop/process/TS`. Every shared process key applies: state pairs and units, `type`, constant-property keys, path identity, styling, endpoint markers, adaptive sampling, `export coordinates`, and `log coordinates`. Exported endpoint macros retain the complete SI state. Their plotted `X` value is scaled entropy and their plotted `Y` value is the selected temperature coordinate; the exported `TemperatureSI` value remains absolute kelvin.

`/luacoolprop/process/TS/entropy scale`=⟨*number*⟩ (no default)

Default: `0.001`.

Multiplies process endpoint and path entropy coordinates.

Example: `entropy scale=0.001`.

Accepted alias: `s scale` forwards to `entropy scale`.

`/luacoolprop/process/TS/temperature scale`=⟨*number*⟩ (no default)

Default: `1`.

Multiplies process endpoint and path temperature coordinates.

Example: `temperature scale=1`.

Accepted alias: `t scale` forwards to `temperature scale`.

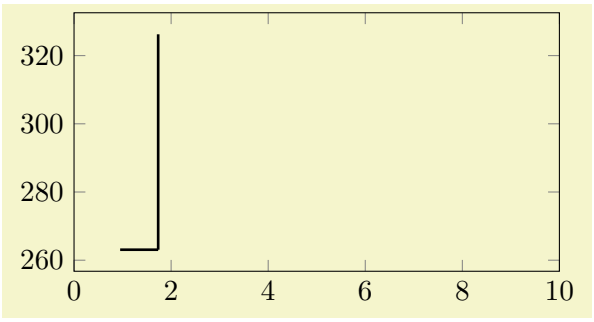
`/luacoolprop/process/TS/temperature weight`=⟨*positive number*⟩ (no default)

Default: `0.01`.

Controls the temperature contribution to adaptive TS process sampling.

Example: `temperature weight=0.02`.

Chain TS processes through exported SI entropy




```
\begin{tikzpicture}
\begin{axis}[width=8cm,height=5cm,auto node placement,xmin=0,xmax=10,]
\LCPAddTSProcess[type=isobar,pressure=2bar,
  from={Q=0},to={Q=1},export coordinates=EvapTS]
\LCPAddTSProcess[type=isentrope,
  from={p=2bar,Q=1},
  to={p=12bar,s=\EvapTSToEntropySI}]
\end{axis}
\end{tikzpicture}
```

For TS, an isobar is parameterised by entropy. Consequently, an evaporation or condensation process crosses the two-phase region as the physically exact horizontal saturation plateau instead of jumping between separately evaluated single-phase states. Isochores are parameterised by temperature; isenthalps and qualities retain logarithmic-pressure sampling. These choices affect only the path sampler, not the public state syntax.

29.7 Plain TeX and ConTeXt

The TS commands belong to the generic layer, so their spelling and semantics are identical in every supported format:

Plain LuaTeX TS diagram

```
\input p-luacoolprop.tex
\pgfplotsset{compat=1.18}
\LCPTSDiagram[fluid=R134a,isenthalp=true]
\bye
```

ConTeXt TS diagram

```
\usemodule[t][luacoolprop]
\pgfplotsset{compat=1.18}
\starttext
\LCPTSDiagram[fluid=R134a,isenthalp=true]
\stoptext
```

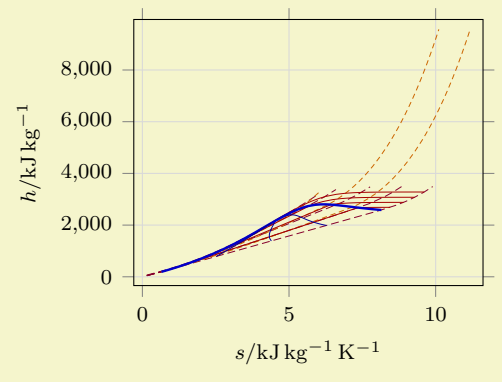
Only LaTeX uses siunitx to format generated numerical labels and axis units. Plain TeX and ConTeXt use the generic math fallbacks and do not acquire a LaTeX dependency.

30 HS enthalpy–entropy diagrams

An HS, or Mollier, diagram places mass-specific entropy on the horizontal axis and mass-specific enthalpy on the vertical axis. The default scales convert CoolProp's SI results to $\mathrm{kJ\,kg^{-1}\,K^{-1}}$ and $\mathrm{kJ\,kg^{-1}}$; both axes are linear. The implementation retains the complete SI thermodynamic state on every sampled point, so the same curve records can support drawing, intersections, and exported process coordinates without a second property model.

Four independently selectable backgrounds are available: constant quality, constant mass-specific volume, constant temperature, and constant pressure. All are calculated from the selected CoolProp fluid. Constant-quality curves share one exact quality-independent critical limit. A subcritical isotherm includes both saturated endpoints and its straight two-phase mixture segment. Isobars are parameterised by entropy rather than by temperature; this makes evaporation and condensation continuous through the two-phase region. No water-specific coordinates or fitted curve data are stored in the package.

A Water HS diagram with four families



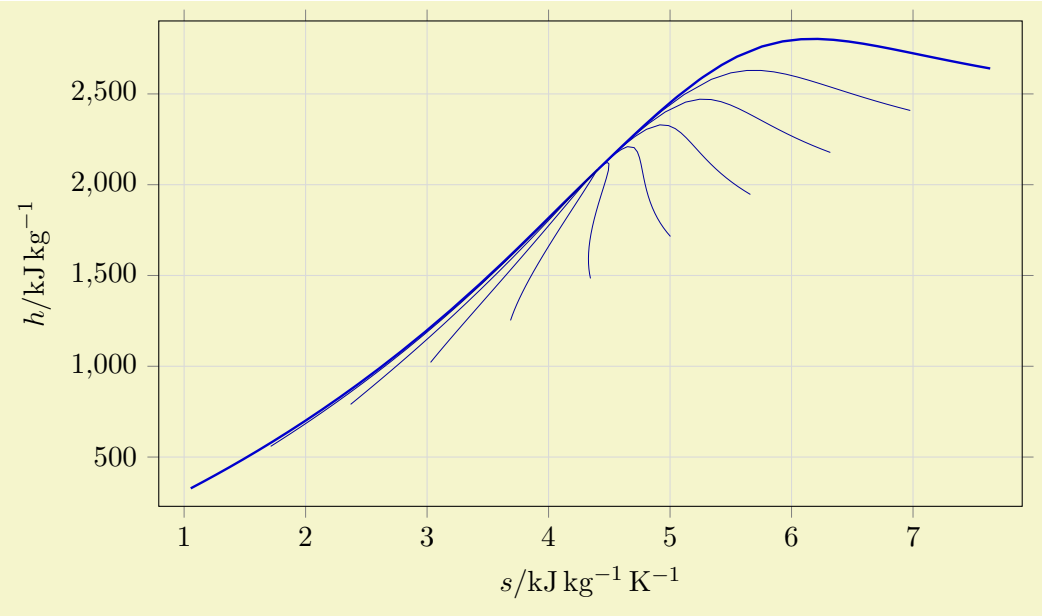
```
\LCPHSDiagram[
  fluid=Water,
  pressure min=10000,
  pressure max=30000000,
  quality values={0,0.25,0.5,0.75,1},
  isochore=true,
  specific volume values={0.002,0.01,0.1,1},
  isotherm=true,
  temperature values={100,200,300,400},
  isobar=true,
  isobar values={0.1,1,10,100,250},
  isobar temperature min=10,
  isobar temperature max=500,
  labels=false,
  axis options={width=6.2cm,height=5.1cm,
    font=\scriptsize}
]
```

30.1 HS commands

\LCPHSDiagram[⟨*options*⟩]

Creates the TikZ picture and a complete linear HS axis, derives padded bounds from the generated data, and draws every enabled family. It is the convenient form of \LCPDiagram{HS}.

A compact Water saturation dome

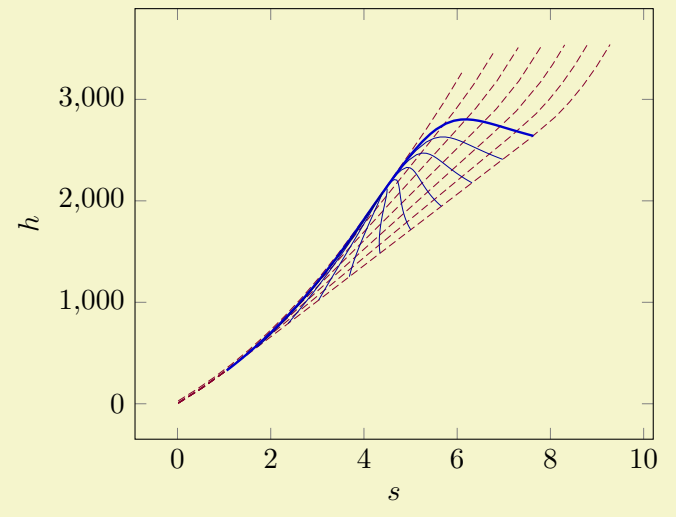



```
\LCPHSDiagram[fluid=Water,
  quality preset=boundaries]
```

\LCPAddHSPlots[⟨*options*⟩]

Adds all enabled HS families to an existing PGFPlots axis. The document owns the axis limits and presentation in this low-level form.

Add an HS background to a custom axis

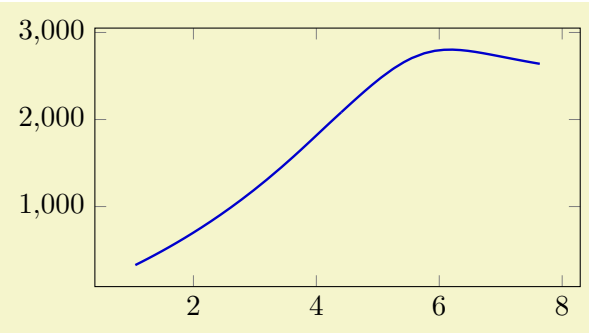



```
\begin{tikzpicture}
\begin{axis}[xlabel={$s$},ylabel={$h$}]
  \LCPAddHSPlots[fluid=Water,isobar=true]
\end{axis}
\end{tikzpicture}
```

\LCPAddHSQuality[⟨*options*⟩]

Adds only constant-quality curves. Their stable paths are named `lcp-hs-q-<value>`; for example, $Q = 0.5$ gives `lcp-hs-q-0p5`. The approach to the critical state is adaptively resolved in logarithmic reduced pressure, using the shared `tolerance` and `max depth` settings with the HS coordinate-error metric.

Only the Water saturation boundaries

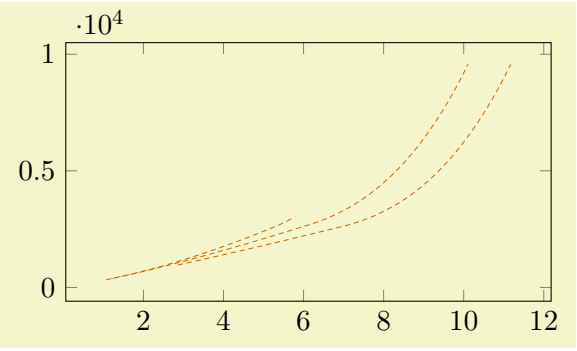



```
\begin{tikzpicture}
\begin{axis}[width=8cm,height=5cm,auto node placement,]
\LCPAddHSQuality[fluid=Water,
  quality values={0,1}]
\end{axis}
\end{tikzpicture}
```

\LCPAddHSIsochores[⟨*options*⟩]

Adds only constant-mass-specific-volume curves. Values and labels obey `specific volume unit`; the default is $\mathrm{m^3\,kg^{-1}}$.

Three Water isochores

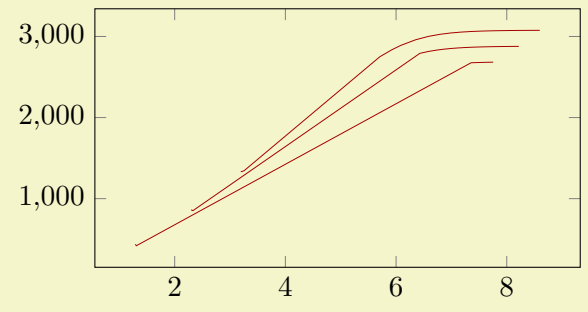



```
\begin{tikzpicture}
\begin{axis}[width=8cm,height=5cm,auto node placement,]
\LCPAddHSIsochores[fluid=Water,
  specific volume values={0.01,0.1,1}]
\end{axis}
\end{tikzpicture}
```

\LCPAddHSIsotherms[⟨*options*⟩]

Adds only constant-temperature curves. Input and labels obey `temperature unit`, which defaults to degrees Celsius.

Selected Water isotherms

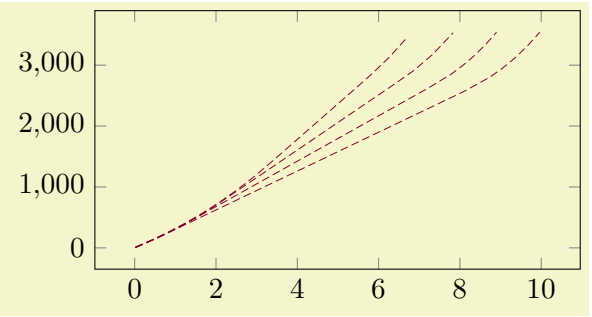



```
\begin{tikzpicture}
\begin{axis}[width=8cm,height=5cm,auto node placement,]
\LCPAddHSIsotherms[fluid=Water,
  temperature values={100,200,300}]
\end{axis}
\end{tikzpicture}
```

\LCPAddHSIsobars[⟨*options*⟩]

Adds only constant-pressure curves. Their stable names include their displayed value and unit: with the defaults, `isobar values={10}` creates `lcp-hs-p-10-bar`.

Selected Water isobars

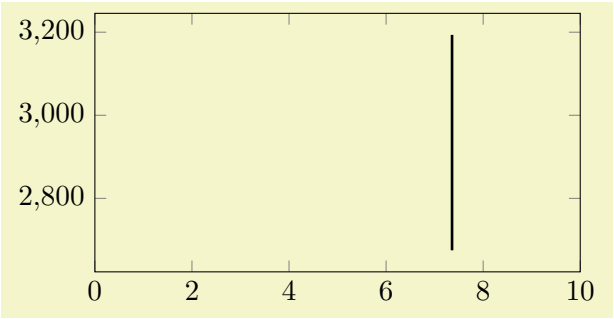



```
\begin{tikzpicture}
\begin{axis}[width=8cm,height=5cm,auto node placement,]
\LCPAddHSIsobars[fluid=Water,
  isobar values={0.1,1,10,100}]
\end{axis}
\end{tikzpicture}
```

\LCPAddHSProcess[⟨*options*⟩]

Adds one thermodynamic path to the current HS axis. It accepts the common state language and the process types `isobar`, `isotherm`, `isentropic`/`isentrope`, `isenthalpic`/ `isenthalp`, `isochore`, and `quality`. Isentropes are exactly vertical and isenthalps exactly horizontal in HS coordinates.

An isentropic Water compression

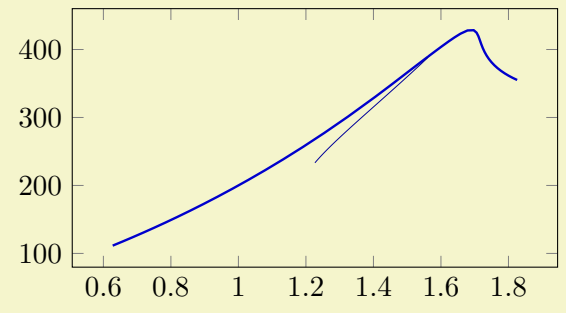



```
\begin{tikzpicture}
\begin{axis}[width=8cm,height=5cm,auto node placement,xmin=0,xmax=10,]
\LCPAddHSProcess[fluid=Water,type=isentrope,
  from={p=1bar,Q=1},to={p=10bar}]
\end{axis}
\end{tikzpicture}
```

\LCPAddHSIsoQuality[⟨*options*⟩]

Alternate spelling for `\LCPAddHSQuality`; the latter is canonical.

Alternate spelling




```
\begin{tikzpicture}
\begin{axis}[width=8cm,height=5cm,auto node placement,]
\LCPAddHSIsoQuality[quality values={0,0.5,1}]
\end{axis}
\end{tikzpicture}
```

The generic forms are equivalent and make coordinate selection data-driven:

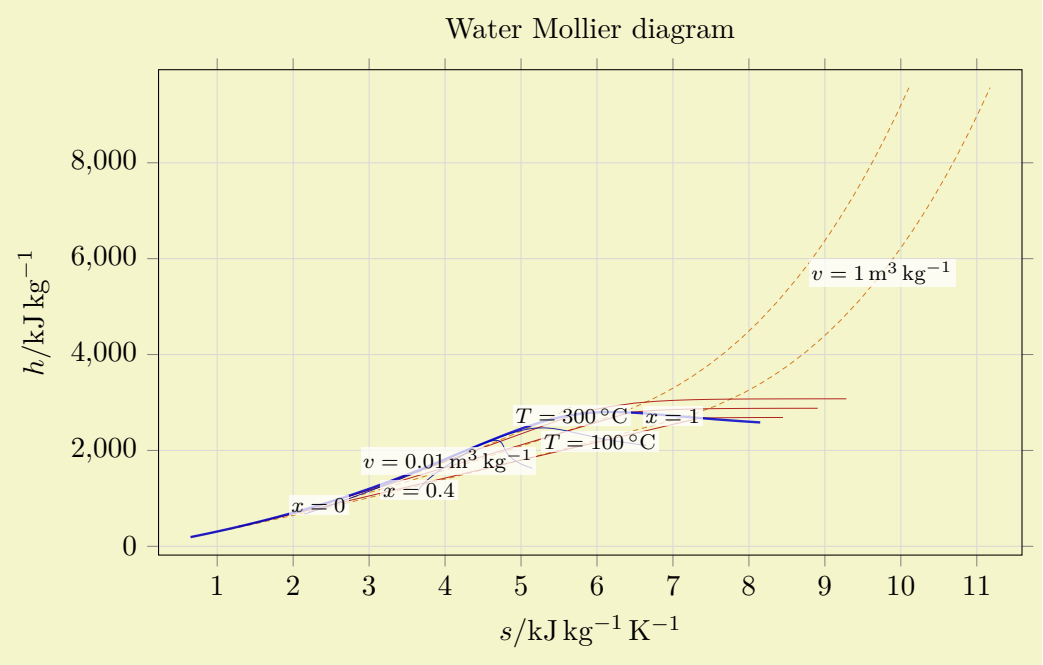
Generic HS dispatch

```
\LCPDiagram{HS}[<options>]
\LCPAddDiagramPlots{HS}[<options>]
\LCPAddDiagramFamily{HS}{quality}[<options>]
\LCPAddDiagramFamily{HS}{isochore}[<options>]
\LCPAddDiagramFamily{HS}{isotherm}[<options>]
\LCPAddDiagramFamily{HS}{isobar}[<options>]
\LCPAddDiagramProcess{HS}[<options>]
```

30.2 Shared vocabulary and physical ranges

The canonical namespace is `/luacoolprop/diagram/HS`. It inherits the shared vocabulary for the fluid and library, pressure range, quality grid, temperature grid, specific-volume grid, thermodynamic symbols, adaptive sampling, labels, pgfplots-autonode placement, curve styles, per-curve overrides, legends, and `axis options`. HS owns both coordinate-unit choices: entropy horizontally and enthalpy vertically.

Shared controls in an HS diagram




```
\LCPHSDiagram[
  fluid=Water,
  pressure min=10000,
  pressure max factor=1.02,
  quality symbol=x,
  quality step=0.2,
  isotherm=true,
  temperature unit=celsius,
  temperature values={100,200,300},
  isochore=true,
  specific volume values={0.01,0.1,1},
  labels=true,
  label placement=autonode,
  curve style={opacity=.85},
  axis options={title={Water Mollier diagram}}
]
```

The `pressure min/max` interval bounds quality, isochore, and isotherm sampling, and supplies the default isobar grid. It should remain inside the fluid equation-of-state validity range. An explicitly requested curve may still have fewer visible points when CoolProp reports that part of the state domain as invalid; invalid individual samples are omitted, while a wholly invalid interval yields no curve. Valid components on opposite sides of an invalid interval are never joined; select `domain policy=warning` to audit omissions.

30.3 HS switches, scale, and error metric

`/luacoolprop/diagram/HS/isobar`=⟨*boolean*⟩ (no default)

Default: `false`.

Enables the constant-pressure background family.

Example: `isobar=true`.

Accepted alias: `isobars` forwards to `isobar`.

`/luacoolprop/diagram/HS/entropy scale`=⟨*number*⟩ (no default)

Default: `0.001`.

Advanced raw multiplier for horizontal SI entropy; it produces a scaled-SI label.

Example: `entropy scale=0.002`.

Accepted alias: `s scale` forwards to `entropy scale`.

`/luacoolprop/diagram/HS/entropy axis unit`=⟨*jkgk or kjkgk*⟩ (no default)

Default: `kjkgk`.

Atomically selects the horizontal entropy conversion and label.

Example: `entropy axis unit=jkgk`.

`/luacoolprop/diagram/HS/enthalpy axis unit`=⟨*jkg or kjkg*⟩ (no default)

Default: `kjkg`.

Atomically selects the vertical enthalpy conversion and label.

Example: `enthalpy axis unit=jkg`.

`/luacoolprop/diagram/HS/enthalpy weight`=⟨*positive number*⟩ (no default)

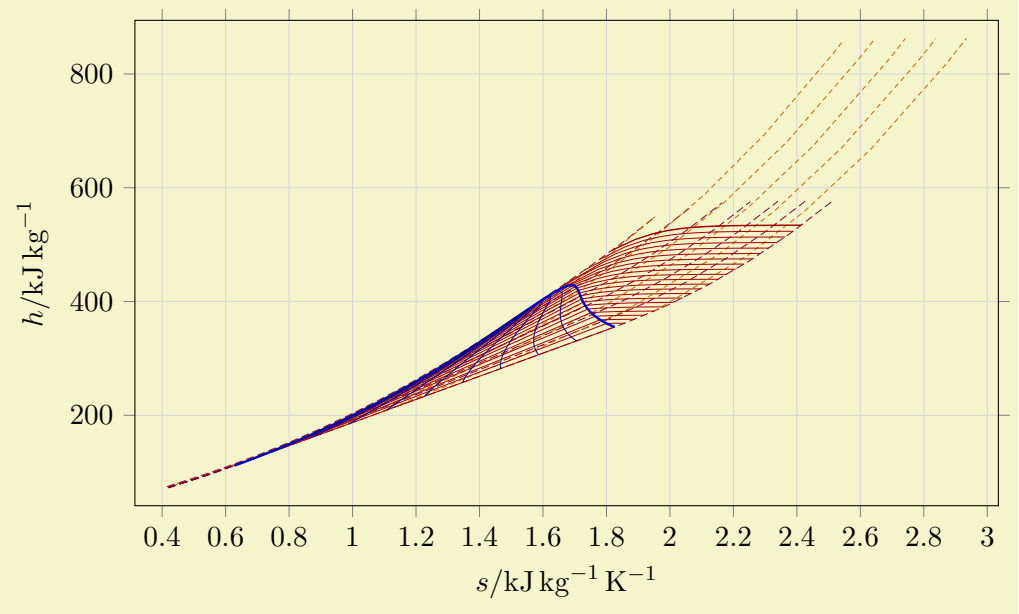
Default: `0.01`.

Weights vertical enthalpy error in the adaptive midpoint test. Increasing it inserts more samples where the HS curve bends sharply.

Example: `enthalpy weight=0.02`.

The other declared HS switches select their corresponding families:

Every HS family switch




```
\LCPHSDiagram[quality=true,isochore=true,
  isotherm=true,isobar=true]
```

Their grids, styles, labels, units, and symbols use the shared `quality ...`, `specific volume ...`, and `temperature ...` keys documented in the PH reference.

30.4 Isobar grid and units

`/luacoolprop/diagram/HS/isobar unit=`⟨*unit*⟩ (no default)

Default: `bar`.

Controls the input grid, stable path names, and generated pressure labels. Accepted values are `bar`, `kpa`, `mpa`, and `pa` or `si`; all values are converted to pascals before calling CoolProp.

Example: `isobar unit=mpa`.

`/luacoolprop/diagram/HS/isobar min=`⟨*positive number or auto*⟩ (no default)

Default: `auto`.

Lower automatically generated pressure in `isobar unit`. Automatic means the diagram's lower pressure bound.

Example: `isobar min=0.1`.

`/luacoolprop/diagram/HS/isobar max=`⟨*positive number or auto*⟩ (no default)

Default: `auto`.

Upper automatically generated pressure in `isobar unit`.

Example: `isobar max=250`.

`/luacoolprop/diagram/HS/isobar step=`⟨*positive number or auto*⟩ (no default)

Default: `auto`.

Spacing used by a linear grid. With `auto`, `isobar count` determines the grid.

Example: `isobar step=10`.

`/luacoolprop/diagram/HS/isobar mode=`⟨*choice*⟩ (no default)

Default: `log`.

Selects `log`, `linear`, or `list` grid generation. Supplying `isobar values` selects list mode automatically. Logarithmic spacing is usually the most readable choice over a wide pressure range.

Example: `isobar mode=linear`.

`/luacoolprop/diagram/HS/isobar count=`⟨*positive integer*⟩ (no default)

Default: `7`.

Number of automatically spaced isobars when no explicit step is given.

Example: `isobar count=9`.

`/luacoolprop/diagram/HS/isobar values=`⟨*comma list*⟩ (no default)

Default: `empty`.

Explicit constant pressures in `isobar unit`.

Example: `isobar values={0.1,1,10,100,250}`.

`/luacoolprop/diagram/HS/isobar temperature min=`⟨*number or auto*⟩ (no default)

Default: `auto`.

Lower endpoint of each isobar. Numeric values use `temperature unit`; automatic selection stays inside CoolProp's fluid validity range.

Example: `isobar temperature min=10`.

/luacoolprop/diagram/HS/isobar temperature max=⟨*number or auto*⟩ (no default)

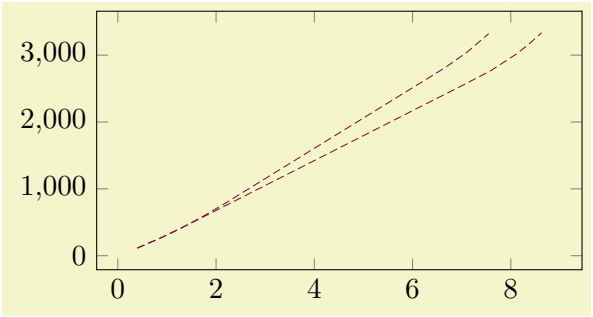
Default: `auto`.

Upper endpoint of each isobar in `temperature unit`.

Example: `isobar temperature max=500`.

For example, this deliberately uses kelvin for both endpoint controls:

Kelvin-bounded isobars




```
\begin{tikzpicture}
\begin{axis}[width=8cm,height=5cm,auto node placement,]
\LCPAddHSIsobars[fluid=Water,
  temperature unit=kelvin,
  isobar values={1,10},
  isobar temperature min=300,
  isobar temperature max=700]
\end{axis}
\end{tikzpicture}
```

30.5 Isobar appearance and automatic labels

/luacoolprop/diagram/HS/isobar labels=⟨*boolean or auto*⟩ (no default)

Default: `auto`.

Overrides the common `labels` switch for the isobar family.

Example: `isobar labels=true`.

/luacoolprop/diagram/HS/isobar label pos=⟨*fraction*⟩ (no default)

Default: `0.68`.

Preferred normalised arc-length position supplied to pgfplots-autonode.

Example: `isobar label pos=0.55`.

/luacoolprop/diagram/HS/isobar label every=⟨*positive integer*⟩ (no default)

Default: `2`.

Offers every nth isobar, and always the final isobar, for labelling.

Example: `isobar label every=1`.

/luacoolprop/diagram/HS/isobar label sloped=⟨*boolean or auto*⟩ (no default)

Default: `auto`.

Aligns each generated pressure label with the local curve tangent.

Example: `isobar label sloped=true`.

`/luacoolprop/diagram/HS/isobar label allow upside down=`⟨*boolean or auto*⟩ (no default)

Default: `auto`.

Permits the local tangent to leave an isobar label upside down.

Example: `isobar label allow upside down=false`.

`/luacoolprop/diagram/HS/isobar label style=`⟨*TikZ style name*⟩ (no default)

Default: `luacoolprop isobar label node`.

Selects the node style attached to generated pressure labels.

Example: `isobar label style=luacoolprop label node`.

`/luacoolprop/diagram/HS/isobar label node style=`⟨*TikZ options*⟩ (no default)

Default: `package default`.

Redefines the dedicated `luacoolprop isobar label node` style in the current TeX scope.

Example: `isobar label node style={fill=violet!10,inner sep=1.5pt}`.

`/luacoolprop/diagram/HS/isobar color=`⟨*colour*⟩ (no default)

Default: `purple!70!black`.

Sets the base colour of every constant-pressure curve.

Example: `isobar color=violet!80!black`.

`/luacoolprop/diagram/HS/isobar style=`⟨*TikZ plot options*⟩ (no default)

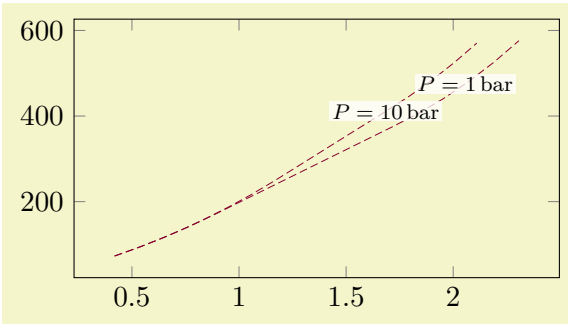
Default: `thin dashed line`.

Appends family-specific drawing options after the colour.

Example: `isobar style={line width=.35pt,densely dashdotted}`.

The common `pressure symbol` key also controls isobar label text:

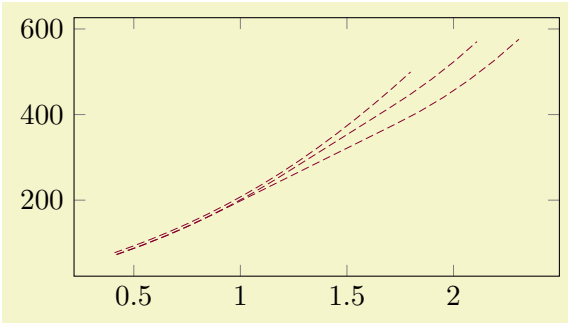
Use a custom pressure symbol




```
\begin{tikzpicture}
\begin{axis}[width=8cm,height=5cm,auto node placement,]
\LCPAddHSIsobars[pressure symbol=P,
  isobar values={1,10},labels=true]
\end{axis}
\end{tikzpicture}
```

Per-curve overrides use the stable isobar name. For label text containing fragile formatting commands such as `\mathrm`, use a protected macro: the text passes through expansion while the overrides are accumulated and serialized. The generic e-TeX `\protected\def` form below keeps that formatting intact until the label is typeset.

Emphasise the ten-bar path




```
\protected\def\BoilerLabel{$p_{\mathrm{boiler}}$}
\begin{tikzpicture}
\begin{axis}[width=8cm,height=5cm,auto node placement,]
\LCPAddHSIsobars[isobar values={1,10,100},
  curve style for={lcp-hs-p-10-bar}{line width=1pt},
  label text for={lcp-hs-p-10-bar}{\BoilerLabel}]
\end{axis}
\end{tikzpicture}
```

30.6 Isobar sampling and HS axis bounds

`/luacoolprop/diagram/HS/isobar initial intervals`=⟨*positive integer*⟩ (no default)

Default: `16`.

Initial entropy intervals before adaptive isobar refinement.

Example: `isobar initial intervals=24`.

`/luacoolprop/diagram/HS/isobar max depth`=⟨*nonnegative integer*⟩ (no default)

Default: `8`.

Maximum recursive midpoint-bisection depth for an isobar.

Example: `isobar max depth=10`.

`/luacoolprop/diagram/HS/isobar tolerance`=⟨*positive number*⟩ (no default)

Default: `0.10`.

Maximum scaled midpoint error accepted while sampling an isobar.

Example: `isobar tolerance=0.06`.

`/luacoolprop/diagram/HS/entropy axis min`=⟨*number or auto*⟩ (no default)

Default: `auto`.

Overrides the automatically padded horizontal minimum after `entropy scale`.

Example: `entropy axis min=0`.

`/luacoolprop/diagram/HS/entropy axis max`=⟨*number or auto*⟩ (no default)

Default: `auto`.

Overrides the horizontal maximum in plotted entropy units.

Example: `entropy axis max=10`.

`/luacoolprop/diagram/HS/enthalpy axis min`=⟨*number or auto*⟩ (no default)

Default: `auto`.

Overrides the automatically padded vertical minimum after `enthalpy scale`.

Example: `enthalpy axis min=0`.

`/luacoolprop/diagram/HS/enthalpy axis max`=⟨*number or auto*⟩ (no default)

Default: `auto`.

Overrides the vertical maximum in plotted enthalpy units.

Example: `enthalpy axis max=4500`.

The common `initial intervals`, `max depth`, and `tolerance` remain fallbacks. Family-specific controls take precedence: isobars use the three keys above, while isotherms and isochores use their inherited `isotherm ...` and `isochore ...` controls. Smaller tolerances generate more CoolProp calls and more TeX coordinates; refine only when the plotted result requires it.

30.7 HS process keys and exported coordinates

The process namespace is `/luacoolprop/process/HS`. Every shared process key applies unchanged: fluid and library; endpoint state pairs and units; process type and conserved value; names, styles, labels, and endpoint markers; adaptive sampling; and `export coordinates`/`log coordinates`. The exported `X` value is scaled entropy and `Y` is scaled enthalpy, while all `...SI` endpoint macros retain SI units.

`/luacoolprop/process/HS/entropy scale`=⟨*number*⟩ (no default)

Default: `0.001`.

Multiplies entropy coordinates along an HS process path.

Example: `entropy scale=0.001`.

Accepted alias: `s scale` forwards to `entropy scale`.

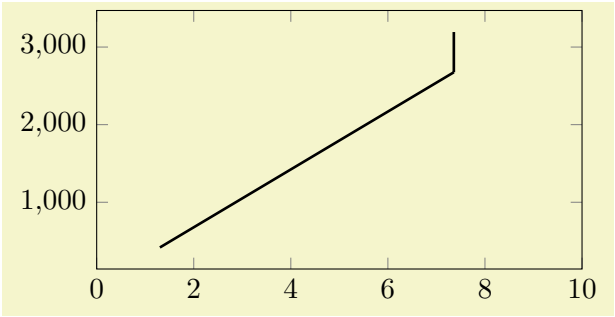
`/luacoolprop/process/HS/enthalpy weight`=⟨*positive number*⟩ (no default)

Default: `0.01`.

Controls the vertical enthalpy contribution to adaptive HS process refinement.

Example: `enthalpy weight=0.02`.

Chain HS processes through exported entropy




```
\begin{tikzpicture}
\begin{axis}[width=8cm,height=5cm,auto node placement,xmin=0,xmax=10,]
\LCPAddHSProcess[fluid=Water,type=isobar,
  from={p=1bar,Q=0},to={p=1bar,Q=1},
  export coordinates=EvapHS,log coordinates=true]
\LCPAddHSProcess[fluid=Water,type=isentrope,
  from={p=1bar,Q=1},
  to={p=10bar,s=\EvapHSToEntropySI}]
% Plot coordinates: \EvapHSFromX, \EvapHSFromY,
%                   \EvapHSToX,   \EvapHSToY.
% Raw SI results:   \EvapHSToEntropySI, \EvapHSToEnthalpySI.
\end{axis}
\end{tikzpicture}
```

For an isobar, entropy is the sampling parameter and the saturation mixture is therefore represented continuously. Isotherms include their exact mixture segment; isochores are parameterised by temperature; constant-quality paths use logarithmic pressure. The choice of sampler is an implementation detail: the public endpoint language stays identical across PH, PV, TS, HS, and PT.

30.8 Plain TeX and ConTeXt

HS is implemented entirely in the generic layer. The same API is therefore available through both non-LaTeX wrappers:

Plain LuaTeX HS diagram

```
\input p-luacoolprop.tex
\pgfplotsset{compat=1.18}
\LCPHSDiagram[fluid=Water,isobar=true,
  isobar values={1,10,100}]
\bye
```

ConTeXt HS diagram

```
\usemodule[t][luacoolprop]
\pgfplotsset{compat=1.18}
\starttext
\LCPHSDiagram[fluid=Water,isotherm=true,
  temperature values={100,200,300}]
\stoptext
```

Only the LaTeX wrapper uses siunitx for numeric labels and units. It neither resets nor changes an existing siunitx configuration. Plain TeX and ConTeXt use the generic math fallback supplied by `luacoolprop.tex`.

31 PT pressure–temperature diagrams

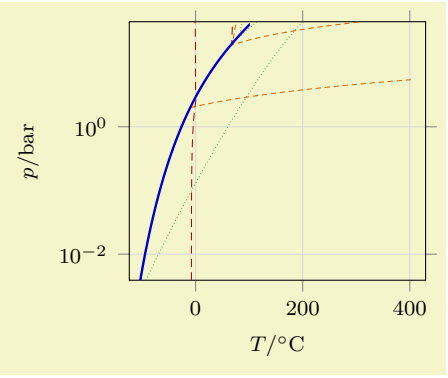
A PT diagram places absolute thermodynamic temperature on the horizontal axis and pressure on the vertical axis. The high-level renderer uses a linear temperature scale and a logarithmic pressure scale. With the default bounds, the vertical axis extends from the fluid's triple-point pressure to $1.10p_c$, while the liquid–vapour coexistence curve itself ends exactly at the critical point.

For a pure substance, saturated liquid and saturated vapour at equilibrium have the same T and p. Consequently, PT contains one coexistence locus, not separate bubble and dew curves and not a quality grid. The starting record uses CoolProp's exact (T_t, p_t) constants; the final record uses the exact (T_c, p_c) constants. Vapour quality is undefined at the critical point and cannot be read anywhere from PT coordinates alone.

Information lost by the PT projection

Every two-phase state at a given saturation pressure maps to the same PT point, independently of its quality. A PT curve therefore establishes phase equilibrium and operating pressure/temperature, but it cannot replace a PH, PV, TS, or HS diagram when quality, latent enthalpy, entropy, or specific volume must be read graphically. Isentropes, isenthalps, and isochores may coincide with part of the coexistence locus while they traverse a two-phase domain; this is the correct projection, not a sampling defect.

A complete R134a PT background



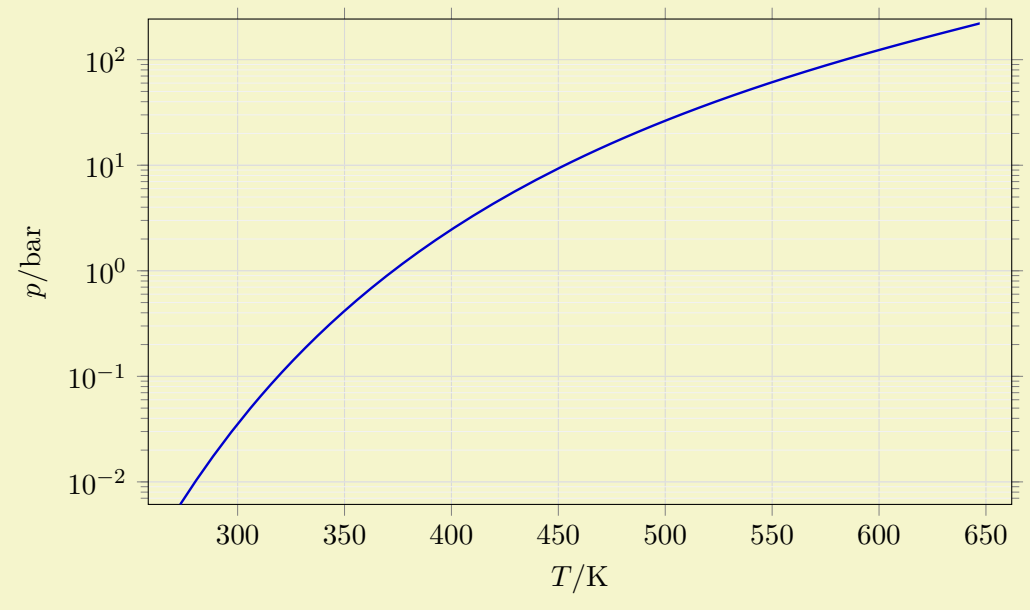

```
\LCPPTDiagram[
  fluid=R134a,
  temperature axis unit=celsius,
  pressure axis unit=bar,
  isentrope=true,
  entropy values={1.4,1.7,2.0},
  isenthalp=true,
  enthalpy values={200,300,400},
  isochore=true,
  specific volume values={0.001,0.01,0.1},
  axis options={width=5.5cm,height=5.0cm,
    font=\scriptsize}
]
```

31.1 PT commands

\LCPPTDiagram[⟨*options*⟩]

Creates a TikZ picture and a PT axis, applies the coupled unit labels, obtains the automatic fluid bounds, and draws all enabled families.

Minimal PT diagram

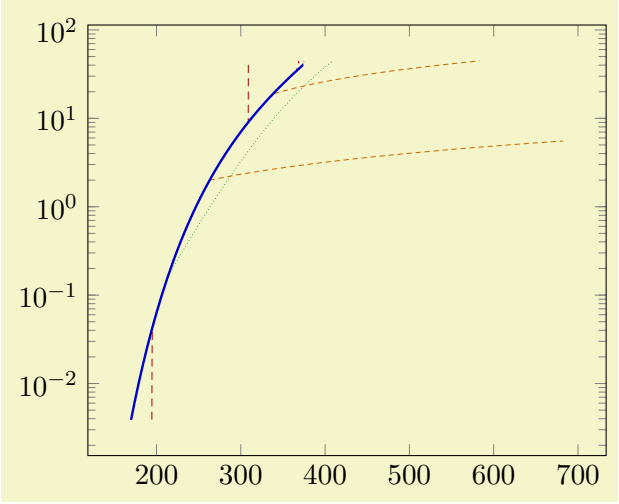



```
\LCPPTDiagram[fluid=Water]
```

\LCPAddPTPlots[⟨*options*⟩]

Adds all enabled PT families inside an existing axis. The caller must supply the linear-temperature/log-pressure axis and labels.

All enabled families in an existing axis

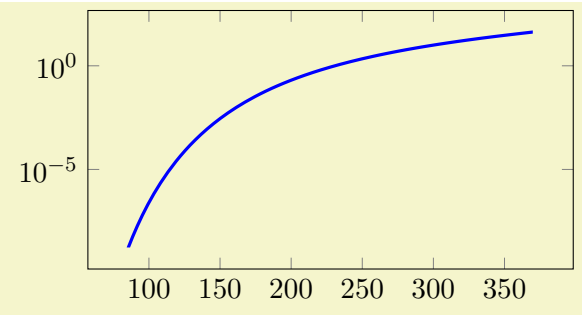



```
\begin{tikzpicture}
\begin{axis}[ymode=log]
  \LCPAddPTPlots[phase envelope=true,
    isentrope=true,entropy values={1.5,1.8},
    isenthalp=true,enthalpy values={250,350},
    isochore=true,specific volume values={0.01,0.1}]
\end{axis}
\end{tikzpicture}
```

\LCPAddPTPhaseEnvelope[⟨*options*⟩]

Adds only the triple-to-critical liquid–vapour equilibrium curve.

Coexistence curve only



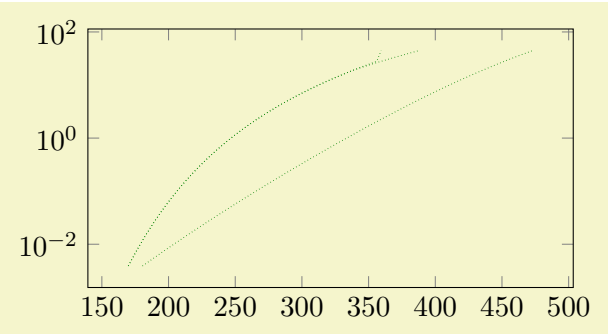

```
\begin{tikzpicture}
\begin{axis}[width=8cm,height=5cm,auto node placement,ymode=log]
\LCPAddPTPhaseEnvelope[fluid=Propane,
  phase envelope color=blue,phase envelope style={very thick}]
\end{axis}
\end{tikzpicture}
```

Accepted alias: \LCPAddPTSaturation.
Target: \LCPAddPTPhaseEnvelope.
Example: \LCPAddPTSaturation[fluid=R134a].

\LCPAddPTIsentropes[⟨*options*⟩]

Adds only constant-mass-specific-entropy curves.

Selected PT isentropes

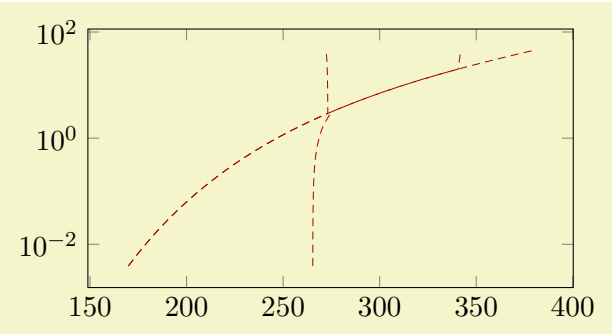



```
\begin{tikzpicture}
\begin{axis}[width=8cm,height=5cm,auto node placement,ymode=log]
\LCPAddPTIsentropes[entropy values={1.4,1.7,2.0}]
\end{axis}
\end{tikzpicture}
```

\LCPAddPTIsenthalps[⟨*options*⟩]

Adds only constant-mass-specific-enthalpy curves.

Selected PT isenthalps

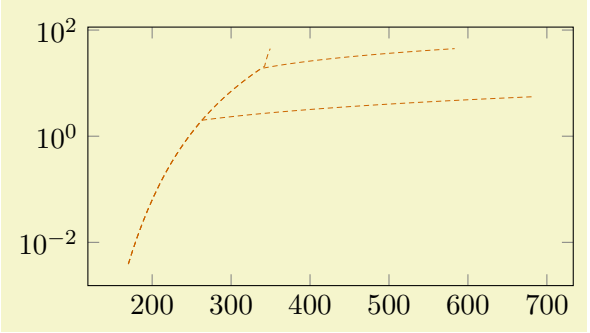



```
\begin{tikzpicture}
\begin{axis}[width=8cm,height=5cm,auto node placement,ymode=log]
\LCPAddPTIsenthalps[enthalpy values={200,300,400}]
\end{axis}
\end{tikzpicture}
```

\LCPAddPTIsochores[⟨*options*⟩]

Adds only constant-mass-specific-volume curves.

Selected PT isochores

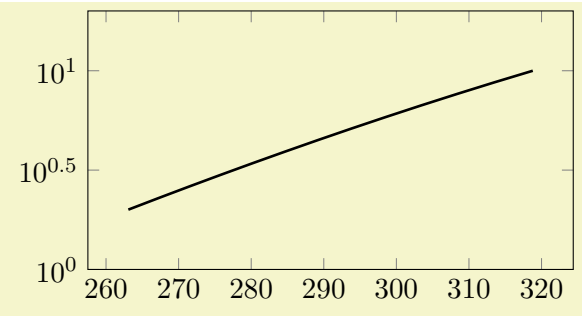


```
\begin{tikzpicture}
\begin{axis}[width=8cm,height=5cm,auto node placement,ymode=log]
\LCPAddPTIsochores[specific volume values={0.001,0.01,0.1}]
\end{axis}
\end{tikzpicture}
```

\LCPAddPTProcess[⟨*options*⟩]

Projects any of the six shared conserved-property process kinds onto PT.

An isentropic compression in PT coordinates




```
\begin{tikzpicture}
\begin{axis}[width=8cm,height=5cm,auto node placement,ymin=1,ymax=20,ymode=log]
\LCPAddPTProcess[type=isentrope,
  from={p=2bar,quality=1},to={p=10bar},
  name=pt-compression,export coordinates=PTCompression]
\end{axis}
\end{tikzpicture}
```

The generic equivalents are listed below. The family dispatcher also accepts `saturation` and `equilibrium` on the Lua side; the canonical public family name is `phase_envelope`.

- `\LCPDiagram{PT}`;
- `\LCPAddDiagramPlots{PT}`;
- `\LCPAddDiagramFamily{PT}{phase_envelope}`; and
- `\LCPAddDiagramProcess{PT}`.

31.2 Families, ranges, and coordinate units

PT options use `/luacoolprop/diagram/PT`; the lowercase `/luacoolprop/pt diagram` path is an alternate public spelling.

/luacoolprop/diagram/PT/fluid=⟨*CoolProp pure-fluid identifier*⟩ (no default)

Default: `global fluid`.

Selects the pure fluid used by every PT family.

Example: `fluid=Water`.

/luacoolprop/diagram/PT/library=⟨*file path*⟩ (no default)

Default: `global library`.

Selects an external CoolProp shared library for this diagram.

Example: `library=/opt/coolprop/libCoolProp.dylib`.

`/luacoolprop/diagram/PT/reference state`=⟨*DEF, IIR, ASHRAE, or NBP*⟩ (no default)

Default: `DEF`.

Selects the locked enthalpy/entropy origin used by isenthalps and isentropes.

Example: `reference state=DEF`.

`/luacoolprop/diagram/PT/phase envelope`=⟨*boolean*⟩ (no default)

Default: `true`.

Enables the single liquid–vapour equilibrium curve.

Example: `phase envelope=false`.

Accepted alias: `saturation` forwards to `phase envelope`.

Accepted alias: `equilibrium curve` forwards to `phase envelope`.

`/luacoolprop/diagram/PT/isentrope`=⟨*boolean*⟩ (no default)

Default: `false`.

Enables constant-mass-specific-entropy curves.

Example: `isentrope=true`.

Accepted alias: `isentropes` forwards to `isentrope`.

`/luacoolprop/diagram/PT/isenthalp`=⟨*boolean*⟩ (no default)

Default: `false`.

Enables constant-mass-specific-enthalpy curves.

Example: `isenthalp=true`.

Accepted alias: `isenthalps` forwards to `isenthalp`.

Accepted alias: `enthalpy curves` forwards to `isenthalp`.

`/luacoolprop/diagram/PT/isochore`=⟨*boolean*⟩ (no default)

Default: `false`.

Enables constant-mass-specific-volume curves.

Example: `isochore=true`.

Accepted alias: `isochores` forwards to `isochore`.

`/luacoolprop/diagram/PT/pressure min`=⟨*positive number or auto*⟩ (no default)

Default: `auto`.

Sets the axis minimum in Pa; automatic PT bounds use p_t. An explicit value also clips every sampled family.

Example: `pressure min=10000`.

`/luacoolprop/diagram/PT/pressure max`=⟨*positive number or auto*⟩ (no default)

Default: `auto`.

Sets the axis maximum in Pa; automatic PT bounds use $1.10p_c$.

Example: `pressure max=5E6`.

`/luacoolprop/diagram/PT/pressure max factor=`⟨*positive number*⟩ (no default)

Default: `1.10`.

Sets the automatic pressure maximum as a multiple of p_c.

Example: `pressure max factor=1.25`.

`/luacoolprop/diagram/PT/temperature axis min=`⟨*number or auto*⟩ (no default)

Default: `auto`.

Overrides the sampled horizontal minimum in the selected temperature axis unit.

Example: `temperature axis min=-100`.

`/luacoolprop/diagram/PT/temperature axis max=`⟨*number or auto*⟩ (no default)

Default: `auto`.

Overrides the sampled horizontal maximum in the selected temperature axis unit.

Example: `temperature axis max=150`.

`/luacoolprop/diagram/PT/temperature axis unit=`⟨*kelvin or celsius*⟩ (no default)

Default: `kelvin`.

Changes both horizontal coordinates and the axis label. Celsius applies $T_{\mathrm{plot}} = T_{\mathrm{SI}} - 273.15$, not a multiplicative approximation.

Example: `temperature axis unit=celsius`.

`/luacoolprop/diagram/PT/pressure axis unit=`⟨*pa, kpa, mpa, or bar*⟩ (no default)

Default: `bar`.

Changes both vertical coordinates and the generated pressure-axis label.

Example: `pressure axis unit=mpa`.

`/luacoolprop/diagram/PT/temperature scale=`⟨*number*⟩ (no default)

Default: `global scale`.

Provides an advanced raw multiplier with zero offset and an explicit scaled-SI axis label.

Example: `temperature scale=1`.

Accepted alias: `t scale` forwards to `temperature scale`.

`/luacoolprop/diagram/PT/pressure scale=`⟨*number*⟩ (no default)

Default: `global scale`.

Provides an advanced raw multiplier from Pa to plotted pressure.

Example: `pressure scale=1E-5`.

Accepted alias: `p scale` forwards to `pressure scale`.

`/luacoolprop/diagram/PT/temperature symbol=`⟨*TeX math material*⟩ (no default)

Default: `T`.

Changes the horizontal-axis symbol without changing the CoolProp property.

Example: `temperature symbol={\theta }`.

/luacoolprop/diagram/PT/pressure symbol=⟨*TeX math material*⟩ (no default)

Default: p.

Changes the vertical-axis symbol.

Example: pressure symbol={P}.

/luacoolprop/diagram/PT/enthalpy symbol=⟨*TeX math material*⟩ (no default)

Default: h.

Changes generated isenthalp labels.

Example: enthalpy symbol={i}.

/luacoolprop/diagram/PT/entropy symbol=⟨*TeX math material*⟩ (no default)

Default: s.

Changes generated isentrope labels.

Example: entropy symbol={\sigma }.

/luacoolprop/diagram/PT/specific volume symbol=⟨*TeX math material*⟩ (no default)

Default: v.

Changes generated isochore labels.

Example: specific volume symbol={\nu }.

The entropy and specific-volume family grids use the complete shared keys documented in Section 26: entropy unit/min/max/step/mode/count/preset/values and specific volume unit/min/max/step/mode/count/preset/values, including their s ... and v ... aliases. Values are converted to SI before calling CoolProp.

31.3 Enthalpy grid and isenthalp presentation

/luacoolprop/diagram/PT/enthalpy unit=⟨*kjkg, jkg, or si*⟩ (no default)

Default: kjkg.

Sets the unit used by the grid and generated labels.

Example: enthalpy unit=jkg.

Accepted alias: h unit forwards to enthalpy unit.

/luacoolprop/diagram/PT/enthalpy min=⟨*number or auto*⟩ (no default)

Default: auto.

Sets the lower automatic-grid bound.

Example: enthalpy min=200.

/luacoolprop/diagram/PT/enthalpy max=⟨*number or auto*⟩ (no default)

Default: auto.

Sets the upper automatic-grid bound.

Example: enthalpy max=450.

/luacoolprop/diagram/PT/enthalpy step=⟨*positive number or auto*⟩ (no default)

Default: auto.

Sets the linear grid spacing.

Example: enthalpy step=50.

`/luacoolprop/diagram/PT/enthalpy mode`=⟨*linear or list*⟩ (no default)

Default: `linear`.

Selects generated values; an explicit value list implies list mode.

Example: `enthalpy mode=linear`.

`/luacoolprop/diagram/PT/enthalpy count`=⟨*positive integer*⟩ (no default)

Default: `8`.

Sets the requested automatic count when no step is supplied.

Example: `enthalpy count=6`.

`/luacoolprop/diagram/PT/enthalpy values`=⟨*numeric list*⟩ (no default)

Default: `empty`.

Supplies an explicit grid in `enthalpy unit`.

Example: `enthalpy values={200,300,400}`.

Accepted alias: `h min` forwards to `enthalpy min`.
Accepted alias: `h max` forwards to `enthalpy max`.
Accepted alias: `h step` forwards to `enthalpy step`.
Accepted alias: `h mode` forwards to `enthalpy mode`.
Accepted alias: `h count` forwards to `enthalpy count`.
Accepted alias: `h values` forwards to `enthalpy values`.

`/luacoolprop/diagram/PT/isenthalp labels`=⟨*boolean or auto*⟩ (no default)

Default: `auto`.

Overrides the shared `labels` switch for isenthalps.

Example: `isenthalp labels=true`.

`/luacoolprop/diagram/PT/isenthalp label pos`=⟨*fraction*⟩ (no default)

Default: `0.62`.

Sets the preferred autonode arc-length position.

Example: `isenthalp label pos=0.7`.

`/luacoolprop/diagram/PT/isenthalp label every`=⟨*positive integer*⟩ (no default)

Default: `2`.

Labels one isenthalp out of every n.

Example: `isenthalp label every=1`.

`/luacoolprop/diagram/PT/isenthalp label sloped`=⟨*boolean or auto*⟩ (no default)

Default: `auto`.

Overrides the common rotation policy.

Example: `isenthalp label sloped=true`.

`/luacoolprop/diagram/PT/isenthalp label allow upside down`=⟨*boolean or auto*⟩ (no default)

Default: `auto`.

Overrides the common tangent-orientation policy.

Example: `isenthalp label allow upside down=false`.

/luacoolprop/diagram/PT/isenthalp label style=⟨*TikZ style*⟩ (no default)

Default: `luacoolprop isenthalp label node`.

Selects the generated label-node style.

Example: `isenthalp label style=luacoolprop isenthalp label node`.

/luacoolprop/diagram/PT/isenthalp label node style=⟨*TikZ node options*⟩ (no default)

Default: `package style`.

Redefines the PT isenthalp label style.

Example: `isenthalp label node style={fill=red!8,text=red!70!black}`.

/luacoolprop/diagram/PT/isenthalp color=⟨*colour*⟩ (no default)

Default: `red!65!black`.

Sets the isenthalp colour.

Example: `isenthalp color=purple`.

/luacoolprop/diagram/PT/isenthalp style=⟨*TikZ plot options*⟩ (no default)

Default: `thin, densely dashed`.

Sets the isenthalp drawing style.

Example: `isenthalp style={line width=.4pt,dashed}`.

31.4 Phase-envelope presentation and sampling

/luacoolprop/diagram/PT/phase envelope labels=⟨*boolean or auto*⟩ (no default)

Default: `auto`.

Overrides `labels` for the coexistence curve.

Example: `phase envelope labels=true`.

/luacoolprop/diagram/PT/phase envelope label text=⟨*TeX text*⟩ (no default)

Default: `liquid–vapour equilibrium`.

Sets the coexistence-curve label.

Example: `phase envelope label text={saturation}`.

/luacoolprop/diagram/PT/phase envelope label pos=⟨*fraction*⟩ (no default)

Default: `0.45`.

Sets the preferred autonode position.

Example: `phase envelope label pos=0.6`.

/luacoolprop/diagram/PT/phase envelope label sloped=⟨*boolean or auto*⟩ (no default)

Default: `auto`.

Overrides label rotation.

Example: `phase envelope label sloped=true`.

/luacoolprop/diagram/PT/phase envelope label allow upside down=⟨*boolean or auto*⟩ (no default)

Default: `auto`.

Overrides tangent orientation.

Example: `phase envelope label allow upside down=false`.

/luacoolprop/diagram/PT/phase envelope label style=⟨*TikZ style*⟩ (no default)

Default: `luacoolprop phase envelope label node`.

Selects the label style.

Example: `phase envelope label style=luacoolprop phase envelope label node`.

/luacoolprop/diagram/PT/phase envelope label node style=⟨*TikZ node options*⟩ (no default)

Default: `package style`.

Redefines the label-node style.

Example: `phase envelope label node style={fill=blue!8,text=blue!70!black}`.

/luacoolprop/diagram/PT/phase envelope color=⟨*colour*⟩ (no default)

Default: `blue!80!black`.

Sets the coexistence-curve colour.

Example: `phase envelope color=black`.

/luacoolprop/diagram/PT/phase envelope style=⟨*TikZ plot options*⟩ (no default)

Default: `line width=0.9pt`.

Sets its drawing style.

Example: `phase envelope style={very thick}`.

/luacoolprop/diagram/PT/phase envelope initial intervals=⟨*positive integer*⟩ (no default)

Default: `20`.

Sets the initial logarithmic-pressure subdivision.

Example: `phase envelope initial intervals=28`.

/luacoolprop/diagram/PT/phase envelope max depth=⟨*nonnegative integer*⟩ (no default)

Default: `9`.

Limits recursive adaptive refinement.

Example: `phase envelope max depth=10`.

/luacoolprop/diagram/PT/phase envelope tolerance=⟨*positive number*⟩ (no default)

Default: `0.08`.

Sets the midpoint error tolerance in PT display coordinates.

Example: `phase envelope tolerance=0.04`.

The coexistence sampler requires both limiting `P,Q=0` and `P,Q=1` evaluations to be valid. Near p_c it performs an additional adaptive pass in $\log_{10}((p_c - p)/p_c)$ before appending the exact critical constant. At the lower endpoint, a dimensionless pressure-offset bisection guards against minute inconsistencies between a backend's exact T_t constant and its first saturation evaluations. No fluid-specific cutoff is embedded in the package.

31.5 Shared styling, adaptive sampling, and stable names

The shared controls `labels`, `label placement`, family-specific isentrope/isochore label controls, autonode controls, `curve style`, `legend`, `forget plot`, per-curve overrides, `coord digits`, and `axis options` behave exactly as documented for the other projections. PT additionally accepts:

/luacoolprop/diagram/PT/initial intervals=⟨*positive integer*⟩ (no default)

Default: 18.

Sets the common initial logarithmic-pressure subdivision.

Example: initial intervals=24.

/luacoolprop/diagram/PT/max depth=⟨*nonnegative integer*⟩ (no default)

Default: 8.

Limits common adaptive recursion.

Example: max depth=9.

/luacoolprop/diagram/PT/tolerance=⟨*positive number*⟩ (no default)

Default: 0.10.

Sets the common PT midpoint tolerance.

Example: tolerance=0.05.

/luacoolprop/diagram/PT/log weight=⟨*nonnegative number*⟩ (no default)

Default: 30.

Weights logarithmic pressure in the midpoint metric.

Example: log weight=40.

/luacoolprop/diagram/PT/domain policy=⟨*ignore, warning, or error*⟩ (no default)

Default: ignore.

Controls diagnostics for invalid CoolProp samples without reconnecting separate valid domains.

Example: domain policy=warning.

/luacoolprop/diagram/PT/isenthalp initial intervals=⟨*positive integer*⟩ (no default)

Default: 18.

Overrides initial intervals for isenthalps.

Example: isenthalp initial intervals=24.

/luacoolprop/diagram/PT/isenthalp max depth=⟨*nonnegative integer*⟩ (no default)

Default: 8.

Overrides recursion depth for isenthalps.

Example: isenthalp max depth=9.

/luacoolprop/diagram/PT/isenthalp tolerance=⟨*positive number*⟩ (no default)

Default: 0.10.

Overrides the isenthalp midpoint tolerance.

Example: isenthalp tolerance=0.05.

Isentrope sampling can be overridden with isentrope initial intervals, isentrope max depth, and isentrope tolerance. The analogous isochore ... keys control isochores. Curve identifiers are stable and unit-aware: lcp-pt-phase-envelope, lcp-pt-s-..., lcp-pt-h-..., and lcp-pt-v-.... They work with label ... for, curve style for, and TikZ intersections.

31.6 PT process projection and exports

PT process options use `/luacoolprop/process/PT`. All shared process keys, strict quantity syntax, endpoint validation, styles, labels, markers, sampling controls, named paths, and SI exports from the PH process reference apply without reduction.

`/luacoolprop/process/PT/temperature axis unit`=⟨*kelvin or celsius*⟩ (no default)

Default: `global unit`.

Couples process X coordinates to the PT axis; Celsius subtracts exactly 273.15 from absolute temperature.

Example: `temperature axis unit=celsius`.

`/luacoolprop/process/PT/pressure axis unit`=⟨*pa, kpa, mpa, or bar*⟩ (no default)

Default: `global unit`.

Couples process Y coordinates to the PT pressure axis.

Example: `pressure axis unit=bar`.

`/luacoolprop/process/PT/temperature scale`=⟨*number*⟩ (no default)

Default: `global scale`.

Sets an advanced raw X-coordinate multiplier with zero offset.

Example: `temperature scale=1`.

Accepted alias: `t scale` forwards to `temperature scale`.

With `export coordinates=StateAB`, `\StateABFromX` is the scaled/offset temperature and `\StateABFromY` is scaled pressure. The property exports `FromTemperatureSI`, `FromPressureSI`, `FromEnthalpySI`, `FromEntropySI`, and `FromSpecificVolumeSI` remain SI values. Isobars and isotherms are exact horizontal and vertical segments. Isentropes, isenthalps, isochores, and quality processes are adaptively sampled in logarithmic pressure. A quality process is valid, but every one of its points lies on the single coexistence locus because PT cannot distinguish its quality value.

31.7 Plain TeX and ConTeXt

The PT implementation resides entirely in the generic layer.

Plain LuaTeX PT diagram

```
\input p-luacoolprop.tex
\pgfplotsset{compat=1.18}
\LCPPTDiagram[temperature axis unit=celsius,
  isentrope=true,entropy values={1.4,1.7,2.0}]
\bye
```

ConTeXt MkIV PT diagram

```
\usemodule[t][luacoolprop]
\pgfplotsset{compat=1.18}
\starttext
  \LCPPTDiagram[temperature axis unit=celsius,
    isenthalp=true,enthalpy values={200,300,400}]
\stoptext
```

Compile the first example with `luatex -shell-escape` and the second with `context -luatex -shell-escape`. ConTeXt's default LMTX engine does not provide the LuaTeX FFI module.

32 PGFPlots fluid style

`/pgfplots/luacoolprop fluid`=⟨*CoolProp identifier*⟩ (no default)

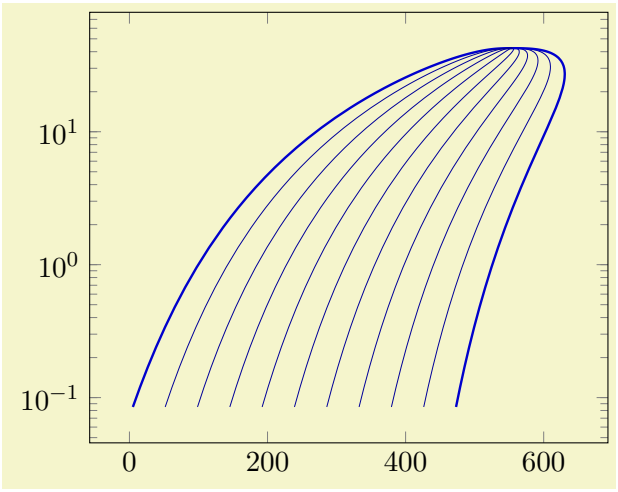
Default: `none`.

Sets the global fluid while processing an axis option list. This is useful when the axis, rather than a LuaCoolProp command, owns document configuration.

Example: `luacoolprop fluid=Propane`.

Accepted alias: `lcp fluid` forwards to `luacoolprop fluid`.

Fluid selected by the PGFPlots axis




```
\begin{tikzpicture}
\begin{axis}[luacoolprop fluid=Propane,ymode=log]
  \LCPAddPHQuality
\end{axis}
\end{tikzpicture}
```

33 Named states and reusable cycles

A named model separates the thermodynamic construction from the diagram used to show it. Define the physical states once, connect them with conserved-property processes, and assemble an ordered cycle. PH, PV, TS, HS and PT are views of the same SI states. These commands complement the explicit process commands; they do not change their input syntax or defaults.

Model scope The model fixes its pure fluid, library selection and enthalpy/entropy reference convention at declaration. Defaults are copied from the current global setup. Names and subsequent definitions follow TeX groups; the first nested mutation forks the model. Leaving that group restores the outer definitions. Native-library loading and reference locks still last for the compiler process.

Named process scope A definition refers to states in one model. Process names cannot be reused in that model. A supplied destination specification creates a new state only after complete validation; otherwise the destination must already exist. A failed definition changes neither states nor processes. Physical quantities use the state-input unit contract.

Model view scope Options affect only that rendering or export call. The model's physical fluid, reference, endpoints and conserved quantity cannot be overridden. Coordinate scales and units come from the current `\LCPSetup`; this never changes stored SI properties. Ordinary PGFPlots axis limits, transformations and axis labels remain the user's responsibility.

`\LCPDeclareModel[`⟨*options*⟩`]{`⟨*model*⟩`}`

Declare a model using the `/luacoolprop/model` family. Model, state, process and cycle names start with an ASCII letter and may then contain ASCII letters, digits, underscores, colons or hyphens. Names are case sensitive. State, process and cycle names have separate namespaces within a model. Declaring the same model twice in the same TeX group is an error; an inner group may shadow an outer model with a new declaration.

`\LCPDefineState{`⟨*model*⟩`}{`⟨*state*⟩`}{`⟨*properties*⟩`}`

Resolve a state from the existing numeric property-pair syntax, such as `T=0C,q=1`. Redundant properties are checked for consistency. Store the completed pressure, temperature, enthalpy, entropy, specific volume and density in SI; store quality only where it is defined. Duplicate state names are errors.

`\LCPDefineProcess{`⟨*model*⟩`}{`⟨*process*⟩`}[`⟨*options*⟩`]`

Use `/luacoolprop/model process`. Type, source and destination are required. The conserved quantity is taken from the source state. When `to state` is supplied, the destination must be new; its missing conserved quantity is filled before the state is resolved. Otherwise both endpoint names must exist. Conservation and consistency are checked before committing the definition. This command does not assume a compressor efficiency or assign heat and work.

`\LCPDefineCycle{`⟨*model*⟩`}{`⟨*cycle*⟩`}{`⟨*process list*⟩`}`

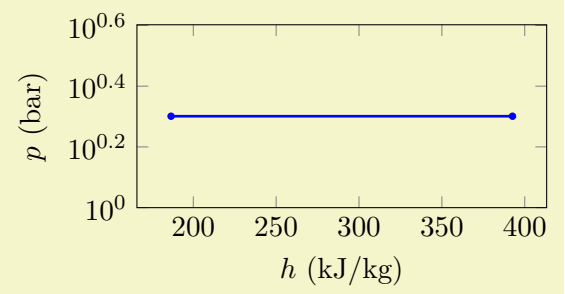
The comma-separated list contains at least two previously defined processes, in traversal order. Every destination name must equal the next source name, including the last-to-first connection. Nearness on a graph does not close a cycle. Duplicate cycle names and open or disconnected sequences are errors.

`\LCPAddCycle{`⟨*diagram type*⟩`}{`⟨*model*⟩`}{`⟨*cycle*⟩`}[`⟨*options*⟩`]`

Inside an existing PGFPlots axis, draw the ordered process paths using `/luacoolprop/model view`. Supported views are PH, PV, TS, HS and PT. Sampling is adapted to each projection while endpoint SI values remain identical. A PT view cannot distinguish two-phase states with different qualities at the same saturation pressure: coincident endpoints in that view are physically normal.

`\LCPAddModelProcess{`⟨*diagram type*⟩`}{`⟨*model*⟩`}{`⟨*process*⟩`}[`⟨*options*⟩`]`

Draw only one named process. Use several calls to give different processes individual colours or TikZ styles. This uses the same view keys as the full cycle.



```
\LCPDeclareModel[fluid=R134a]{boiler}
\LCPDefineState{boiler}{liquid}{p=2bar,q=0}
\LCPDefineProcess{boiler}{boil}[
  type=isobar,from=liquid,to=vapour,to state={q=1}]
\begin{tikzpicture}
\begin{axis}[width=7cm,height=4cm,ymode=log,
  xlabel={$h$ (kJ/kg)},ylabel={$p$ (bar)},ymin=1,ymax=4]
  \LCPAddModelProcess{PH}{boiler}{boil}[color=blue,mark endpoints=true]
\end{axis}
\end{tikzpicture}
```

\LCPStateValue{⟨*model*⟩}{⟨*state*⟩}{⟨*property*⟩}

Expand to a round-trip numerical value in SI: `p` in Pa, `T` in K, `h` in J/kg, `s` in J/(kg K), `v` in cubic metres per kilogram, `rho` in kilograms per cubic metre, or dimensionless `q`. Use inside an `\edef` to retain a value in a TeX macro, then apply ordinary PGF mathematics or `\num` for display. Requesting single-phase quality is an error, not a zero or a silently extrapolated value. Unknown names/properties fail.

\LCPStateCoordinate{⟨*diagram type*⟩}{⟨*model*⟩}{⟨*state*⟩}{⟨*x or y*⟩}

Expand to a plotted coordinate under the current global unit/scale settings. This only transforms the stored state; it does not perform another property flash. Use these values with ordinary `axis cs` nodes and coordinates. They do not include PGFPlots' subsequent logarithmic or canvas transformation.

\LCPLogState{⟨*model*⟩}{⟨*state*⟩}

Write all SI properties to the log with fluid, reference and explicit `origin=model` provenance. Undefined quality is written as `nan`. This is an equation-of-state result, not a graphical reading.

\LCPExportStates{⟨*model*⟩}{⟨*filename*⟩}

Export a whitespace-separated state table in definition order, with a JSON comment manifest identifying origin, units, fluid, reference and native/runtime provenance. Columns are `state p_Pa T_K h_Jkg s_JkgK v_m3kg rho_kgm3 Q`. Undefined quality is `nan`. No new property evaluation is needed. Parent directories are created; writing uses an atomic replacement.

\LCPExportCycleTable{⟨*diagram type*⟩}{⟨*model*⟩}{⟨*cycle*⟩}[⟨*options*⟩]{⟨*filename*⟩}

Export the same sampled cycle paths used by the renderer. View keys and coordinate scales match `\LCPAddCycle`. The ordinary table manifest additionally includes the named construction, its SI states and process connectivity. Break rows separate processes and disconnected domains. Export does not typeset labels or emit nodes.

33.1 Model definition and view keys

/luacoolprop/model/fluid=⟨*pure CoolProp fluid*⟩ (no default)

Default: `current global fluid`.

Fixes the fluid for this construction. Mixtures and unsupported backends are rejected by the existing pure-fluid contract.

Example: `fluid=R134a`.

/luacoolprop/model/library=⟨*shared-library path*⟩ (no default)

Default: `current global library selection`.

Selects the external library before the first native load. It cannot replace an already loaded library.

Example: `library=/opt/coolprop/libCoolProp.dylib`.

`/luacoolprop/model/reference state`=⟨*DEF, IIR, ASHRAE or NBP*⟩ (no default)

Default: `current global reference convention`.

Selects and locks the enthalpy/entropy reference before any state is defined. A nested TeX group cannot undo that native lock.

Example: `reference state=DEF`.

`/luacoolprop/model process/type`=⟨*isobar, isotherm, isentropic, isenthalpic, quality or isochore*⟩ (no default)

Default: `required`.

The source state supplies the conserved property. Existing process-type aliases are accepted, including isoquality for quality. No default process is guessed.

Example: `type=isentropic`.

`/luacoolprop/model process/from`=⟨*state name*⟩ (no default)

Default: `required`.

Names a state already defined in the model. This endpoint remains immutable.

Example: `from=A`.

`/luacoolprop/model process/to`=⟨*state name*⟩ (no default)

Default: `required`.

Names an existing destination, or a new destination when to state is supplied.

Example: `to=B`.

`/luacoolprop/model process/to state`=⟨*property specification*⟩ (no default)

Default: `empty`.

Creates the destination from these properties and the conserved source property. Redundant inconsistent values fail before any new name is published.

Example: `to state={p=10bar}`.

`/luacoolprop/model view/color`=⟨*TikZ colour*⟩ (no default)

Default: `black`.

Applies to all paths drawn by this call. Draw individual model processes for different colours.

Example: `color=blue`.

`/luacoolprop/model view/style`=⟨*TikZ plot options*⟩ (no default)

Default: `line width=1pt`.

Use ordinary line, arrow and dash options. The style affects rendering rather than stored state values.

Example: `style={thick,->}`.

`/luacoolprop/model view/mark endpoints`=⟨*boolean*⟩ (no default)

Default: `false`.

Adds markers at the completed endpoints. Normal boolean aliases yes/no, on/off and 1/0 are accepted.

Example: `mark endpoints=true`.

/luacoolprop/model view/coord digits=⟨*integer from 1 to 17*⟩ (no default)

Default: `6`.

Precision of emitted plot coordinates; stored SI states and explicit tables keep round-trip precision.

Example: `coord digits=8`.

/luacoolprop/model view/cache=⟨*off, read-write, refresh or read-only*⟩ (no default)

Default: `current global cache policy`.

Controls sampled process-path caching. Named endpoint values participate in the cache identity. State definitions themselves are resolved on each compiler run.

Example: `cache=read-write`.

/luacoolprop/model view/cache directory=⟨*directory path*⟩ (no default)

Default: `current global cache directory`.

Selects the numerical process cache, distinct from explicit state/cycle table exports and the independent autonode placement cache.

Example: `cache directory=build/model-cache`.

/luacoolprop/model view/tolerance=⟨*positive number*⟩ (no default)

Default: `diagram-specific process tolerance`.

Controls adaptive process-path sampling in the selected coordinate scales. It never changes the stored endpoint states; omitted tolerance uses the existing sampler default.

Example: `tolerance=0.15`.

/luacoolprop/model view/max depth=⟨*nonnegative integer*⟩ (no default)

Default: `7`.

Maximum adaptive refinement depth, with the same meaning as for explicit processes.

Example: `max depth=6`.

/luacoolprop/model view/initial intervals=⟨*positive integer*⟩ (no default)

Default: `12`.

Initial process-sampling intervals. Exact straight segments in a projection do not need adaptive refinement.

Example: `initial intervals=16`.

/luacoolprop/model view/marker style=⟨*TikZ plot options*⟩ (no default)

Default: `only marks,mark=*,mark size=1.2pt`.

Styles endpoint marks when enabled. The path style remains independent.

Example: `marker style={only marks,mark=square*,mark size=2pt}`.

/luacoolprop/model view/domain policy=⟨*ignore, warn or error*⟩ (no default)

Default: `ignore`.

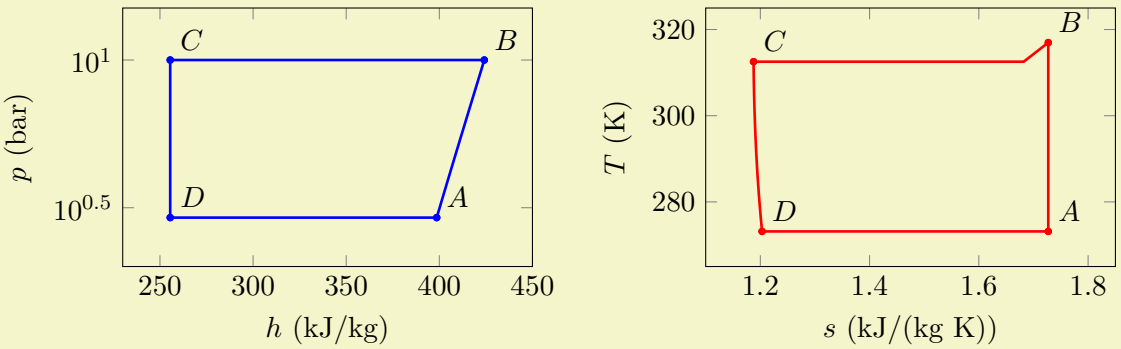
Keeps disconnected sampled regions separate. Select warn to report omitted intervals, or error to reject a process crossing an invalid sampled domain.

Example: `domain policy=warn`.

33.2 One refrigerator, two views

Consider a pure-fluid R134a refrigerator. State A is saturated vapour at 0 °C. An ideal reversible adiabatic compressor takes it to 10 bar; the condenser ends at saturated liquid at that pressure. An isenthalpic valve returns the fluid to the evaporator pressure, then an isobaric evaporation closes the cycle. These assumptions specify four states without choosing a diagram. The same construction is drawn below in PH and TS.

Define the states and processes outside the axes so both views can see them. The conserved entropy determines B; pressure determines the isobar through B and C; enthalpy determines the valve outlet D. The final process refers back to A, so its definition checks the evaporation pressure rather than creating a second, approximately equal endpoint. The source of each numerical value is explicit: it is a CoolProp state, not an intersection read from a printed curve.



The valve outlet has quality 0.279.

```
\LCPDeclareModel[fluid=R134a,reference state=DEF]{fridge}
\LCPDefineState{fridge}{A}{T=0C,q=1}
\LCPDefineProcess{fridge}{compress}[
  type=isentropic,from=A,to=B,to state={p=10bar}]
\LCPDefineProcess{fridge}{condense}[
  type=isobar,from=B,to=C,to state={q=0}]
\edef\LowPressure{\LCPStateValue{fridge}{A}{p}}
\LCPDefineProcess{fridge}{valve}[type=isenthalpic,
  from=C,to=D,to state={p=\LowPressure}]
\LCPDefineProcess{fridge}{evaporate}[type=isobar,from=D,to=A]
\LCPDefineCycle{fridge}{loop}{compress,condense,valve,evaporate}
\begin{tikzpicture}
\begin{axis}[width=7cm,height=5cm,ymode=log,
  xlabel={$h$ (kJ/kg)},ylabel={$p$ (bar)},
  xmin=230,xmax=450,ymin=2,ymax=15]
  \LCPAddCycle{PH}{fridge}{loop}[color=blue,mark endpoints=true]
  \pgfplotsinvokeforeach{A,B,C,D} {
    \node[above right] at (axis cs:
      \LCPStateCoordinate{PH}{fridge}{#1}{x},
      \LCPStateCoordinate{PH}{fridge}{#1}{y}) {$#1$};
  }
\end{axis}
\end{tikzpicture}
\quad
\begin{tikzpicture}
\begin{axis}[width=7cm,height=5cm,
  xlabel={$s$ (kJ/(kg K))},ylabel={$T$ (K)},
  xmin=1.1,xmax=1.85,ymin=265,ymax=325]
  \LCPAddCycle{TS}{fridge}{loop}[color=red,mark endpoints=true]
  \pgfplotsinvokeforeach{A,B,C,D} {
    \node[above right] at (axis cs:
      \LCPStateCoordinate{TS}{fridge}{#1}{x},
      \LCPStateCoordinate{TS}{fridge}{#1}{y}) {$#1$};
  }
\end{axis}
\end{tikzpicture}
\edef\ValveQuality{\LCPStateValue{fridge}{D}{q}}
\par The valve outlet has quality \num[round-mode=places,round-precision=3]{\ValveQuality}.
\LCPLogState{fridge}{D}
\LCPExportStates{fridge}{build/fridge-states.dat}
\LCPExportCycleTable{TS}{fridge}{loop}{build/fridge-ts.dat}
```

The numerical quality printed above is dimensionless. For rounded presentation, use the document's normal number-formatting settings; the stored value should retain its precision for subsequent calculations. A state table can likewise use `\LCPStateValue` in each cell. The explicit exports separate reusable numerical data from the disposable process cache.

For steady flow with negligible changes in kinetic and gravitational potential energy, the ideal compressor work received by the fluid per unit mass is $h_B - h_A$. The condenser heat received is $h_C - h_B < 0$, and the evaporator heat received is $h_A - h_D > 0$. The valve constraint gives $h_D = h_C$, so these terms close the cycle energy balance. The model stores state differences; it does not silently assign heat or work to an arbitrary process. Apply those physical assumptions explicitly when computing a power or coefficient of performance, and specify mass flow separately. Likewise, a zero sum of state entropy differences around a closed cycle is not zero entropy production: the idealised isenthalpic valve is still irreversible.

33.3 Scopes, units and deliberate alternatives

Within a model, state and process definitions are immutable. Reusing an existing state as a destination checks that its properties satisfy the process. A failed state, process or cycle definition leaves the preceding construction intact. To compare a different cycle, declare another model or fork the construction in Lua. Inside a TeX group, adding definitions to an outer model creates

a local copy: the outer construction is restored when the group ends. A new declaration with the same model name in an inner group creates an independent local model. Re-declaring that name in the same group is an error. Define reusable states before entering a picture or axis; those environments also introduce groups.

At the critical point, liquid and vapour are indistinguishable and quality is undefined. A quality-based definition at or above the critical temperature or pressure is rejected, even if the native library echoes an input quality. Define the critical state from its critical temperature and density instead; requesting its quality still raises an error. Use the quality-curve family to show a saturation branch ending at its common critical limit.

Input uses SI by default and accepts the existing unit suffixes and dot-decimal scientific notation. Output from `\LCPStateValue` is always SI. Coordinate output and paths follow the current global unit settings, such as `\LCPSetup{temperature axis unit=celsius}`. Changing these settings changes a view, not the stored state or its thermodynamic reference. Keep ordinary axis labels consistent with the selected coordinate units. Named-model rendering does not infer scales from an arbitrary surrounding PGFPlots axis.

Generated TikZ path names are distinct for each model instance and projection; processes with the same identifier in different models do not overwrite each other's paths. Treat those generated names as opaque. For a deliberate graphical intersection, give an individual `\LCPAddModelProcess` call an explicit `style={name path global=my-compressor}` and keep that user-chosen selector unique, as for any ordinary TikZ path.

Reference conventions follow the same process-lifetime lock as ordinary diagrams. For example, a model using DEF and another using IIR for the same already locked fluid cannot coexist in one compiler run. Use separate compilations for that comparison. TeX grouping and model shadowing restore names, not the external library's global reference configuration. Duplicate names, unavailable states, open cycles, inconsistent constraints and invalid numeric input fail explicitly.

33.4 Model values versus graphical readings

TikZ intersections remain useful for teaching how to read a diagram. Their accuracy depends on sampled paths, clipping and printed scales. Named states instead come from a validated thermodynamic property pair. They retain `origin=model` in logs and exported state/construction metadata. Do not present these values as measured intersections. Conversely, the package never promotes a named TikZ coordinate into a thermodynamic state implicitly. To model a graphical reading, explicitly supply its estimated physical properties to a new state and retain the reading uncertainty in the surrounding discussion.

Every projection uses the same endpoint snapshots. Process interiors still use the existing projection-specific adaptive samplers; two views can therefore have different numbers of samples. Disconnected domains keep their breaks. A stored cycle guarantees named connectivity and checked endpoint constraints, not global validity of an unsupported equation-of-state region or physical correctness of unstated machine assumptions. The bounded numerical cache stores process samples with endpoint values, coordinate scales, fluid/reference and runtime/native identity. Changing a construction cannot reuse samples for different endpoints. A hit does not remove the state-definition flashes at the start of a new run.

33.5 The same construction from Lua

The object returned by `lcp.models.new` owns the definitions and exposes `define_state`, `define_process`, `define_cycle`, `state`, `process`, `cycle`, `value` and `fork`. Definition and getter results are detached snapshots. Projection methods `project_state`, `project_process` and `project_cycle` return coordinates with unchanged SI endpoints; `plot_process` and `plot_cycle` serialize PGFPlots paths. `export_states` and `export_cycle` publish explicit tables. The field-

by-field contracts are in `doc/luatex/luacoolprop/luacoolprop-lua-api.md`. Only the five built-in projections have a named-state coordinate contract.

This independent program resolves an ideal compressor and checks that its PH and TS views retain the same inlet enthalpy. It executes with texlua as part of the documentation build; its output is numerical, not a diagram.

```
local lcp = require("luacoolprop")
local model = lcp.models.new{fluid="R134a"}
model:define_state("A", "T=0C,q=1")
model:define_process("compress", {
  type="isentropic", from="A", to="B", to_state="p=10bar"})
local ph, ph_meta = model:project_process("PH", "compress")
local ts, ts_meta = model:project_process("TS", "compress")
assert(ph_meta.from.h == ts_meta.from.h)
local specific_work = model:value("B", "h") - model:value("A", "h")
io.write(string.format("Ideal compressor work: %.6g J/kg\n", specific_work))
```

34 Steady-flow components and cycle balances

Component definition scope Components belong to a named model and share its process namespace and TeX lifetime. Type and endpoint names are required. Definitions are atomic. Physical options cannot be changed by view settings. Heat/work use the steady-flow single-stream convention below.

Balance scope and units Settings obey TeX groups and apply to subsequent balance queries, logs and exports. Mass flow is a positive finite number in kg/s, with dot-decimal notation and optional exponent; no suffix is accepted. It has no default. Specific energies are J/kg, specific entropy is J/(kg K), and powers are W. Both heat and shaft work received by the fluid are positive. There is one common flow through a closed cycle.

`\LCPDefineComponent{`⟨*model*⟩`}{`⟨*component*⟩`}[`⟨*options*⟩`]`

Use `/luacoolprop/component` to define a compressor, pump, turbine, valve, heat exchanger, pressure loss or custom component. The source must exist. A nonempty `to state` creates a new destination. A passive closure may also create a new destination from its pressure and heat options alone. Otherwise the destination must exist and is checked rather than overwritten. Components are drawn and assembled with the ordinary named-process/cycle commands. The worked examples below exercise every component type and both outlet modes.

`\LCPBalanceSetup{`⟨*options*⟩`}`

Set `/luacoolprop/balance` options in the current TeX group. These settings never change stored thermodynamic states. Use `mass flow=` to clear an inherited flow. Numeric validation occurs when a balance is requested.

`\LCPComponentValue{`⟨*model*⟩`}{`⟨*component*⟩`}{`⟨*field*⟩`}`

Expand to a round-trip numeric SI value, suitable for `\edef` and `\num`. Fields are `heat`, `work`, `delta_h`, `delta_s`, `energy_residual`, `entropy_generation`, `efficiency`, `mass_flow`, `heat_rate`, and `work_rate`. Efficiency exists for compressors, pumps and turbines; entropy generation only for the explicitly adiabatic models; flow and rates require an explicit mass flow. Unknown or unavailable fields raise an error rather than printing zero.

`\LCPCycleValue{`⟨*model*⟩`}{`⟨*cycle*⟩`}{`⟨*field*⟩`}`

Return a numeric closed-cycle balance field. Every member must be an explicit component. Use the component fields above except `efficiency`, plus `heat_input`, `heat_rejected`,

`work_input`, `work_output`, which are positive gross specific-energy magnitudes. Entropy generation is available only when known for every component. No coefficient of performance or efficiency of an arbitrary cycle is inferred.

`\LCPLogBalance{`⟨*model*⟩`}{`⟨*cycle*⟩`}`

Write a JSON cycle balance to the terminal/log, marked `origin=model`, including each component's assumptions, signed specific energies and available rates. The numerical fields follow the SI contract above.

`\LCPExportBalance{`⟨*model*⟩`}{`⟨*cycle*⟩`}{`⟨*filename*⟩`}`

Atomically write an explicit table of component balances in cycle order. The JSON manifest includes construction states, assumptions, reference convention, units, sign convention and runtime/native provenance. Undefined entropy production or powers are `nan` in table columns; corresponding JSON fields are omitted. This command does not change the current cache policy or draw anything.

34.1 Component and balance keys

`/luacoolprop/component/type=`⟨*compressor, pump, turbine, valve, heat exchanger, pressure loss or custom*⟩ (no default)

Default: `required`.

Selects the physical model. Compressor, pump, turbine and valve are adiabatic. Pumps require liquid states; turbines require a gas-like inlet and allow equilibrium wet outlets. Heat exchangers have zero shaft work and are isobaric unless a pressure loss is specified. A pressure-loss element requires an explicit thermal model and pressure specification. Custom uses existing endpoints and explicit heat. No model is inferred from the plotted shape.

Example: `type=compressor`.

`/luacoolprop/component/from=`⟨*state name*⟩ (no default)

Default: `required`.

Names the existing inlet in this model. Its SI properties and reference are retained unchanged.

Example: `from=A`.

`/luacoolprop/component/to=`⟨*state name*⟩ (no default)

Default: `required`.

Names an existing checked outlet, or a new state to create when to state is supplied. Passive closures can also create a new outlet from pressure and heat options alone. Names follow the named-state contract.

Example: `to=B`.

`/luacoolprop/component/to state=`⟨*property specification*⟩ (no default)

Default: `empty`.

Compressor, pump, turbine and valve accept only an outlet pressure, such as p=10bar. Heat exchangers accept one independent thermal property, such as T=30C or q=0, with optional outlet pressure; absent pressure options preserve the inlet pressure. Pressure-loss elements accept only outlet pressure here. Do not combine a supplied pressure with pressure drop or pressure ratio, or a thermal property with heat. Custom rejects this option.

Example: `to state={p=10bar}`.

/luacoolprop/component/efficiency=⟨*number in the interval (0,1]*⟩ (no default)

Default: `required for compressor, pump and turbine; otherwise absent`.

Isentropic efficiency determines the actual outlet enthalpy. Compression divides the ideal enthalpy rise by efficiency; expansion multiplies the ideal enthalpy drop by efficiency. Unit efficiency reuses the ideal state and isentropic path. Other component types reject a nonempty efficiency. No default is guessed.

Example: `efficiency=0.75`.

/luacoolprop/component/heat=⟨*specific energy*⟩ (no default)

Default: `required for custom and specified heat; otherwise absent`.

Signed heat received, in J/kg by default, with the same energy-unit suffixes as state enthalpy. For custom, shaft work is the enthalpy rise minus this heat. For heat exchangers or pressure loss with thermal model=specified heat, shaft work is zero and this heat fixes the enthalpy rise. Rejects redundant outlet thermal data and nonthermal component types. Zero is an explicit net heat, not an adiabatic assertion. Heat rates in W are not construction inputs.

Example: `heat=-2kJ/kg`.

/luacoolprop/component/pressure drop=⟨*nonnegative pressure*⟩ (no default)

Default: `absent`.

For heat exchangers and pressure-loss elements only. Sets outlet pressure to inlet pressure minus this value, in Pa by default; existing state pressure suffixes and scientific notation are accepted. The resulting pressure must be positive. Exclusive with pressure ratio and pressure in a new to state. An existing outlet is checked, never overwritten. Zero is allowed. Definition-local, with normal model grouping.

Example: `pressure drop=1.2E5Pa`.

/luacoolprop/component/pressure ratio=⟨*dimensionless number in (0,1]*⟩ (no default)

Default: `absent`.

For heat exchangers and pressure-loss elements only. Always outlet pressure divided by inlet pressure. Exclusive with pressure drop and pressure in a new to state; existing outlet pressure must agree. One means no loss. Without any pressure route, an exchanger remains isobaric; a pressure-loss element fails. Definition-local; dot decimals only.

Example: `pressure ratio=0.95`.

/luacoolprop/component/thermal model=⟨*adiabatic or specified heat*⟩ (no default)

Default: `required for pressure loss; otherwise absent`.

Only pressure-loss elements accept this key. Adiabatic rejects heat and conserves enthalpy with zero shaft work; entropy generation is the entropy increase. Specified heat requires heat, including explicit zero, conserves zero shaft work and leaves entropy generation unavailable. Its path is schematic. No boundary temperature, geometry or flow correlation is inferred. Definition-local; normal model grouping applies.

Example: `thermal model=adiabatic`.

/luacoolprop/balance/mass flow=⟨*positive number in kg/s, or empty*⟩ (no default)

Default: `empty`.

Enables powers for subsequent queries, logs and exports. The same flow traverses every component. Empty clears it; specific energies remain available, but power queries fail. This setting respects TeX grouping.

Example: `mass flow=0.1`.

34.2 A refrigerator with an explicit compressor efficiency

A state difference becomes a heat or work calculation only after a physical model has been specified. Consider the same pure-fluid R134a refrigerator as before, now with an adiabatic compressor of isentropic efficiency $\eta_s = 0.75$. Saturated vapour enters at 0 °C, the condenser operates at 10 bar, and the mass flow is 0.1 kg/s. All four devices are steady-flow, single-inlet, single-outlet components; changes in kinetic and gravitational potential energy are negligible. The condenser and evaporator have no shaft work or pressure loss. The valve has neither heat transfer nor shaft work.

For every component, the convention is

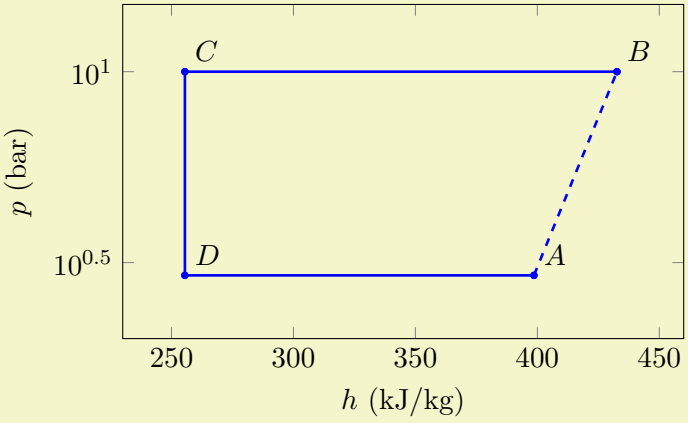
$$h_{\text{out}} - h_{\text{in}} = q + w, \qquad \dot{Q} = \dot{m}q, \qquad \dot{W} = \dot{m}w,$$

where heat q and shaft work w are positive when received by the fluid. These symbols denote specific energies; the state property `q` in `\LCPStateValue` still denotes vapour quality. The balance fields use the unambiguous names `heat` and `work`. Flow work is already included in enthalpy and must not be added again.

The compressor first resolves the hypothetical isentropic outlet at the required pressure. Its actual outlet follows from

$$s_{2s} = s_1, \qquad \eta_s = \frac{h_{2s} - h_1}{h_2 - h_1}, \qquad h_2 = h_1 + \frac{h_{2s} - h_1}{\eta_s}.$$

This specifies two endpoint states, not an interior trajectory. Accordingly the nonideal compressor is drawn as a dashed schematic connector. Its shape in PH, PV, TS, HS or PT is not an equation of state for the compression path. At unit efficiency the resolved isentropic outlet is reused exactly, and the ordinary isentropic sampler supplies a thermodynamic path. The default dash can be overridden with the usual model-view `style`, but that does not change its physical interpretation. A custom component also has only a schematic endpoint connector. Neither connector creates intermediate thermodynamic states.



Compressor power received: 3404.8 W.
Condenser heat rate received: −17 715.5 W.
Cycle energy residual: 0.000 000 J/kg.

```
\LCPDeclareModel[fluid=R134a,reference state=DEF]{plant}
\LCPDefineState{plant}{A}{T=0C,q=1}
\LCPDefineComponent{plant}{compress}[
  type=compressor,from=A,to=B,to state={p=10bar},efficiency=0.75]
\LCPDefineComponent{plant}{condense}[
  type=heat exchanger,from=B,to=C,to state={q=0}]
\edef\LowPressure{\LCPStateValue{plant}{A}{p}}
\LCPDefineComponent{plant}{expand}[
  type=valve,from=C,to=D,to state={p=\LowPressure}]
\LCPDefineComponent{plant}{evaporate}[
  type=heat exchanger,from=D,to=A]
\LCPDefineCycle{plant}{loop}{compress,condense,expand,evaporate}
\LCPBalanceSetup{mass flow=0.1}
\begin{tikzpicture}
\begin{axis}[width=9cm,height=6cm,ymode=log,
  xlabel={$h$ (kJ/kg)},ylabel={$p$ (bar)},
  xmin=230,xmax=460,ymin=2,ymax=15]
  \LCPAddCycle{PH}{plant}{loop}[color=blue,mark endpoints=true]
  \pgfplotsinvokeforeach{A,B,C,D} {
    \node[above right] at (axis cs:
      \LCPStateCoordinate{PH}{plant}{#1}{x},
      \LCPStateCoordinate{PH}{plant}{#1}{y}) {$#1$};
  }
\end{axis}
\end{tikzpicture}
\edef\InputPower{\LCPComponentValue{plant}{compress}{work_rate}}
\edef\CondenserHeat{\LCPComponentValue{plant}{condense}{heat_rate}}
\edef\EnergyResidual{\LCPCycleValue{plant}{loop}{energy_residual}}
\par Compressor power received:
  \num[round-mode=places,round-precision=1]{\InputPower}\ W.
\par Condenser heat rate received:
  \num[round-mode=places,round-precision=1]{\CondenserHeat}\ W.
\par Cycle energy residual:
  \num[round-mode=places,round-precision=6]{\EnergyResidual}\ J/kg.
\LCPLogBalance{plant}{loop}
\LCPExportBalance{plant}{loop}{build/component-balances.dat}
\LCPExportCycleTable{TS}{plant}{loop}{build/component-ts.dat}
```

The negative condenser heat rate means that heat leaves the refrigerant. `heat_rejected` is its positive magnitude in J/kg, summed over devices that reject heat. Likewise `heat_input`, `work_input` and `work_output` are positive gross magnitudes; `heat` and `work` are signed net sums. For this refrigerator, the cooling coefficient of performance is `heat_input/work_input`, while the heating coefficient is `heat_rejected/work_input`. These ratios are appropriate for the stated machine; they are not automatically assigned to every closed cycle. Supply a positive nonzero denominator and identify the intended useful effect explicitly.

Changing `mass flow` changes powers, not states or specific energies. The same flow passes through every component: these balances do not describe branches, mixers, separators or multiple streams. Omitting the flow still allows specific-energy queries, but requesting `heat_rate` or `work_rate` then raises an error. TeX groups restore the previous balance settings.

34.3 Limits, consistency and the entropy balance

A compressor requires a strict pressure rise and $0 < \eta_s \leq 1$, with no assumed efficiency. Its inlet, ideal outlet and actual outlet must be dry vapour or supported gas-like single-phase states; wet compression and liquid pumping are outside this model. A valve requires a strict pressure drop. Saturated or single-phase destinations are resolved through the same pure-fluid/reference contract as named states. By default a heat exchanger follows an isobar and may cross the saturation dome. An imposed pressure loss instead defines endpoints, as explained below. It models the refrigerant stream only: no second stream, overall heat-transfer coefficient, pinch condition or reservoir temperature is inferred.

For the adiabatic compressor and valve, the specific entropy generation is $s_{\text{out}} - s_{\text{in}}$. An appreciable negative value is rejected. Native solver roundoff is retained: differences within 10^{-7} times the larger of 1 and the endpoint entropy magnitudes are tolerated, not clamped to zero. For heat exchangers and custom components, entropy transfer is unspecified, so `entropy_generation` is unavailable. In particular, zero *net* heat for a custom component does not assert that every boundary is adiabatic. The closed-cycle sum `delta_s` vanishes because entropy is a state function; it does not establish reversibility or zero entropy production.

Existing compressor outlets are checked against the calculated enthalpy with relative tolerance 10^{-7}, using a minimum absolute scale of 1 J/kg. Conserved-property components reuse the ordinary process consistency checks. Every balance reports `energy_residual=heat+work -delta_h`, retaining floating-point residuals rather than forcing them to zero. Such numerical checks establish consistency with the selected model, not the accuracy of unstated assumptions or the native equation of state.

Components share the process namespace. Draw them with `\LCPAddModelProcess`; assemble them with `\LCPDefineCycle`. Definitions are atomic and obey the same group/fork rules as named processes. Ordinary conserved-property processes remain supported, but a cycle containing one cannot provide a heat/work balance until an explicit component construction is used. Equal-looking paths are not sufficient to determine their energy exchanges.

The model's physical endpoints and assumptions enter exported construction metadata. Existing adaptive samplers and numerical caches are reused for conserved-property paths. Schematic connectors merely project two stored states; they require no numerical sample cache and can operate under a read-only policy without a cache entry. Cache policies still validate normally. Exported balance tables are explicit data products, never disposable cache files. Their JSON manifest records SI units, sign conventions, endpoints, assumptions and provenance; undefined entropy generation and omitted powers are written as `nan`.

34.4 Supplying a custom relation

For advanced use, determine both states from measurements or your own equations, then specify the heat received. The package calculates the shaft work needed to satisfy the steady-flow energy balance. It does not verify your heat-transfer law or the second law without the missing thermal boundary information. For example, the following imposed compressor endpoints and cooling load require positive shaft work; the displayed value includes the specified heat loss.

Required shaft work:42 795.1 J/kg.

```
\LCPDeclareModel[fluid=R134a]{measured}
\LCPDefineState{measured}{A}{p=3bar,T=10C}
\LCPDefineState{measured}{B}{p=6bar,T=60C}
\LCPDefineComponent{measured}{device}[
  type=custom,from=A,to=B,heat=-2kJ/kg]
\edef\SpecificWork{\LCPComponentValue{measured}{device}{work}}
Required shaft work:
\num[round-mode=places,round-precision=1]{\SpecificWork}\ J/kg.
```

A compressor, pump, turbine, valve or heat exchanger can also use an already defined outlet: leave `to state` empty and name that outlet with `to`. Its properties are checked and never overwritten. For a compressor, pump or turbine, `efficiency` remains required. A custom component always requires existing states and explicit `heat`; it does not infer an outlet from an underdetermined balance.

34.5 A steam power plant: liquid pump and turbine

A Rankine cycle turns heat into net shaft work. Water leaves the condenser at 1 bar and 80 °C, enters a pump with isentropic efficiency 0.80, and reaches 50 bar. The boiler heats it at that

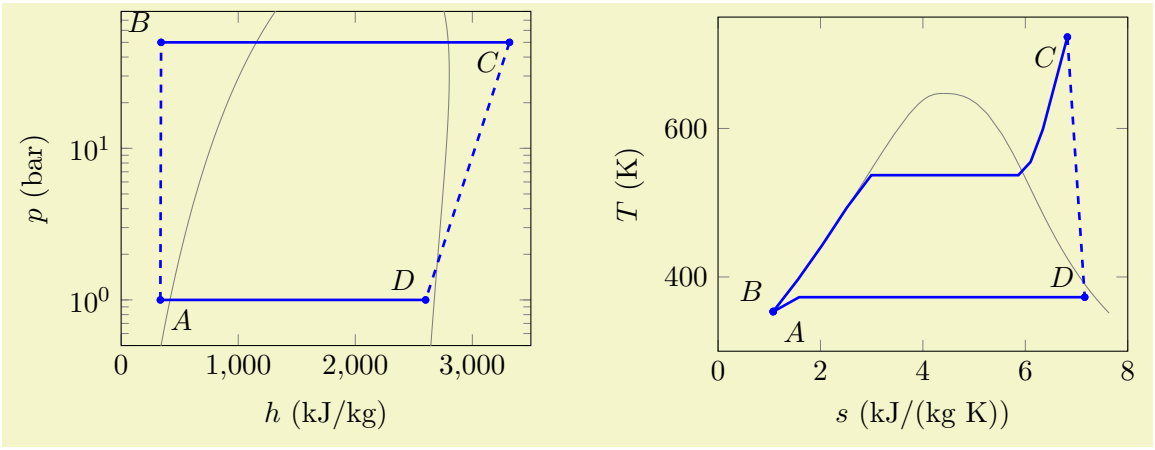
pressure to 450 °C. A turbine of isentropic efficiency 0.85 expands the steam to 1 bar, and the condenser returns it to the starting state. The common mass flow is 0.2 kg/s. Pump and turbine are adiabatic; boiler and condenser have zero shaft work and pressure loss. Kinetic and potential changes are neglected throughout. These are equilibrium component models, not equipment sizing or cavitation calculations.

Both machines resolve the hypothetical state $2s$ at the outlet pressure and inlet entropy. Their efficiencies have different definitions:

$$\eta_{\text{pump}} = \frac{h_{2s} - h_1}{h_2 - h_1}, \qquad \eta_{\text{turbine}} = \frac{h_1 - h_2}{h_1 - h_{2s}}.$$

Consequently the pump receives positive work and the turbine receives negative work. The pump uses the real-fluid enthalpy change; $v\,\Delta p$ is only an incompressible approximation, not its implementation. No efficiency is assumed when the key is omitted.

The construction below is defined once and projected into PH and TS. Solid grey saturation boundaries provide context. Dashed machine connectors join calculated endpoints; they do not assert a continuous irreversible path.



```
\begingroup
\LCPDeclareModel[fluid=Water,reference state=DEF]{rankine}
\LCPSetup{temperature axis unit=kelvin}
\LCPDefineState{rankine}{A}{p=1bar,T=80C}
\LCPDefineComponent{rankine}{pump}[
  type=pump,from=A,to=B,to state={p=50bar},efficiency=0.80]
\LCPDefineComponent{rankine}{boiler}[
  type=heat exchanger,from=B,to=C,to state={T=450C}]
\LCPDefineComponent{rankine}{turbine}[
  type=turbine,from=C,to=D,to state={p=1bar},efficiency=0.85]
\LCPDefineComponent{rankine}{condenser}[
  type=heat exchanger,from=D,to=A]
\LCPDefineCycle{rankine}{loop}{pump,boiler,turbine,condenser}
\LCPBalanceSetup{mass flow=0.2}
\tikzset{
  rankine A/.style={below right}, rankine B/.style={above left},
  rankine C/.style={below left}, rankine D/.style={above left}}
\begin{tikzpicture}
\begin{axis}[width=7cm,height=6cm,ymode=log,
  xlabel={$h$ (kJ/kg)},ylabel={$p$ (bar)},
  xmin=0,xmax=3500,ymin=0.5,ymax=80]
  \LCPAddDiagramFamily{PH}{quality}[
    fluid=Water,quality values={0,1},labels=false,
    boundary color=gray,boundary style={thin}]
  \LCPAddCycle{PH}{rankine}{loop}[color=blue,mark endpoints=true]
  \pgfplotsinvokeforeach{A,B,C,D} {
    \node[rankine #1] at (axis cs:
      \LCPStateCoordinate{PH}{rankine}{#1}{x},
      \LCPStateCoordinate{PH}{rankine}{#1}{y}) {$#1$};
  }
\end{axis}
\end{tikzpicture}\hfill
\begin{tikzpicture}
\begin{axis}[width=7cm,height=6cm,
  xlabel={$s$ (kJ/(kg K))},ylabel={$T$ (K)},
  xmin=0,xmax=8,ymin=300,ymax=750]
  \LCPAddDiagramFamily{TS}{quality}[
    fluid=Water,quality values={0,1},labels=false,
    temperature axis unit=kelvin,
    boundary color=gray,boundary style={thin}]
  \LCPAddCycle{TS}{rankine}{loop}[color=blue,mark endpoints=true]
  \pgfplotsinvokeforeach{A,B,C,D} {
    \node[rankine #1] at (axis cs:
      \LCPStateCoordinate{TS}{rankine}{#1}{x},
      \LCPStateCoordinate{TS}{rankine}{#1}{y}) {$#1$};
  }
\end{axis}
\end{tikzpicture}
\endgroup
```

For the numerical report, the next example repeats the same state and component construction so it can also be run independently. Query signed powers with the explicit mass flow; use specific work and boiler heat to calculate thermal efficiency. The three exports can then be used without reading values off a plot.

Pump work rate received: 1259.3 W.
Turbine work rate received: −143 264.5 W.
Net work rate received: −142 005.3 W.
Thermal efficiency: 0.2386.

```
\begingroup
\LCPDeclareModel[fluid=Water,reference state=DEF]{rankine}
\LCPSetup{temperature axis unit=kelvin}
\LCPDefineState{rankine}{A}{p=1bar,T=80C}
\LCPDefineComponent{rankine}{pump}[
  type=pump,from=A,to=B,to state={p=50bar},efficiency=0.80]
\LCPDefineComponent{rankine}{boiler}[
  type=heat exchanger,from=B,to=C,to state={T=450C}]
\LCPDefineComponent{rankine}{turbine}[
  type=turbine,from=C,to=D,to state={p=1bar},efficiency=0.85]
\LCPDefineComponent{rankine}{condenser}[
  type=heat exchanger,from=D,to=A]
\LCPDefineCycle{rankine}{loop}{pump,boiler,turbine,condenser}
\LCPBalanceSetup{mass flow=0.2}
\edef\PumpPower{\LCPComponentValue{rankine}{pump}{work_rate}}
\edef\TurbinePower{\LCPComponentValue{rankine}{turbine}{work_rate}}
\edef\NetPower{\LCPCycleValue{rankine}{loop}{work_rate}}
\edef\NetWork{\LCPCycleValue{rankine}{loop}{work}}
\edef\BoilerHeat{\LCPComponentValue{rankine}{boiler}{heat}}
\pgfkeys{/pgf/fpu=true}
\pgfmathparse{-\NetWork/\BoilerHeat}
\pgfmathfloattofixed{\pgfmathresult}
\edef\ThermalEfficiency{\pgfmathresult}
\pgfkeys{/pgf/fpu=false}
\par Pump work rate received:
  \num[round-mode=places,round-precision=1]{\PumpPower}\ W.
\par Turbine work rate received:
  \num[round-mode=places,round-precision=1]{\TurbinePower}\ W.
\par Net work rate received:
  \num[round-mode=places,round-precision=1]{\NetPower}\ W.
\par Thermal efficiency:
  \num[round-mode=places,round-precision=4]{\ThermalEfficiency}.
\LCPLogBalance{rankine}{loop}
\LCPExportBalance{rankine}{loop}{build/rankine-balances.dat}
\LCPExportCycleTable{PH}{rankine}{loop}{build/rankine-ph.dat}
\LCPExportCycleTable{TS}{rankine}{loop}{build/rankine-ts.dat}
\endgroup
```

The net work received is negative: the plant supplies shaft power. Its thermal efficiency is $-w_{\mathrm{net}}/q_{\mathrm{boiler}}$, not the gross turbine output divided by boiler heat. The pump consumption must be subtracted. The exported tables retain SI states, sign conventions and native provenance; displayed axis units never change the balance. Changing mass flow scales powers without changing this efficiency or the resolved endpoints. The boiler and condenser boundary temperatures remain unspecified, so the total entropy generation of this cycle is unavailable.

34.6 Machine phase limits and ideal references

A pump requires a strict pressure rise and liquid inlet, ideal outlet and actual outlet. Saturated liquid (`q=0`) is accepted as a thermodynamic boundary, without a claim of positive suction margin. Native phases `liquid` and `supercritical_liquid` are accepted: the latter allows dense liquid above critical pressure while below critical temperature. Wet feeds, vapour, critical points and general supercritical states are rejected. No vapour fraction or efficiency is silently adjusted to make a pump admissible.

A turbine requires a strict pressure decrease and a dry saturated-vapour or gas-like inlet, with the same gas-phase classification as the compressor. Its ideal and actual outlets may additionally lie on the subcritical liquid–vapour equilibrium dome, including its saturation boundaries. The quality then describes a homogeneous equilibrium bulk state. Droplet slip, blade erosion and wet-stage losses are not modelled. A compressed-liquid outlet and a liquid inlet are outside this turbine model; hydraulic expanders are a separate problem. Each native flash and both outlet phases must be valid.

For pumps and turbines, a supplied existing outlet is checked using $10^{-7}\max(1, |\Delta h|, |\Delta h_{\text{target}}|)$ in J/kg. The entropy decrease tolerance is $10^{-7}\max(1, |\Delta h|/T_{\text{in}}, |\Delta h|/T_{\text{out}})$ in J/(kg K), with temperatures in kelvin. These scales use differences rather than absolute enthalpy/entropy references. Raw residuals within tolerance are retained, not clamped to zero. The compressor and valve retain the previously described consistency checks.

Unit efficiency reuses the exact ideal snapshot and permits the ordinary isentropic sampler in any projection. This grouped example also demonstrates checking an already defined outlet without changing it:

Work received by the ideal turbine:−842 732.6 J/kg.

```
\begingroup
\LCPDeclareModel[fluid=Water]{idealMachines}
\LCPDefineState{idealMachines}{liquid}{p=1bar,q=0}
\LCPDefineComponent{idealMachines}{pump}[
  type=pump,from=liquid,to=pumped,to state={p=50bar},efficiency=1]
\LCPDefineComponent{idealMachines}{checked-pump}[
  type=pump,from=liquid,to=pumped,efficiency=1]
\LCPDefineState{idealMachines}{steam}{p=50bar,T=450C}
\LCPDefineComponent{idealMachines}{turbine}[
  type=turbine,from=steam,to=expanded,to state={p=1bar},efficiency=1]
\LCPDefineComponent{idealMachines}{checked-turbine}[
  type=turbine,from=steam,to=expanded,efficiency=1]
\edef\TurbineWork{\LCPComponentValue{idealMachines}{turbine}{work}}
Work received by the ideal turbine:
\num[round-mode=places,round-precision=1]{\TurbineWork}\ J/kg.
\endgroup
```

An invalid phase, redundant outlet property, missing efficiency or unsupported native state rejects the whole definition: neither a component nor a partial outlet is published. Efficiencies outside $(0, 1]$ and comma-decimal values are errors. The custom-component route remains available for caller-supplied relations with different assumptions, but does not certify those assumptions.

34.7 Component balances from Lua

The methods live on the same model object as the state/process API. All getters return detached snapshots. This program uses an explicit efficiency, inspects the ideal outlet, and demonstrates specific work and power without plotting.

```
local lcp = require("luacoolprop")
local model = lcp.models.new{fluid="R134a"}
model:define_state("A", "T=0C,q=1")
model:define_component("compress", {
  type="compressor", from="A", to="B",
  to_state="p=10bar", efficiency=0.75,
})
local device = model:component("compress")
local balance = model:component_balance("compress", {mass_flow=0.1})
assert(balance.work > 0 and balance.heat == 0)
assert(balance.entropy_generation > 0)
assert(device.component.ideal_outlet.s == device.from.s)
print(balance.work, balance.work_rate) - J/kg, W
```

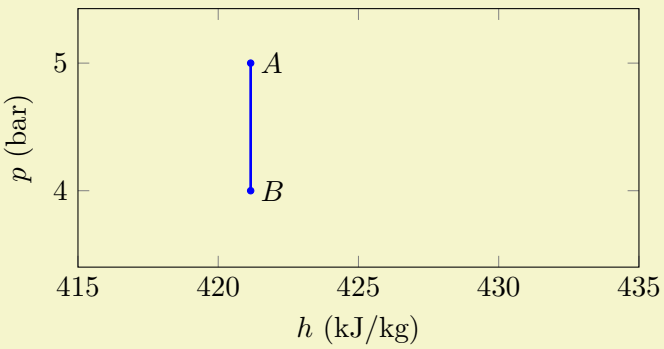
34.8 An adiabatic pressure loss in a refrigerant line

A pipe, filter or fitting can be represented by a lumped passive element when its pressure loss is known and kinetic and potential energy changes are negligible. The model does not calculate the loss from its geometry. For an adiabatic element with no shaft work,

$$h_{\text{out}} = h_{\text{in}}, \qquad s_{\text{gen}} = s_{\text{out}} - s_{\text{in}} \geq 0.$$

Pressure loss does not imply a decrease of enthalpy: flow work is already included in h. Subtracting an extra $v\Delta p$ would count it twice. The temperature may change with a real fluid, and flashing may occur.

Here R134a enters at 5 bar and 30 °C. A specified loss of 1 bar sets the outlet to 4 bar. The line follows the isenthalpic equilibrium locus; it does not describe the internal spatial pressure profile of a pipe. The same construction can be projected into PV, TS, HS or PT by changing the projection argument and the axis labels/limits.



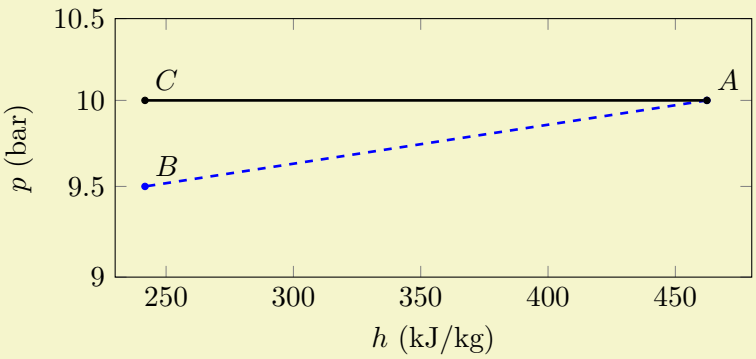
Outlet temperature: 300.91 K.
Entropy generated: 16.504 J/(kg K).

```
\LCPDeclareModel[fluid=R134a]{lineLoss}
\LCPDefineState{lineLoss}{A}{p=5bar,T=30C}
\LCPDefineComponent{lineLoss}{filter}[
  type=pressure loss,from=A,to=B,
  pressure drop=1E5Pa,thermal model=adiabatic]
\begin{tikzpicture}
\begin{axis}[width=9cm,height=5cm,ymode=log,
  xlabel={$h$ (kJ/kg)},ylabel={$p$ (bar)},
  xmin=415,xmax=435,ymin=3.5,ymax=5.5,
  xtick={415,420,425,430,435},ytick={4,5},yticklabels={4,5}]
  \LCPAddModelProcess{PH}{lineLoss}{filter}[
    color=blue,mark endpoints=true]
  \pgfplotsinvokeforeach{A,B} {
    \node[right] at (axis cs:
      \LCPStateCoordinate{PH}{lineLoss}{#1}{x},
      \LCPStateCoordinate{PH}{lineLoss}{#1}{y}) {$#1$};
  }
\end{axis}
\end{tikzpicture}
\edef\LossEntropy{\LCPComponentValue{lineLoss}{filter}{entropy_generation}}
\edef\OutletTemperature{\LCPStateValue{lineLoss}{B}{T}}
\par Outlet temperature:
  \num[round-mode=places,round-precision=2]{\OutletTemperature}\ K.
\par Entropy generated:
  \num[round-mode=places,round-precision=3]{\LossEntropy}\ J/(kg K).
```

The equivalent new-outlet pressure routes are `pressure ratio=0.8` or `to state={p=4bar}`. Select exactly one route: providing both a drop and a ratio is an error, even if they agree numerically. Pressure drops are nonnegative, ratios satisfy $0 < r \leq 1$, and the resulting pressure must be positive. A pressure-loss element requires an explicit pressure specification; `pressure drop=0` is a useful identity limit. An existing outlet requires an explicit drop or ratio, used to check its pressure. No existing state is modified. The thermal model is mandatory, and `adiabatic` rejects any `heat` option, even zero.

34.9 A cooler with an imposed pressure loss

A single-stream cooler has no shaft work, so $q = h_{\text{out}} - h_{\text{in}}$. Specify its outlet temperature (or another independent thermal property) and one pressure route. The example cools R134a from 80 °C at 10 bar to 30 °C, retaining 95% of the inlet pressure. Without the loss option, the same outlet temperature would be reached at 10 bar. Both cases use the same inlet and the same property model.



Heat received: −220 683.6 J/kg.
Heat rate received: −22 068.4 W.

```
\LCPDeclareModel[fluid=R134a]{coolingLoss}
\LCPDefineState{coolingLoss}{A}{p=10bar,T=80C}
\LCPDefineComponent{coolingLoss}{cooler}[
  type=heat exchanger,from=A,to=B,
  pressure ratio=0.95,to state={T=30C}]
\LCPDefineComponent{coolingLoss}{isobaric}[
  type=heat exchanger,from=A,to=C,to state={T=30C}]
\LCPBalanceSetup{mass flow=0.1}
\begin{tikzpicture}
\begin{axis}[width=10cm,height=5cm,ymode=log,
  xlabel={$h$ (kJ/kg)},ylabel={$p$ (bar)},
  xmin=230,xmax=480,ymin=9,ymax=10.5,
  ytick={9,9.5,10,10.5},yticklabels={9,9.5,10,10.5}]
  \LCPAddModelProcess{PH}{coolingLoss}{cooler}[
    color=blue,mark endpoints=true]
  \LCPAddModelProcess{PH}{coolingLoss}{isobaric}[
    color=black,mark endpoints=true]
  \pgfplotsinvokeforeach{A,B,C} {
    \node[above right] at (axis cs:
      \LCPStateCoordinate{PH}{coolingLoss}{#1}{x},
      \LCPStateCoordinate{PH}{coolingLoss}{#1}{y}) {$#1$};
  }
\end{axis}
\end{tikzpicture}
\edef\CoolerHeat{\LCPComponentValue{coolingLoss}{cooler}{heat}}
\edef\CoolerPower{\LCPComponentValue{coolingLoss}{cooler}{heat_rate}}
\par Heat received:
  \num[round-mode=places,round-precision=1]{\CoolerHeat}\ J/kg.
\par Heat rate received:
  \num[round-mode=places,round-precision=1]{\CoolerPower}\ W.
```

The blue dashed connector joins calculated endpoints only: the total heat and pressure loss do not determine where condensation starts, the distributed heat flux or an intermediate pressure–enthalpy law. The black isobar is a conserved-property locus. Do not read intermediate states from the blue line; changing its style to solid cannot supply missing physical information. Both negative heat values denote heat rejected by the refrigerant. An outlet quality such as `q=0` or `q=1` is also possible at a valid subcritical saturation pressure. A pressure–temperature pair on saturation cannot determine quality and is rejected by the native property solver.

34.10 Prescribed heat, existing outlets and unavailable entropy

For an exchanger, `heat` is an alternative to an outlet thermal property. It is a signed specific energy, in J/kg by default; `-20kJ/kg` removes 20 kJ/kg. For a passive heated line, explicitly choose `thermal model=specified heat`. These constructions determine $h_{\mathrm{out}} = h_{\mathrm{in}} + q$, not a heat rate. Supply mass flow only to later balance queries; W is not an accepted construction unit. No default flow, reservoir temperature or heat-transfer law is guessed.

Added heat: 2000 J/kg; checked heat: 2000 J/kg.
Zero-loss adiabatic entropy generation: 0 J/(kg K).

```
\LCPDeclareModel[fluid=R134a]{heatedLine}
\LCPDefineState{heatedLine}{A}{p=5bar,T=30C}
\LCPDefineComponent{heatedLine}{pipe}[
  type=pressure loss,from=A,to=B,pressure ratio=0.9,
  thermal model=specified heat,heat=2kJ/kg]
\LCPDefineComponent{heatedLine}{checked}[
  type=heat exchanger,from=A,to=B,
  pressure drop=0.5bar,heat=2kJ/kg]
\LCPDefineComponent{heatedLine}{netZero}[
  type=pressure loss,from=A,to=C,to state={p=4bar},
  thermal model=specified heat,heat=0]
\LCPDefineComponent{heatedLine}{identity}[
  type=pressure loss,from=A,to=D,
  pressure drop=0,thermal model=adiabatic]
\edef\AddedHeat{\LCPComponentValue{heatedLine}{pipe}{heat}}
\edef\CheckedHeat{\LCPComponentValue{heatedLine}{checked}{heat}}
\edef\IdentityEntropy{\LCPComponentValue{heatedLine}{identity}{entropy_generation}}
\par Added heat: \num{\AddedHeat}\ J/kg;
  checked heat: \num{\CheckedHeat}\ J/kg.
\par Zero-loss adiabatic entropy generation:
  \num{\IdentityEntropy}\ J/(kg K).
```

The checked exchanger reuses state B without overwriting it. For a new outlet, `heat` and outlet thermal data are redundant and rejected. Existing outlet enthalpy is a consistency check instead. Unknown keys, nonfinite numbers, comma decimals and inconsistent existing endpoints fail before either the component or a new state is published. New passive heat comparisons use enthalpy *differences*, making their tolerance independent of the chosen enthalpy reference. The native residual is retained in `energy_residual`; it is not silently forced to zero.

Even with `heat=0`, a specified-net-heat model leaves `entropy_generation` unavailable. Heat could enter and leave through different boundary temperatures. Its connector remains schematic. Requesting that unavailable field raises an error, and balance tables represent it as `nan`. If a single uniform reservoir temperature T_b is independently known, the user may evaluate

$$s_{\mathrm{gen}} = s_{\mathrm{out}} - s_{\mathrm{in}} - q/T_b.$$

This expression assumes one heat-transfer boundary at that temperature; the package never infers T_b from inlet or outlet temperatures. For an ideal gas with $c_p = 1000$ and $R = 250$ J/(kg K), heated from 300 to 350 K with $p_{\mathrm{out}}/p_{\mathrm{in}} = 0.8$, the model gives $q = 50$ kJ/kg and $\Delta s = c_p \ln(350/300) - R \ln(0.8)$. A 400 K reservoir then gives $s_{\mathrm{gen}} \simeq 84.94$ J/(kg K). This analytical illustration is not a constant-heat-capacity approximation used for the real-fluid examples.

These lumped models do not size pipes or valves. Darcy–Weisbach, local loss coefficients, Cv/Kv, choking, acceleration, elevation, two-phase pressure-drop correlations, branches and distributed thermal/flow solvers require additional models. Do not count a single physical loss twice as both an exchanger loss and a separate pipe. The same numerical endpoints, assumptions and schematic-path metadata are retained in every projection and in exported tables.

Part V
Developer reference

35 Architecture

The package is split into three layers:

File	Responsibility
`luacoolprop.lua`	stable Lua facade; loads the adjacent private runtime components
`luacoolprop-*.lua`	native ABI, input validation, thermodynamic state resolution, sampling, projections, records, serialization, and TeX output
`luacoolprop.tex`	generic TeX interface, `pgfkeys` trees, PGFPlots integration, wrappers around Lua calls
`luacoolprop.sty`	LaTeX wrapper loading PGFPlots and the generic TeX layer
`t-luacoolprop.tex`	ConTeXt wrapper loading the ConTeXt PGFPlots frontend and the generic TeX layer
`p-luacoolprop.tex`	plain TeX wrapper loading the generic TeX layer

The Lua module is format-neutral. TeX-facing functions live under `M.tex`; diagram generators live under `M.diagram`. Internal helpers use `snake_case`; TeX macros use the `\LCP...` prefix; PGF keys are lowercase words separated by spaces.

35.1 Lua runtime components and installation

Require only `luacoolprop`; the public namespaces and diagram registry are the application interface. All adjacent `luacoolprop-*.lua` files are mandatory runtime components. Install the complete set beside the facade in `tex/luatex/luacoolprop/lua/`. When the facade is a symbolic link, its components are resolved beside the target file. Missing components cause an installation error rather than a fallback to a different installed revision.

The native component owns the FFI declarations, cached library handle and native state objects. Options and state components validate input and physical consistency. The model component owns reference-state locks and pure-fluid checks. Sampling and projection components compute curves; family-record components select them without typesetting. Serialization and rendering components emit PGFPlots code. The registry exposes the structured API, while the TeX component owns printing, coordinate exports and the one-use axis/plot record cache. The cache and cache-codec components implement optional per-curve/per-process data reuse, bounded memory storage and atomic disk publication. The models component owns named, immutable SI states and reusable process/cycle definitions; the components module adds explicit steady-flow models, signed heat/work balances and powers. Models-tex manages their scoped TeX handles. The tables component exports numerical networks and processes with provenance for independent PGFPlots consumers. The Lua API reference lists the individual files.

Components import private helpers through explicit local bindings. Their initialization order is acyclic, does not evaluate thermodynamic states, and changes neither application globals nor Lua's search path. Application code must not require internal components directly. The library is loaded lazily when a property or diagram request needs it. Automatic label placement remains an external responsibility of `pgfplots-autonode`.

36 PGFPlots lifecycle and label emission

PGFPlots first collects plot data, determines axis limits and transformations, and renders the plots only at the end of the axis. LuaCoolProp respects this lifecycle: curve-family macros emit the full `\addplot` coordinate list immediately. When a curve has a label, the label is emitted as a trailing `\pgfplotsautonode` command attached to that plot. No Lua-side automatic placement pass is run for an axis that enables automatic placement.

This design has two consequences:

- PGFPlots autoscaling remains native because all numeric coordinates are available during the survey phase.
- Collision avoidance uses final PGFPlots geometry through `pgfplots-autonode`, rather than an approximate independent model in Lua.

37 Adaptive curve sampling

Isolines are sampled adaptively in display-like coordinates: $(h, \log p)$ for PH, $(\log v, \log p)$ for PV, (s, T) for TS, and (s, h) for HS. The sampler starts from a coarse grid in $\log p$, evaluates the midpoint of each segment, compares it with the midpoint of the straight chord, and recursively subdivides until the error estimate is below a family-specific tolerance or the maximum depth is reached. Isothermal two-phase plateaus are special: only the saturated vapor and saturated liquid endpoints are emitted for the horizontal plateau.

Constant-quality curves receive a second adaptive pass when their pressure range enters the upper half of the subcritical interval. This pass uses $\log_{10} \epsilon$, where $\epsilon = (p_c - p)/p_c$, as its independent variable over the upper half of the saturation-pressure range. It applies the same family tolerance and the projection's own display-coordinate error metric. Consequently the rapidly changing PH, PV, TS, and HS saturation branches and the PT coexistence locus are resolved without a fluid-specific critical-pressure cutoff. The common exact critical-limit record is then appended separately because quality itself is undefined at p_c.

38 Thermodynamic state resolution

Endpoint and process specifications are parsed as key-value lists. Pressures, temperatures, enthalpies, entropies, densities, specific volumes, and qualities are normalised to SI units before calls to CoolProp. Saturated states require a quality and one saturation coordinate, normally pressure or temperature. General single-phase states are resolved from two independent intensive properties. When a process type declares a conserved property, LuaCoolProp first tries to read the property from the process keys; if it is absent, it attempts to infer it from a sufficiently defined endpoint.

Input is strict at this boundary. Numbers must be finite decimal values and may use an `e` or `E` exponent; only the full stop is a decimal separator. Unit suffixes, boolean values, grid modes, presets, and every numeric-list element are validated in full. Invalid and reversed explicit bounds are reported rather than repaired silently.

For an overdetermined state, the resolver freezes one independent CoolProp input pair, computes the canonical state from that pair, and compares every other explicitly supplied property with the result. The completed endpoints are then compared once more with the conserved process property. A mismatch reports the supplied and resolved SI values, their absolute difference, and the applicable absolute/relative tolerance; consequently a valid isobar can never contain endpoints at two different pressures.

39 Naming conventions

The Lua source uses `snake_case` for local functions and table fields. Public TeX-facing Lua entry points are grouped under `M.tex`; diagram algorithms are grouped under `M.diagram`. User macros use the `\LCP...` prefix. User-level keys are lowercase English words separated by spaces. Generated PGFPlots curve names are stable ASCII identifiers, for example `lcp-ph-q-0p5`, `lcp-ph-T-40C`, `lcp-ph-s-1p6kJkgK`, and `lcp-ph-v-0p01m3kg`.

40 Lua API: scope, contracts, and safe use

The public Lua API separates native calls, diagrams, named models and TeX output. Top-level functions load and wrap CoolProp's C ABI. Functions under `M.diagram` compute thermodynamic data and return Lua tables or PGFPlots strings. Functions under `M.tex` are the narrow output bridge used by the generic TeX layer. A standalone Lua program should normally use the native, diagram and named-model interfaces. The `M.models` namespace owns reusable thermodynamic constructions; diagrams remain projections of their stored SI states.

The complete field-by-field reference is also maintained as `doc/luatex/luacoolprop/luacoolprop-lua-api.md`. Public functions carry LDoc comments next to their implementations in the Lua runtime components; the Markdown reference adds longer examples, extension rules, and a contributor checklist. The file `examples/lua-properties.lua` illustrates property calculations through `texlua` as part of `scripts/build-examples.sh`.

40.1 Loading the module and identifying both versions

Use a local module variable. Loading the Lua module does not by itself require a hard-coded library path; the first property call triggers the same search as `M.load_library()`.

Load LuaCoolProp from texlua

```
local lcp = require("luacoolprop")

local info = lcp.library_info()
io.write("LuaCoolProp ", lcp._VERSION, "\n")
io.write("CoolProp ", info.version, "\n")
io.write("CoolProp revision ", info.revision, "\n")
io.write("Actual library ", info.path, "\n")
io.write("Adapter ", info.adapter, "\n")
```

`M._VERSION` describes this package. In contrast, `M.version()` and `M.gitrevision()` query the loaded external CoolProp library. Record the native identity, actual path, configuration and reference convention alongside reproducible results.

Selection and lifetime. The first nonempty selection is authoritative: an explicit Lua path or the TeX `library` key, then `LUACOOLPROP_LIB`, then `COOLPROP_LIB`. A bad selection fails without falling back to another installation. With none of these set, the loader tries platform filenames in the current directory, the active TeX resolver, and finally the operating system's dynamic loader. Absent or unloadable defaults are skipped; an identified but incompatible binary stops the search. A later request to switch a loaded library raises an error: start a fresh compiler process.

Native diagnostics. `M.library_info()` loads if necessary and returns an independent deep copy. Its fields are `path` (actual readable binary, with POSIX symlinks resolved), `selection` (original spelling), `version`, `revision`, `adapter`, `os`, `architecture`, `abi`, and `capabilities`. The ABI table gives byte sizes for `int`, `long`, `double`, `bool` and `pointer`. The capability map contains:

- Boolean flags `scalar_properties`, `abstract_state`, and `canonical_fluid_identity`;

- `managed_reference_backend` (`HEOS`) and `string_success_status` (1);
- the sorted `required_symbols` list and the typed `configuration_keys` map.

The cache identity and exported table manifests include the adapter, ABI and capabilities, together with native and runtime hashes.

`M.loaded_library()` reports the original successful selection without triggering loading, or `nil` before loading. It remains the second return value of `M.load_library(path)`; the first is an opaque native handle. Use `library_info().path` to identify the actual file rather than a possibly relative selection or loader name. After applying a TeX `library` key through the first calculation (for example `\LCPDeclareFluid`), the same Lua diagnostic is available inside `\directlua`.

Raw FFI handle

After loading, `M.C` contains the raw FFI handle. It is an implementation escape hatch, not a stable LuaCoolProp interface. Prefer documented wrappers: they validate Lua argument types, convert C strings, and translate reported C errors into contextual Lua errors.

40.2 SI-unit contract

Low-level calls preserve CoolProp's SI convention. No implicit display-unit conversion occurs at this boundary.

Quantity	Lua field	Unit
pressure	`p`	Pa
temperature	`T`	K
mass enthalpy	`h`	J/kg
mass entropy	`s`	J/(kg K)
mass density	`rho`	kg/m^3
specific volume	`v`	m^3/kg
vapor quality	`q`	dimensionless, normally from 0 to 1

Diagram point records additionally contain scaled plot coordinates. For PH, the defaults give `x=h*1e-3` in kJ/kg and `y=p*1e-5` in bar. For PV, `x=v` is in m^3/kg and the same pressure scale is used. TS defaults to `x=s*1e-3` and `y=T`; HS defaults to `x=s*1e-3` and `y=h*1e-3`. Semantic axis-unit options couple these conversions to their labels, including the additive Celsius offset on a linear TS ordinate. Raw SI fields remain present on every point and should be used for thermodynamic work.

40.3 Errors and recovery boundaries

Load failures, invalid public arguments, reported CoolProp errors, and invalid process specifications raise Lua errors. They do not return an ambiguous `nil,error` pair. Catch an error only where recovery or additional context is possible. In particular, wrapping every property call in `pcall` would conceal invalid fluid models. The adaptive sampler is an intentional exception: it protects individual sample evaluations because a valid curve can cross a region where one property pair is undefined.

Catch an error at an application boundary

```
local lcp = require("luacoolprop")

local ok, result = xpcall(function()
  return lcp.propsSI("H", "P", 1e5, "Q", 1, "R134a")
end, debug.traceback)

if not ok then
  io.stderr:write(result, "\n")
  os.exit(1)
end

io.write(string.format("h = %.3f kJ/kg\n", result * 1e-3))
```

Diagram entry points accept only HEOS fluids reported as pure by CoolProp. They reject mixtures before applying algorithms which assume one triple point, one critical point, and one saturation dome; low-level wrappers remain mixture-capable. Enthalpy and entropy are relative properties. Select `DEF`, `IIR`, `ASHRAE`, or `NBP` with `set_reference_state` during initialization. The first diagram call locks that choice for its canonical native fluid, including all aliases and the explicit `HEOS::` prefix, and process metadata records it. Managed references reject other backends instead of modifying HEOS offsets for an unrelated model; low-level property and state wrappers remain backend-capable. Energy balances should use enthalpy differences computed under one convention.

Vapor quality is defined only in the two-phase region below the critical state. When a plotted quality curve reaches the critical pressure, LuaCoolProp adds a common graphical limiting record evaluated from T_c and ρ_c. That record is marked `critical_limit=true` and `quality_defined=false`; it has no thermodynamic quality value. Thus the shared endpoint represents convergence of the subcritical curves, not the simultaneous existence of every quality at the critical state.

40.4 Library metadata and direct property wrappers

String metadata. The string-returning metadata functions are:

- `M.global_param_string(param,n)`;
- `M.parameter_information_string(param,n)`; and
- `M.fluid_param_string(fluid,param,n)`.

Their buffer length `n` is optional and defaults to 4096 bytes; it must be an integer from 1 to 16777216. The parameter-information wrapper requests the native `long` description and needs at least 5 bytes for its input selector. Failed native string calls raise errors rather than returning an apparently valid empty result. `M.fluid_param_string_len(fluid,param)` can be used before requesting unusually long metadata.

Read fluid and parameter metadata

```
local lcp = require("luacoolprop")

local aliases = lcp.fluid_param_string("R134a", "aliases")
local information = lcp.parameter_information_string("Hmass")
io.write(aliases, "\n", information, "\n")
```

Two-input property call. `M.propsSI(output,name1,value1,name2,value2,fluid)` is the principal high-level CoolProp wrapper. All keys are CoolProp identifiers and both inputs are SI values. The mixed-case `M.PropsSI` name is an accepted alias.

Compute two properties at a saturated state

```
local lcp = require("luacoolprop")
local pressure = 2.5e5
local quality = 1.0

local h = lcp.propsSI("Hmass", "P", pressure, "Q", quality, "R134a")
local T = lcp.propsSI("T", "P", pressure, "Q", quality, "R134a")
```

Other direct wrappers. `M.props1SI(fluid,output)` obtains a single fluid property. `M.hapropsSI(...)` accepts the three input pairs required by humid-air calculations. `M.phaseSI(...)` returns a textual phase name. Accepted aliases preserve CoolProp's spellings `Props1SI`, `HAPropsSI`, and `PhaseSI`.

Indices and validation. `M.is_valid_fluid_string(fluid)` returns a boolean. `M.param_index(param)` and `M.input_pair_index(pair)` return numeric indices owned by the loaded CoolProp version. An application may cache an index for a hot loop, but must never persist it or reuse it with another CoolProp build. `M.saturation_ancillary(...)` exposes the corresponding C-API correlation and follows its SI-unit contract.

Global CoolProp configuration. `M.set_config_string`, `M.set_config_double`, `M.set_config_bool`, and `M.set_debug_level` mutate process-wide CoolProp state. Configure them once before calculations and record non-default settings. Pass a real Lua boolean to `set_config_bool`: the string `"false"` is truthy in Lua and therefore means true. `M.get_debug_level()` reads the current numeric level. The `configuration_keys` capability map gives the names and types supported by the detected revision. Unknown keys and wrong setter types are rejected before calling C. Integer-valued configuration keys are reported as `integer`, but the consumed C API has no integer setter: do not pass them to `set_config_double`. Configure only keys available in that binary.

Inspect a configuration capability

```
local lcp = require("luacoolprop")
local keys = lcp.library_info().capabilities.configuration_keys
if keys.REFPROP_RESOLVE_COOLPROP_ALIASES == "bool" then
  lcp.set_config_bool("REFPROP_RESOLVE_COOLPROP_ALIASES", true)
end
io.write("Delimiter type: ", keys.LIST_STRING_DELIMITER, "\n")
```

40.5 Managed AbstractState objects

`M.AbstractState(backend,fluids)` constructs a Lua wrapper around one native CoolProp handle. The method table is exposed as `M.State` for introspection, but instances must be created by the constructor.

- `state:update(pair,value1,value2)` accepts a CoolProp input-pair name or numeric index, updates the native object, and returns the same state for chaining;
- `state:keyed_output(param)` accepts a property name or index and returns an SI value;
- `state:phase()` returns the numeric CoolProp phase index;
- `state:fluid_names()` returns the native fluid specification; and
- `state:free()` releases the native handle and is idempotent.

Always free a state deterministically. The attached FFI finalizer is only a safety net because Lua does not guarantee when garbage collection occurs.

AbstractState with one deterministic cleanup point

```
local lcp = require("luacoolprop")
local state

local ok, result = xpcall(function()
  state = lcp.AbstractState("HEOS", "R134a")
  state:update("PT_INPUTS", 3e5, 293.15)
  return {
    h = state:keyed_output("Hmass"),
    s = state:keyed_output("Smass"),
    phase = state:phase(),
  }
end, debug.traceback)

if state ~= nil then
  state:free()
  state = nil
end
if not ok then error(result, 0) end

io.write(string.format("h=%.3f kJ/kg, s=%.4f kJ/(kg K)\n",
  result.h * 1e-3, result.s * 1e-3))
```

Never copy `state.handle`, invoke the native free function directly, or use a state after `free()`. Finalizers must remain non-throwing and no application logic should depend on their execution order at engine shutdown.

40.6 Fluid-constant table

`M.fluid_constants(fluid,library)` returns a fresh table. The optional `library` path is passed to the cached loader. Unavailable properties may be nil.

Field	Meaning	Unit
`fluid`	requested fluid specification	–
`pcrit`	critical pressure	Pa
`ptriple`	triple-point pressure	Pa
`Tcrit`	critical temperature	K
`Ttriple`	triple-point temperature	K
`Tmin`, `Tmax`	model temperature bounds	K
`rhocrit`	critical mass density	kg/m^3

40.7 Structured diagram interface

The preferred accessors are `M.diagram.get_type("PH")`, `M.diagram.get_type("PV")`, and `M.diagram.get_type("TS")`, `M.diagram.get_type("HS")`, and `M.diagram.get_type("PT")`. They decouple application code from the lowercase storage fields `M.diagram.ph`, `M.diagram.pv`, `M.diagram.ts`, `M.diagram.hs`, and `M.diagram.pt`. All implementations contain these renderers:

- `axis_style(opts)` returns data-dependent PGFPlots axis keys;
- `plots(opts)` returns all enabled families as PGFPlots code;
- `family(opts,name)` returns one canonical family;
- `process_points(opts)` returns sampled data and metadata;

- `process_plot(opts)` returns PGFPlots code and metadata;
- each entry in the implementation's `families` table provides its own `curve` and `plots` functions; and
- the corresponding `*_plots(opts)` functions select values and serialize one family.

The computation layer never calls `tex.sprint`. This separation makes the renderers usable from `texlua`, test code, and custom TeX bridges.

40.8 Point-record schema and adaptive sampling

Every curve renderer returns an array in plotting order. Every point contains `x`, `y`, `p`, and `t=log10(p)`. PH points contain `h` and use scaled enthalpy for X; PV points contain `v` and `rho` and use scaled specific volume for X. TS points use scaled entropy for X and an offset, scaled temperature for Y while retaining both raw SI properties. Family-specific records may also contain `q`, `T`, or `s`, always in SI units. Samples are not uniformly spaced: midpoint-error subdivision adapts to curvature, and invalid state pairs can split a curve into valid segments.

For a quality curve that enters the upper half of the subcritical interval, the high-pressure portion is also sampled adaptively in `log10((pcrit-p)/pcrit)`. This reduced-pressure pass uses the same `tolerance`, `max_depth`, and projection-specific coordinate weights as the main pass. Tightening the quality-family sampling controls therefore refines TS and HS curves all the way to their common critical limit; it does not merely select a denser fixed pressure table.

The array also exposes topology metadata: `segments`, `domain_gap_count`, `omitted_sample_count`, and `partially_omitted`. Its flat compatibility view marks each new component with `_break_before`; the serializer inserts a PGFPlots jump. Thus no line is ever fabricated across an invalid CoolProp domain. The `domain_policy` option selects silent, warning, or error diagnostics without changing the topology.

Generate and inspect one PH isotherm

```
local lcp = require("luacoolprop")
local ph = lcp.diagram.get_type("PH")

local points = ph.isotherm_curve({
  fluid = "R134a",
  pressure_min = 1e5,
  pressure_max = 2e6,
  enthalpy_scale = 1e-3,
  pressure_scale = 1e-5,
  isotherm_tolerance = 0.25,
}, 293.15)

for index, point in ipairs(points) do
  io.write(string.format("%d %.8g %.8g\n", index, point.x, point.y))
end
```

Generate and inspect one log-log PV isotherm

```
local lcp = require("luacoolprop")
local pv = lcp.diagram.get_type("PV")

local points = pv.isotherm_curve({
  fluid = "R134a",
  pressure_min = 1e5,
  pressure_max = 5e6,
  specific_volume_scale = 1,
  pressure_scale = 1e-5,
  isotherm_tolerance = 0.18,
}, 293.15)

for index, point in ipairs(points) do
  io.write(string.format("%d %.8g %.8g %.8g\n",
    index, point.x, point.y, point.rho))
end
```

Generate and inspect one TS isenthalp

```
local lcp = require("luacoolprop")
local ts = lcp.diagram.get_type("TS")

local points = ts.isenthalp_curve({
  fluid = "R134a",
  pressure_min = 1e5,
  pressure_max = 2e6,
  entropy_scale = 1e-3,
  temperature_scale = 1,
  isenthalp_tolerance = 0.10,
}, 300e3)

for index, point in ipairs(points) do
  io.write(string.format("%d %.8g %.8g %.8g\n",
    index, point.x, point.y, point.h))
end
```

Single-curve functions require explicit, positive pressure limits and expect the second argument in SI units: quality is dimensionless, temperature is kelvin, entropy is J/(kg K), enthalpy is J/kg, and specific volume is m^3/kg.

40.9 Shared option contract and declared capabilities

Lua option names use `snake_case`. The structured API accepts semantic `*_axis_unit` fields whenever a named display unit is required. Raw scales are advanced low-level controls. Every type normalizes through the same boundary and validates singular canonical family switches against its own registry declaration.

Field	Default	Contract
`fluid`	`"R134a"`	CoolProp fluid name
`library`	search	explicit shared-library candidate
`reference_state`	`"DEF"`	locked DEF, IIR, ASHRAE, or NBP convention
`pressure_min`	fluid based	lower pressure in Pa
`pressure_max`	fluid based	upper pressure in Pa
`enthalpy_scale`	`1e-3`	multiplier from SI enthalpy to plot x
`pressure_scale`	`1e-5`	multiplier from SI pressure to plot y
`coord_digits`	6	serialization precision
`initial_intervals`	family based	initial log-pressure intervals
`max_depth`	family based	maximum adaptive recursion depth
`tolerance`	family based	midpoint error threshold
`domain_policy`	ignore	omitted-domain diagnostic
`log_weight`	30	pressure contribution to error
`quality`	true	include quality curves
`isotherm`	false	include isotherms
`isentrope`	false	include isentropes
`isochore`	false	include isochores
`labels`	false	request labels
`label_placement`	autonode	label backend selection

PH, PV, TS, HS, and PT are peer implementations. Their declared background families are:

- PH: quality, isotherm, isentrope, and isochore;
- PV: quality, isotherm, and isentrope;
- TS: quality and isenthalp; and
- HS: quality, isochore, isotherm, and isobar.

An unavailable family switch is an error. Process paths nevertheless support all six conserved properties in every projection.

PV uses `specific_volume_scale` (default 1) and `log_x_weight` (default 30) to control the volume term in the log–log midpoint error.

TS uses the shared pressure, quality, label, style, autonode, and override fields. Its coordinate fields are `entropy_scale` (default `1e-3`) and `temperature_scale` (default `1`). Set `isenthalp=true` to enable its second background family and select that family with `enthalpy_values`, `enthalpy_min`, `enthalpy_max`, `enthalpy_step`, or `enthalpy_count`. Explicit values default to kJ/kg; `enthalpy_unit="si"` selects J/kg. The `temperature_weight` field converts vertical deviations in kelvin into the adaptive midpoint error for the linear TS axis.

Family-specific sampling controls prefix the generic name, for example `isotherm_tolerance`, `isentrope_max_depth`, and `isochore_initial_intervals`. Selection fields include `quality_values`, `temperature_values`, `entropy_values`, and `specific_volume_values`. A Lua value list is passed as a comma-separated string because the same normalized representation arrives from the generic TeX layer.

Return PGFPlots code without printing to TeX

```
local lcp = require("luacoolprop")
local ph = lcp.diagram.get_type("PH")

local code = ph.plots({
  fluid = "R134a",
  quality = true,
  isotherm = true,
  isentrope = false,
  isochore = false,
  quality_values = "0,0.25,0.5,0.75,1",
  temperature_values = "-20,0,20,40",
  temperature_unit = "celsius",
  labels = true,
})
io.write(code, "\n")
```

The returned string contains trusted TeX source. Styles and label text are not security-sanitized; never forward untrusted external input directly into them.

40.10 Lua process API

Every built-in diagram table provides `process_points(opts)`. Thus `ph`, `pv`, `ts`, and `hs` each require a process `type` plus `from` and `to` state strings. Canonical process types are `isobar`, `isotherm`, `isentropic`, `isenthalpic`, `quality`, and `isochore`. Bare state numbers use SI units; explicit suffixes such as `bar` and `C` make application code easier to review.

Resolve a process and consume its metadata

```
local lcp = require("luacoolprop")
local ph = lcp.diagram.get_type("PH")

local points, metadata = ph.process_points({
  fluid = "R134a",
  type = "isentropic",
  from = "pressure=2bar,quality=1",
  to = "pressure=10bar",
})

io.write(metadata.id, " ", metadata.kind, "\n")
io.write(string.format("%d points, h2=%.3f kJ/kg\n",
  #points, metadata.to.h * 1e-3))
```

Metadata contains a stable `id`, the normalized `kind`, the SI `constant`, and completed `from` and `to` state tables. The corresponding `process_plot(opts)` returns serialized PGFPlots code followed by the same metadata. Process style fields include `process_color`, `process_style`, `mark_endpoints`, `marker_style`, `label`, `label_pos`, and `label_style`.

TS process metadata follows the same contract. Isotherms and isentropes are exact horizontal and vertical TS segments. Isobars use entropy as their sampling variable so a two-phase evaporation or condensation includes the constant-temperature saturation plateau; isochores use temperature, and quality paths and isenthalps use pressure.

HS process metadata likewise retains all SI properties while plotting scaled entropy as x and scaled enthalpy as y. Isentropes and isenthalps are exact vertical and horizontal segments; isobars use entropy, isotherms include their exact two-phase segment, isochores use temperature, and qualities use pressure.

PT process metadata plots scaled/offset temperature as x and scaled pressure as y. Isobars and isotherms are exact horizontal and vertical segments; isentropes, isenthalps, isochores, and quality paths use logarithmic pressure. Every quality path projects onto the same liquid–vapour coexistence locus, because PT coordinates do not contain enough information to recover quality.

40.11 The built-in registry and custom diagram types

`M.diagram.register_type(code,implementation)` requires an uppercase identifier, rejects duplicate registration, assigns `implementation.type`, and returns the registered table. `M.diagram.get_type(code)` either returns it or raises an error listing available types.

A new implementation should provide the following fields:

- `plots` and `axis_style`;
- `family` and a `families` registry; and
- `process_points` and `process_plot`.

Numerical renderers must remain independent of the global `tex` table.

PH, PV, TS, HS, and PT are registered by the built-in module. They expose the same core renderer and process shape; PH families are quality, isotherm, isentrope, and isochore; PV families are quality, isotherm, and isentrope; TS families are quality and isenthalp; HS families are quality, isochore, isotherm, and isobar; PT families are phase envelope, isentrope, isenthalp, and isochore.

Minimal shape of a custom XY extension

```
local lcp = require("luacoolprop")

local xy = {
  plots = function(opts)
    assert(type(opts) == "table", "opts must be a table")
    return "% XY plots would be serialized here"
  end,
  axis_style = function(_)
    return "xlabel={$T$},ylabel={$p$}"
  end,
  families = {},
}

lcp.diagram.register_type("XY", xy)
```

An extension module should not mutate the built-in PH, PV, TS, HS, or PT tables. Registration is process-local; a duplicate error should reveal an accidental double load rather than being silently ignored.

40.12 TeX bridge and direct entry points

`M.tex.print_diagram(type,operation,opts,family)` is the structured bridge. It accepts `plots`, `axis_style`, `family`, or `process`, selects the registered renderer, and writes the returned code with `tex.sprint`. It therefore requires an active LuaTeX `tex` table and is not a standalone `texlua` API.

The flat `M.diagram.ph_*` and `M.tex.print_ph_*` functions provide direct PH entry points. Mixed-case property functions mirror CoolProp names. Generic Lua integrations use lowercase wrappers and select an implementation with one of these equivalent registry calls:

- `M.diagram.get_type("PH")`;
- `M.diagram.get_type("PV")`;
- `M.diagram.get_type("TS")`;
- `M.diagram.get_type("HS")`; or
- `M.diagram.get_type("PT")`.

They use singular family names and full property names.

Lua implementation checklist

New Lua code must keep module state local, declare temporaries with `local`, validate arguments at public boundaries, preserve SI units at the CoolProp boundary, free native resources deterministically, keep finalizers non-throwing, and separate calculation from TeX output. Diagram functions return tables or strings; only `M.tex` writes to TeX. Automatic label placement remains exclusively in `pgfplots-autonode`.

41 Curve records and the autonode bridge

A curve record stores the curve identifier, family, thermodynamic value, points, PGFPlots style, label text, preferred label position, label style, priority, and per-curve overrides. The function `add_coords_plot_with_autonode` serializes one record as:

Autonode serialization

```
\addplot[name path=<id>,<style>] coordinates {<points>}
  \pgfplotsautonode[<options>]{<label>};
```

The bridge is intentionally small. LuaCoolProp forwards high-level preferences such as `label pos for`, `label fixed for`, `label style for`, and `label show for`; `pgfplots-autonode` remains responsible for candidate search, overlap minimization, border constraints, and final node drawing.

The implementation and numerical API of the external label solver are documented in `pgfplots-autonode-manual.pdf`. LuaCoolProp relies only on its public PGFPlots keys and trailing-path command.

42 Process validation

Process paths are specified by a `type`, a starting endpoint, and an ending endpoint. LuaCoolProp first parses units, then attempts to resolve each endpoint with CoolProp. If the conserved property is missing, the validator tries to infer it from one sufficiently defined endpoint. When inference is impossible, the error message names the missing property and suggests the minimal additional state variable, for example adding `p=...` or `T=...` to a saturated state defined by `Q=...`.

Redundant inputs are assertions, not display-only annotations. After choosing one independent pair and resolving the full state, LuaCoolProp checks all additional supplied properties against CoolProp. It subsequently checks the declared constant at both completed endpoints. Contradictions stop the build and the diagnostic contains both values, their difference, and the tolerance; no inconsistent value is silently retained in exported metadata.

A Complete executable tutorial definitions

The following files are the source of the tutorial figures in this manual. They are read into this document before the tutorials are evaluated, and the standalone worksheet and solution drivers read the same files. Each invocation shown beside a tutorial result therefore uses the definitions printed here. The shorter listings in the tutorial prose explain individual steps of these complete constructions; they are not independent programs.

A.1 R717 worksheet and solution

```
% Shared R717 PH diagram used by the student and teacher versions.
% The thin drivers select whether the cycle and reading aids are visible.


\usetikzlibrary{arrows.meta,intersections}
\pgfplotsset{compat=1.18}

\definecolor{RSevenTemperature}{RGB}{175,45,45}
\definecolor{RSevenVolume}{RGB}{35,75,175}
\definecolor{RSevenCycle}{RGB}{220,80,20}
\definecolor{RSevenGuide}{RGB}{75,90,115}

\tikzset{
  r717 state/.style={
    circle,fill=white,draw=RSevenCycle,very thick,
    inner sep=1.2pt,font=\sffamily\bfseries\small
  },
  r717 process label/.style={
    fill=white,fill opacity=.88,text opacity=1,
    rounded corners=1pt,inner sep=1.5pt,
    font=\sffamily\scriptsize,text=black
  }
}


\newcommand{\RSevenDiagram}{%
\LCPSetup{fluid=R717}
\begin{tikzpicture}
\begin{axis}[
  lcp fluid=R717,
  width=24cm,
  height=15cm,
  xmin=100,
  xmax=2200,
  ymin=0.07,
  ymax=140,
  ymode=log,
  xlabel={Specific enthalpy $h$ ($\mathrm{kJ\,kg^{-1}}$)},
  ylabel={Pressure $p$ (bar)},
  title={R717 pressure-enthalpy diagram},
  grid=both,
  major grid style={black!32},
  minor grid style={black!13},
  tick label style={font=\small},
  label style={font=\small},
  title style={font=\sffamily\large},
  clip mode=individual,
  auto node placement,
  auto node algorithm=repair,
  auto node bbox mode=oriented,
  auto node failure mode=hide-low-priority,
  auto node border margin=2pt,
```

```
  auto node candidates=41
]

  % Use explicit grids so the worksheet remains pedagogically stable when the
  % package presets evolve.  Every value is expressed in the displayed unit.
  \LCPAddPHQuality[
    pressure min=7000,
    pressure max=14000000,
    quality values={0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1},
    quality color=black,
    quality boundary color=black,
    interior style={line width=.32pt},
    boundary style={line width=.9pt},
    labels=true,
    quality label every=1,
    quality label pos=.18,
    quality label sloped=true,
    quality label node style={font=\scriptsize,fill=white,inner sep=.8pt}
  ]
  \LCPAddPHIsotherms[
    pressure min=7000,
    pressure max=14000000,
    temperature values={-60,-40,-20,0,20,40,60,80,100,120,140,160,180,200,220,240},
    isotherm color=RSevenTemperature,
    isotherm style={line width=.34pt,densely dotted},
    labels=true,
    isotherm label every=1,
    isotherm label pos=.72,
    isotherm label sloped=true,
    isotherm label node style={font=\scriptsize,fill=white,inner sep=.8pt}
  ]
  \LCPAddPHIsentropes[
    pressure min=7000,
    pressure max=14000000,
    entropy values={1,2,3,4,5,6,7,8},
    isentrope color=black,
    isentrope style={line width=.32pt},
    labels=true,
    isentrope label every=1,
    isentrope label pos=.22,
    isentrope label sloped=true,
    isentrope label node style={font=\scriptsize,fill=white,inner sep=.8pt}
  ]
  \LCPAddPHIsochores[
    pressure min=7000,
    pressure max=14000000,
    specific volume values={0.01,0.02,0.05,0.1,0.2,0.5,1,2,10,20},
    isochore color=RSevenVolume,
    isochore style={line width=.36pt,dashed},
    labels=true,
    isochore label every=1,
    isochore label pos=.55,
    isochore label sloped=true,
    isochore label node style={font=\scriptsize,fill=white,inner sep=.8pt}
  ]

\ifRSevenSolution
  % State data are calculated by CoolProp.  Stable path names let TikZ recover
  % all five states as intersections, while export coordinates publishes the
  % exact plotted endpoint values for annotations and reproducible logs.
  \LCPAddPHProcess[
    type=isentrope,
    from={pressure=3bar,quality=1},
    to={pressure=10bar},
```

```
  color=RSevenCycle,
  style={line width=1.35pt,-{Latex[length=2.2mm]}},
  mark endpoints=false,
  name=r717-ab,
  export coordinates=RSevenAB,
  log coordinates=true
]
\LCPAddPHProcess[
  type=isobar,
  pressure=10bar,
  from={entropy=6.2276889kJkgK},
  to={quality=1},
  color=RSevenCycle,
  style={line width=1.35pt,-{Latex[length=2.2mm]}},
  mark endpoints=false,
  name=r717-bc,
  export coordinates=RSevenBC
]
\LCPAddPHProcess[
  type=isobar,
  pressure=10bar,
  from={quality=1},
  to={quality=0},
  color=RSevenCycle,
  style={line width=1.35pt,-{Latex[length=2.2mm]}},
  mark endpoints=false,
  name=r717-cd,
  export coordinates=RSevenCD
]
\LCPAddPHProcess[
  type=isenthalp,
  from={pressure=10bar,quality=0},
  to={pressure=3bar},
  color=RSevenCycle,
  style={line width=1.35pt,-{Latex[length=2.2mm]}},
  mark endpoints=false,
  name=r717-de,
  export coordinates=RSevenDE
]
\LCPAddPHProcess[
  type=isobar,
  pressure=3bar,
  from={enthalpy=462.75654kJkg},
  to={quality=1},
  color=RSevenCycle,
  style={line width=1.35pt,-{Latex[length=2.2mm]}},
  mark endpoints=false,
  name=r717-ea,
  export coordinates=RSevenEA
]

% Each state is the intersection of the two physical processes that meet
% there.  No duplicated state coordinate is maintained by hand.
\path[name intersections={of=r717-ea and r717-ab,by=RSevenA}];
\path[name intersections={of=r717-ab and r717-bc,by=RSevenB}];
\path[name intersections={of=r717-bc and r717-cd,by=RSevenC}];
\path[name intersections={of=r717-cd and r717-de,by=RSevenD}];
\path[name intersections={of=r717-de and r717-ea,by=RSevenE}];

% Projection coordinates combine the state intersections with axis-edge
% anchors.  The |- and -| operations remain correct for a resized axis and
% for logarithmic pressure because TikZ performs them in canvas space.
\coordinate (RSevenAxisBottom) at (axis cs:100,.07);
\coordinate (RSevenLabelBottom) at (axis cs:100,.075);
```

```
  \path (RSevenD |- RSevenAxisBottom) coordinate (RSevenDBottom);
  \path (RSevenA |- RSevenAxisBottom) coordinate (RSevenABottom);
  \path (RSevenC |- RSevenAxisBottom) coordinate (RSevenCBottom);
  \path (RSevenB |- RSevenAxisBottom) coordinate (RSevenBBottom);
  \path (RSevenD |- RSevenLabelBottom) coordinate (RSevenDLabel);
  \path (RSevenA |- RSevenLabelBottom) coordinate (RSevenALabel);
  \path (RSevenC |- RSevenLabelBottom) coordinate (RSevenCLabel);
  \path (RSevenB |- RSevenLabelBottom) coordinate (RSevenBLabel);

  % Projection lines turn the teacher copy into a model graphical reading.
  \draw[RSevenGuide,densely dashed,line width=.55pt]
    (RSevenDBottom) - (RSevenD);
  \draw[RSevenGuide,densely dashed,line width=.55pt]
    (RSevenABottom) - (RSevenA);
  \draw[RSevenGuide,densely dashed,line width=.55pt]
    (RSevenCBottom) - (RSevenC);
  \draw[RSevenGuide,densely dashed,line width=.55pt]
    (RSevenBBottom) - (RSevenB);
  \draw[RSevenGuide,densely dashed,line width=.55pt]
    (RSevenA -| RSevenAxisBottom) - (RSevenA);
  \draw[RSevenGuide,densely dashed,line width=.55pt]
    (RSevenB -| RSevenAxisBottom) - (RSevenB);

  % Spell out the five marker commands because PGFPlots defers axis paths;
  % an ordinary \foreach loop variable would be out of scope at visualization.
  \filldraw[draw=RSevenCycle,fill=white,line width=1pt]
    (RSevenA) circle[radius=2pt];
  \filldraw[draw=RSevenCycle,fill=white,line width=1pt]
    (RSevenB) circle[radius=2pt];
  \filldraw[draw=RSevenCycle,fill=white,line width=1pt]
    (RSevenC) circle[radius=2pt];
  \filldraw[draw=RSevenCycle,fill=white,line width=1pt]
    (RSevenD) circle[radius=2pt];
  \filldraw[draw=RSevenCycle,fill=white,line width=1pt]
    (RSevenE) circle[radius=2pt];

  \node[r717 state,anchor=south east] at (RSevenA) {A};
  \node[r717 state,anchor=south west] at (RSevenB) {B};
  \node[r717 state,anchor=south east] at (RSevenC) {C};
  \node[r717 state,anchor=south east] at (RSevenD) {D};
  \node[r717 state,anchor=north east] at (RSevenE) {E};

  \node[r717 process label,rotate=56] at (axis cs:1664,5.4)
    {isentropic compression};
  \node[r717 process label] at (axis cs:1695,11.6) {desuperheating};
  \node[r717 process label] at (axis cs:1040,11.6) {condensation};
  \node[r717 process label,rotate=90] at (axis cs:438,5.4) {expansion};
  \node[r717 process label] at (axis cs:1030,2.55) {evaporation};

  \node[font=\sffamily\scriptsize,anchor=west,rotate=90,text=RSevenGuide]
    at (RSevenDLabel)
    {$h_D=h_E=\pgfmathprintnumber[fixed,precision=0]{\RSevenCDToX}$};
  \node[font=\sffamily\scriptsize,anchor=west,rotate=90,text=RSevenGuide]
    at (RSevenALabel)
    {$h_A=\pgfmathprintnumber[fixed,precision=0]{\RSevenABFromX}$};
  \node[font=\sffamily\scriptsize,anchor=west,rotate=90,text=RSevenGuide]
    at (RSevenCLabel)
    {$h_C=\pgfmathprintnumber[fixed,precision=0]{\RSevenBCToX}$};
  \node[font=\sffamily\scriptsize,anchor=west,rotate=90,text=RSevenGuide]
    at (RSevenBLabel)
    {$h_B=\pgfmathprintnumber[fixed,precision=0]{\RSevenABToX}$};
\fi

\end{axis}
```

```
\end{tikzpicture}
}
```

A.2 R134a air conditioner

```
% Advanced R134a PH air-conditioner tutorial; all states are recomputed.
% Six pictures reproduce the correction's progressive graphical reasoning.


\usetikzlibrary{arrows.meta,intersections}
\pgfplotsset{compat=1.18}

\definecolor{RAirTemperature}{RGB}{35,90,185}
\definecolor{RAirEntropy}{RGB}{35,125,65}
\definecolor{RAirCycle}{RGB}{205,55,35}
\definecolor{RAirGuide}{RGB}{80,88,105}

\tikzset{
  r134a state/.style={
    circle,draw=RAirCycle,fill=white,very thick,
    inner sep=1.2pt,font=\sffamily\bfseries\small
  },
  r134a note/.style={
    fill=white,fill opacity=.9,text opacity=1,rounded corners=1pt,
    inner sep=2pt,font=\sffamily\scriptsize,align=left
  }
}

\newcommand{\RAirBaseQuality}[1]{%
  \LCPAddPHQuality[
    pressure min=70000,pressure max=5500000,
    quality values={#1},
    quality color=black,quality boundary color=black,
    interior style={line width=.35pt},boundary style={line width=1pt},
    labels=false
  ]%
}

\newcommand{\RAirLowPressureGuide}[1]{%
  \LCPAddPHProcess[
    type=isobar,pressure=3bar,
    from={temperature=-10C},to={temperature=20C},
    name=#1,color=RAirGuide,style={line width=.8pt}
  ]%
}

\newcommand{\RAirHighPressureGuide}[1]{%
  \LCPAddPHProcess[
    type=isobar,pressure=18bar,
    from={temperature=40C},to={temperature=100C},
    name=#1,color=RAirGuide,style={line width=.8pt}
  ]%
}

\newcommand{\RAirStateMarker}[2]{%
  \filldraw[draw=RAirCycle,fill=white,line width=1pt]
    (#1) circle[radius=2pt];
  \node[r134a state,#2] at (#1) {};
}

\newcommand{\RAirDiagram}[1]{%
  \def\RAirTitle{Phase regions}%
```

```
\ifcase#1\relax
\or\def\RAirTitle{Where the ideal-gas approximation becomes plausible}%
\or\def\RAirTitle{Locating state 1 from pressure and temperature}%
\or\def\RAirTitle{Following the isentropic compression to state 2}%
\or\def\RAirTitle{Locating state 3 after isobaric cooling}%
\or\def\RAirTitle{Complete R134a air-conditioning cycle}%
\fi
\begin{tikzpicture}
\begin{axis}[
  lcp fluid=R134a,
  width=17cm,height=11cm,
  xmin=140,xmax=520,
  ymin=.7,ymax=55,ymode=log,
  xlabel={Specific enthalpy $h$ ($\mathrm{kJ\,kg^{-1}}$)},
  ylabel={Pressure $p$ (bar)},
  title={\RAirTitle},
  grid=both,
  major grid style={black!28},minor grid style={black!10},
  tick label style={font=\small},label style={font=\small},
  title style={font=\sffamily\large},
  clip mode=individual
]

\ifcase#1\relax
  \RAirBaseQuality{0,1}
  \node[r134a note] at (axis cs:190,20) {subcooled\\liquid};
  \node[r134a note] at (axis cs:310,5) {liquid-vapour\\equilibrium};
  \node[r134a note] at (axis cs:455,12) {superheated\\vapour};

\or
  \RAirBaseQuality{0,1}
  \LCPAddPHIsotherms[
    pressure min=70000,pressure max=5500000,
    temperature values={-40,-20,0,20,40,60,80,100,120,140,160},
    isotherm color=RAirTemperature,
    isotherm style={line width=.45pt},labels=false
  ]
  \draw[RAirCycle,dashed,line width=.9pt,rounded corners]
    (axis cs:390,.85) rectangle (axis cs:510,4.5);
  \node[r134a note,anchor=south east,text=RAirCycle]
    at (axis cs:507,4.7)
    {low-pressure superheated vapour:\\isotherms approach isenthalps};

\or
  \RAirBaseQuality{0,1}
  \LCPAddPHIsotherms[
    pressure min=70000,pressure max=5500000,
    temperature values={0,10},
    isotherm color=RAirTemperature,
    isotherm style={line width=.9pt},labels=false
  ]
  \RAirLowPressureGuide{r134a-stage-low}
  \LCPAddPHProcess[
    type=isotherm,temperature=5C,
    from={pressure=2bar},to={pressure=5bar},
    name=r134a-stage-t-one,color=black,style={draw=none}
  ]
  \path[name intersections={of=r134a-stage-low and r134a-stage-t-one,
    by=RAirStageOne}];
  \RAirStateMarker{RAirStageOne}{anchor=south west}
  \node[r134a note,anchor=south west] at (RAirStageOne)
    {state 1: $p_1=3\,\mathrm{bar}$, $T_1=5\,{}^\circ\mathrm C$};
  \node[r134a note,text=RAirTemperature] at (axis cs:245,2.45)
    {$0\,{}^\circ\mathrm C$};
```

```
  \node[r134a note,text=RAirTemperature] at (axis cs:245,3.75)
    {$10\,{}^\circ\mathrm C$};

\or
  \RAirBaseQuality{0,1}
  \LCPAddPHIsotherms[
    pressure min=70000,pressure max=5500000,
    temperature values={70,80},
    isotherm color=RAirTemperature,
    isotherm style={line width=.75pt},labels=false
  ]
  \LCPAddPHIsentropes[
    pressure min=70000,pressure max=5500000,
    entropy values={1.70,1.75},
    isentrope color=RAirEntropy,
    isentrope style={line width=.75pt},labels=false
  ]
  \RAirLowPressureGuide{r134a-stage-four-low}
  \RAirHighPressureGuide{r134a-stage-four-high}
  \LCPAddPHProcess[
    type=isentrope,
    from={pressure=3bar,temperature=5C},to={pressure=18bar},
    name=r134a-stage-one-two,color=RAirCycle,
    style={line width=1.4pt,-{Latex[length=2.2mm]}}
  ]
  \path[name intersections={of=r134a-stage-four-low and
    r134a-stage-one-two,by=RAirStageFourOne}];
  \path[name intersections={of=r134a-stage-one-two and
    r134a-stage-four-high,by=RAirStageFourTwo}];
  \RAirStateMarker{RAirStageFourOne}{anchor=south east}
  \RAirStateMarker{RAirStageFourTwo}{anchor=south west}
  \node[r134a note,anchor=east] at (RAirStageFourOne) {1};
  \node[r134a note,anchor=west] at (RAirStageFourTwo) {2};
  \node[r134a note,text=RAirEntropy,rotate=75]
    at (axis cs:421,6.5) {$s=1.70$ and $1.75\,\mathrm{kJ\,kg^{-1}\,K^{-1}}$};

\or
  \RAirBaseQuality{0,1}
  \LCPAddPHIsotherms[
    pressure min=70000,pressure max=5500000,
    temperature values={60},
    isotherm color=RAirTemperature,
    isotherm style={line width=.9pt},labels=false
  ]
  \LCPAddPHProcess[
    type=isentrope,
    from={pressure=3bar,temperature=5C},to={pressure=18bar},
    name=r134a-stage-five-one-two,color=RAirCycle,
    style={line width=1.2pt,-{Latex[length=2mm]}}
  ]
  \LCPAddPHProcess[
    type=isobar,pressure=18bar,
    from={entropy=1740.778583426JkgK},to={temperature=60C},
    name=r134a-stage-five-two-three,color=RAirCycle,
    style={line width=1.4pt,-{Latex[length=2.2mm]}}
  ]
  \LCPAddPHProcess[
    type=isotherm,temperature=60C,
    from={pressure=12bar},to={pressure=25bar},
    name=r134a-stage-five-t-three,color=black,style={draw=none}
  ]
  \path[name intersections={of=r134a-stage-five-one-two and
    r134a-stage-five-two-three,by=RAirStageFiveTwo}];
  \path[name intersections={of=r134a-stage-five-two-three and
```

```
    r134a-stage-five-t-three,by=RAirStageFiveThree}];
  \RAirStateMarker{RAirStageFiveTwo}{anchor=south west}
  \RAirStateMarker{RAirStageFiveThree}{anchor=south east}
  \node[r134a note,anchor=west] at (RAirStageFiveTwo) {2};
  \node[r134a note,anchor=east] at (RAirStageFiveThree)
    {3: $T_3=60\,{}^\circ\mathrm C$};

\or
  \RAirBaseQuality{0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1}
  \LCPAddPHIsotherms[
    pressure min=70000,pressure max=5500000,
    temperature values={0,10,60,70,80},
    isotherm color=RAirTemperature,
    isotherm style={line width=.42pt,densely dotted},labels=false
  ]
  \LCPAddPHIsentropes[
    pressure min=70000,pressure max=5500000,
    entropy values={1.70,1.75},
    isentrope color=RAirEntropy,
    isentrope style={line width=.5pt,dashed},labels=false
  ]
  \LCPAddPHProcess[
    type=isentrope,
    from={pressure=3bar,temperature=5C},to={pressure=18bar},
    name=r134a-cycle-one-two,export coordinates=RAirOneTwo,
    log coordinates=true,color=RAirCycle,
    style={line width=1.45pt,-{Latex[length=2.2mm]}}
  ]
  \LCPAddPHProcess[
    type=isobar,pressure=18bar,
    from={entropy=1740.778583426JkgK},to={temperature=60C},
    name=r134a-cycle-two-three,export coordinates=RAirTwoThree,
    color=RAirCycle,
    style={line width=1.45pt,-{Latex[length=2.2mm]}}
  ]
  \LCPAddPHProcess[
    type=isenthalp,
    from={pressure=18bar,temperature=60C},to={pressure=3bar},
    name=r134a-cycle-three-four,export coordinates=RAirThreeFour,
    color=RAirCycle,
    style={line width=1.45pt,-{Latex[length=2.2mm]}}
  ]
  \LCPAddPHProcess[
    type=isobar,pressure=3bar,
    from={enthalpy=287.4107701858kJkg},to={temperature=5C},
    name=r134a-cycle-four-one,export coordinates=RAirFourOne,
    color=RAirCycle,
    style={line width=1.45pt,-{Latex[length=2.2mm]}}
  ]
  \path[name intersections={of=r134a-cycle-four-one and
    r134a-cycle-one-two,by=RAirOne}];
  \path[name intersections={of=r134a-cycle-one-two and
    r134a-cycle-two-three,by=RAirTwo}];
  \path[name intersections={of=r134a-cycle-two-three and
    r134a-cycle-three-four,by=RAirThree}];
  \path[name intersections={of=r134a-cycle-three-four and
    r134a-cycle-four-one,by=RAirFour}];

  \coordinate (RAirAxisBottom) at (axis cs:140,.7);
  \path (RAirOne |- RAirAxisBottom) coordinate (RAirOneBottom);
  \path (RAirTwo |- RAirAxisBottom) coordinate (RAirTwoBottom);
  \path (RAirThree |- RAirAxisBottom) coordinate (RAirThreeBottom);
  \draw[RAirGuide,densely dashed]
    (RAirOneBottom)-(RAirOne);
```

```
    \draw[RAirGuide,densely dashed]
      (RAirTwoBottom)-(RAirTwo);
    \draw[RAirGuide,densely dashed]
      (RAirThreeBottom)-(RAirThree);
    \draw[RAirGuide,densely dashed]
      (RAirOne -| RAirAxisBottom)-(RAirOne);
    \draw[RAirGuide,densely dashed]
      (RAirTwo -| RAirAxisBottom)-(RAirTwo);

    \RAirStateMarker{RAirOne}{anchor=south east}
    \RAirStateMarker{RAirTwo}{anchor=south west}
    \RAirStateMarker{RAirThree}{anchor=south east}
    \RAirStateMarker{RAirFour}{anchor=north east}
    \node[r134a note,anchor=east] at (RAirOne) {1};
    \node[r134a note,anchor=west] at (RAirTwo) {2};
    \node[r134a note,anchor=east] at (RAirThree) {3};
    \node[r134a note,anchor=east] at (RAirFour) {4};

    \pgfmathsetmacro{\RAirTTwoC}{\RAirOneTwoToTemperatureK-273.15}
    \pgfmathsetmacro{\RAirTFourC}{\RAirThreeFourToTemperatureK-273.15}
    \pgfmathsetmacro{\RAirCompressorPower}{
      .1*(\RAirOneTwoToX-\RAirOneTwoFromX)}
    \pgfmathsetmacro{\RAirEvaporatorPower}{
      .1*(\RAirFourOneToX-\RAirFourOneFromX)}
    \pgfmathsetmacro{\RAirCOP}{
      \RAirEvaporatorPower/\RAirCompressorPower}
    \node[r134a note,anchor=north west] at (rel axis cs:.015,.985) {%
      \begin{tabular}{@{}c@{\,: }r@{\,}l@{\quad}r@{\,}l@{}}
      1 & \pgfmathprintnumber[fixed,precision=1]{\RAirOneTwoFromX}
        & $\mathrm{kJ\,kg^{-1}}$ & $T_1=5.0$ & $^\circ\mathrm C$\\
      2 & \pgfmathprintnumber[fixed,precision=1]{\RAirOneTwoToX}
        & $\mathrm{kJ\,kg^{-1}}$ &
        $T_2=\pgfmathprintnumber[fixed,precision=1]{\RAirTTwoC}$
        & $^\circ\mathrm C$\\
      3 & \pgfmathprintnumber[fixed,precision=1]{\RAirTwoThreeToX}
        & $\mathrm{kJ\,kg^{-1}}$ & $T_3=60.0$ & $^\circ\mathrm C$\\
      4 & \pgfmathprintnumber[fixed,precision=1]{\RAirThreeFourToX}
        & $\mathrm{kJ\,kg^{-1}}$ &
        $Q_4=\pgfmathprintnumber[fixed,precision=3]{\RAirThreeFourToQuality}$
        & \\
      \end{tabular}\\[-1mm]
      $\dot W_c=\pgfmathprintnumber[fixed,precision=2]{\RAirCompressorPower}
       \,\mathrm{kW}$,\quad
      $\dot Q_e=\pgfmathprintnumber[fixed,precision=2]{\RAirEvaporatorPower}
       \,\mathrm{kW}$,\quad
      $\mathrm{COP}=\pgfmathprintnumber[fixed,precision=2]{\RAirCOP}$
    };
  \fi

  \end{axis}
  \end{tikzpicture}%
}
```

A.3 Butane PV construction

```
% Five-stage butane PV tutorial with computed phase and process diagrams.
% All curves and numerical values are regenerated from the external CoolProp
% library; no imported chart is used by this executable source.


\usetikzlibrary{arrows.meta,intersections}
```

```
\pgfplotsset{compat=1.18}

\definecolor{ButaneTemperature}{RGB}{190,45,42}
\definecolor{ButaneQuality}{RGB}{35,105,205}
\definecolor{ButaneIdeal}{RGB}{20,135,80}
\definecolor{ButaneHeating}{RGB}{225,115,20}
\definecolor{ButaneGuide}{RGB}{80,88,105}

\tikzset{
  butane state/.style={
    circle,draw=black,fill=white,very thick,
    inner sep=1.4pt,font=\sffamily\bfseries\small
  },
  butane note/.style={
    fill=white,fill opacity=.92,text opacity=1,rounded corners=1pt,
    inner sep=2.5pt,font=\sffamily\scriptsize,align=left
  }
}

\newcommand{\ButanePVBackground}{%
  \LCPAddPVQuality[
    fluid=Butane,
    pressure min=50000,pressure max=15000000,pressure scale=1,
    quality values={0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1},
    quality symbol=x,
    quality color=ButaneQuality,quality boundary color=ButaneQuality,
    interior style={line width=.38pt},boundary style={line width=.9pt},
    labels=true,quality label every=1,quality label pos=.2,
    quality label sloped=true,
    initial intervals=28,max depth=9,tolerance=.1,log x weight=45
  ]%
  \LCPAddPVIsotherms[
    fluid=Butane,
    pressure min=50000,pressure max=15000000,pressure scale=1,
    temperature unit=K,
    temperature values={275,285,295,305,315,335,365,415,425.125,
      515,615,715},
    temperature symbol=T,
    isotherm color=ButaneTemperature,
    isotherm style={line width=.42pt},
    curve style for={lcp-pv-T-425p125-K}{densely dashed,line width=.9pt},
    labels=true,isotherm label every=1,isotherm label pos=.76,
    isotherm label sloped=true,
    initial intervals=28,max depth=9,tolerance=.1,log x weight=45
  ]%
}

\newcommand{\ButaneHeatingProcess}[1]{%
  \LCPAddPVProcess[
    fluid=Butane,type=isochore,pressure scale=1,
    from={temperature=305K,quality=0.3},
    to={temperature=335K},
    name=butane-isochoric-heating,
    export coordinates=ButaneHeatingState,
    coord digits=10,log coordinates=true,
    color=ButaneHeating,style={#1}
  ]%
}

\newcommand{\ButaneStateAPaths}{%
  \LCPAddPVProcess[
    fluid=Butane,type=isotherm,temperature=305K,pressure scale=1,
    from={pressure=1bar},to={pressure=10bar},
    name=butane-state-a-isotherm,color=black,style={draw=none}
```

```
  ]%
  \LCPAddPVProcess[
    fluid=Butane,type=quality,quality=0.3,pressure scale=1,
    from={temperature=275K},to={temperature=415K},
    name=butane-state-a-quality,color=black,style={draw=none}
  ]%
}

\newcommand{\ButaneStateA}{%
  \path[name intersections={of=butane-state-a-isotherm and
    butane-state-a-quality,
    by=ButaneA}];%
}

\newcommand{\ButaneStateMarker}[2]{%
  \filldraw[draw=black,fill=white,line width=1pt]
    (#1) circle[radius=2pt];%
  \node[butane state,#2] at (#1) {};%
}

\newcommand{\ButanePVDiagram}[1]{%
  \def\ButanePVTitle{Butane PV worksheet}%
  \ifcase#1\relax
  \or\def\ButanePVTitle{Phase regions and the critical isotherm}%
  \or\def\ButanePVTitle{Ideal-gas limit of the 515 K isotherm}%
  \or\def\ButanePVTitle{Tank state from two independent properties}%
  \or\def\ButanePVTitle{Heating a closed rigid butane tank}%
  \fi
  \begin{tikzpicture}
  \begin{loglogaxis}[
    lcp fluid=Butane,
    width=17cm,height=11cm,
    xmin=.001,xmax=1.4,
    ymin=50000,ymax=15000000,
    xlabel={Specific volume $v$ (\si{\cubic\metre\per\kilogram})},
    ylabel={Pressure $p$ (\si{\pascal})},
    title={\ButanePVTitle},
    grid=both,
    major grid style={black!28},minor grid style={black!10},
    tick label style={font=\small},label style={font=\small},
    title style={font=\sffamily\large},
    clip mode=individual,
    auto node placement,
    auto node algorithm=repair,
    auto node bbox mode=oriented,
    auto node failure mode=hide-low-priority
  ]
  \ButanePVBackground

  \ifcase#1\relax
    % The worksheet contains thermodynamic curves but no solution annotations.

  \or
    \node[butane note,anchor=west] at (axis cs:.00108,260000)
      {compressed\\liquid};
    \node[butane note] at (axis cs:.013,300000)
      {liquid-vapour\\equilibrium};
    \node[butane note] at (axis cs:.31,250000) {superheated\\vapour};
    \node[butane note] at (axis cs:.0033,9500000)
      {supercritical region:\\$T>T_c$ and $p>p_c$};
    \filldraw[draw=black,fill=white,line width=.8pt]
      (axis cs:\ButaneCriticalVolume,\LCPFluidPcritSI)
      circle[radius=1.7pt];
    \node[butane note,anchor=south west]
```

```
    at (axis cs:.0047,5100000)
    {$T_c=425.125\,\mathrm K$\\$p_c=3.796\,\mathrm{MPa}$};
  \node[butane note,text=ButaneTemperature,anchor=south west]
    at (axis cs:.011,4800000) {critical isotherm $\mathcal C$};

\or
  % For an ideal gas, p=(R/M)T/v.  With M=58.12 g/mol and
  % T=515 K, (R/M)T is approximately 7.367e4 Pa m^3/kg.
  \addplot[ButaneIdeal,line width=1.15pt,domain=.025:1.25,samples=2]
    {(8.314462618/0.05812*515)/x};
  \node[butane note,text=ButaneIdeal,anchor=south west]
    at (axis cs:.12,850000)
    {$p=(R/M)T/v$\\slope $-1$ in log-log coordinates};
  \draw[ButaneIdeal,-{Latex[length=2mm]},line width=.8pt]
    (axis cs:.24,520000) - (axis cs:.42,180000);
  \node[butane note,text=ButaneIdeal,anchor=north west]
    at (axis cs:.22,480000)
    {CoolProp and ideal-gas curves\\nearly coincide as $v$ increases};

\or
  % The invisible process exists only to export the exact endpoint data used
  % by the numerical result box.  Point A itself is placed by intersection.
  \ButaneStateAPaths
  \ButaneHeatingProcess{draw=none}
  \ButaneStateA
  \coordinate (ButaneAxisOrigin) at (axis cs:.001,50000);
  \path (ButaneA |- ButaneAxisOrigin) coordinate (ButaneABottom);
  \path (ButaneA -| ButaneAxisOrigin) coordinate (ButaneALeft);
  \draw[ButaneGuide,densely dashed]
    (ButaneABottom)-(ButaneA)-(ButaneALeft);
  \ButaneStateMarker{ButaneA}{anchor=south west}
  \node[butane note,anchor=south west] at (ButaneA)
    {$A$: $T=305\,\mathrm K$, $x=0.30$};
  \pgfmathsetmacro{\ButaneMassGram}{
    0.004/\ButaneHeatingStateFromSpecificVolumeSI}
  \node[butane note,anchor=north east] at (rel axis cs:.985,.985) {%
    $p_A=\qty[round-mode=figures,round-precision=4]
      {\ButaneHeatingStateFromPressureSI}{\pascal}$\\
    $v_A=\qty[round-mode=figures,round-precision=5]
      {\ButaneHeatingStateFromSpecificVolumeSI}
      {\cubic\metre\per\kilogram}$\\
    $m=V/v_A=\qty[round-mode=figures,round-precision=3]
      {\ButaneMassGram}{\gram}$
  };

\or
  \ButaneStateAPaths
  \ButaneHeatingProcess{line width=1.6pt,-{Latex[length=2.6mm]}}
  \ButaneStateA
  \ButaneStateMarker{ButaneA}{anchor=south east}
  \node[butane note,anchor=east] at (ButaneA) {$A$: $x=0.30$};
  \filldraw[draw=ButaneHeating,fill=white,line width=1pt]
    (axis cs:\ButaneHeatingStateToX,\ButaneHeatingStateToY)
    circle[radius=2pt];
  \node[butane note,anchor=south west,text=ButaneHeating]
    at (axis cs:\ButaneHeatingStateToX,\ButaneHeatingStateToY)
    {$T=335\,\mathrm K$\\
     $x=\num[round-mode=places,round-precision=3]
       {\ButaneHeatingStateToQuality}$};
  \node[butane note,anchor=south east,text=ButaneHeating]
    at (axis cs:.035,1150000)
    {$V$ and $m$ constant\\$\Rightarrow v=V/m$ constant};
  \typeout{Butane tutorial initial pressure (Pa):
    \ButaneHeatingStateFromPressureSI}
```

```
    \typeout{Butane tutorial initial specific volume (m3/kg):
      \ButaneHeatingStateFromSpecificVolumeSI}
    \typeout{Butane tutorial final quality:
      \ButaneHeatingStateToQuality}
  \fi

  \end{loglogaxis}
  \end{tikzpicture}%
}
```

A.4 R1234yf swimming-pool heat pump

```
% Shared worksheet/solution source for an R1234yf swimming-pool heat pump.
% Every thermodynamic curve, cycle state, and reported value is regenerated.


\usetikzlibrary{arrows.meta,calc,intersections}
\pgfplotsset{compat=1.18}

\definecolor{RHeatPumpQuality}{RGB}{205,45,38}
\definecolor{RHeatPumpTemperature}{RGB}{35,85,205}
\definecolor{RHeatPumpEntropy}{RGB}{25,105,50}
\definecolor{RHeatPumpCycle}{RGB}{215,105,15}
\definecolor{RHeatPumpGuide}{RGB}{70,78,92}

\tikzset{
  r1234yf state/.style={
    circle,draw=RHeatPumpCycle,fill=white,very thick,
    inner sep=1.5pt,font=\sffamily\bfseries\small
  },
  r1234yf note/.style={
    fill=white,fill opacity=.93,text opacity=1,rounded corners=1pt,
    inner sep=2.5pt,font=\sffamily\scriptsize,align=left
  }
}

\newcommand{\RHeatPumpBackground}{%
  \LCPAddPHQuality[
    fluid=R1234yf,
    pressure min=100000,pressure max=8000000,
    enthalpy scale=1,pressure scale=1,
    quality values={0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1},
    quality symbol=x,
    quality color=RHeatPumpQuality,
    quality boundary color=RHeatPumpQuality,
    interior style={line width=.38pt},boundary style={line width=.9pt},
    labels=true,quality label every=1,quality label pos=.14,
    quality label sloped=true,
    initial intervals=28,max depth=9,tolerance=.11
  ]%
  \LCPAddPHIsotherms[
    fluid=R1234yf,
    pressure min=100000,pressure max=8000000,
    enthalpy scale=1,pressure scale=1,
    temperature unit=K,
    temperature values={253,263,273,283,293,303,313,
      323,333,343,353,363,373},
    isotherm color=RHeatPumpTemperature,
    isotherm style={line width=.42pt},
    labels=true,isotherm label every=1,isotherm label pos=.13,
    isotherm label sloped=true,
```

```
    initial intervals=28,max depth=9,tolerance=.11
  ]%
  \LCPAddPHIsentropes[
    fluid=R1234yf,
    pressure min=100000,pressure max=8000000,
    enthalpy scale=1,pressure scale=1,
    entropy unit=SI,
    entropy values={1500,1550,1600,1650,1700,
      1750,1800,1850,1900,1950,2000},
    isentrope color=RHeatPumpEntropy,
    isentrope style={line width=.42pt},
    labels=true,isentrope label every=1,isentrope label pos=.82,
    isentrope label sloped=true,
    initial intervals=28,max depth=9,tolerance=.11
  ]%
}

% Derive both operating pressures from the two saturation temperatures.  The
% short invisible quality paths expose the computed pressures as TeX macros;
% operating pressures are calculated rather than copied from a drawing.
\newcommand{\RHeatPumpOperatingPressures}{%
  \LCPAddPHProcess[
    fluid=R1234yf,type=quality,quality=1,
    from={temperature=-10C},to={temperature=-9C},
    enthalpy scale=1,pressure scale=1,
    name=r1234yf-low-saturation-helper,style={draw=none},
    export coordinates=RHeatPumpLowSaturation,coord digits=10
  ]%
  \LCPAddPHProcess[
    fluid=R1234yf,type=quality,quality=0,
    from={temperature=50C},to={temperature=49C},
    enthalpy scale=1,pressure scale=1,
    name=r1234yf-high-saturation-helper,style={draw=none},
    export coordinates=RHeatPumpHighSaturation,coord digits=10
  ]%
  \typeout{R1234yf saturation helper low pressure (Pa):
    \RHeatPumpLowSaturationFromPressureSI}%
  \typeout{R1234yf saturation helper high pressure (Pa):
    \RHeatPumpHighSaturationFromPressureSI}%
}

\newcommand{\RHeatPumpCyclePaths}{%
  % B -> C: adiabatic reversible compression.
  \LCPAddPHProcess[
    fluid=R1234yf,type=isentrope,
    from={pressure=\RHeatPumpLowSaturationFromPressureSI Pa,
      temperature=0C},
    to={pressure=\RHeatPumpHighSaturationFromPressureSI Pa},
    enthalpy scale=1,pressure scale=1,
    name=r1234yf-bc,export coordinates=RHeatPumpBC,
    coord digits=10,log coordinates=true,
    color=RHeatPumpCycle,
    style={line width=1.65pt,-{Latex[length=2.6mm]}}
  ]%
  % C -> D: desuperheating, condensation at 50 C, then 10 K subcooling.
  \LCPAddPHProcess[
    fluid=R1234yf,type=isobar,
    pressure=\RHeatPumpHighSaturationFromPressureSI Pa,
    from={entropy=\RHeatPumpBCToEntropySI JkgK},
    to={temperature=40C},
    enthalpy scale=1,pressure scale=1,
    name=r1234yf-cd,export coordinates=RHeatPumpCD,
    coord digits=10,log coordinates=true,
    color=RHeatPumpCycle,
```

```
    style={line width=1.65pt,-{Latex[length=2.6mm]}}
  ]%
  % D -> A: adiabatic throttle with no moving parts, hence h_A=h_D.
  \LCPAddPHProcess[
    fluid=R1234yf,type=isenthalp,
    from={pressure=\RHeatPumpHighSaturationFromPressureSI Pa,
      temperature=40C},
    to={pressure=\RHeatPumpLowSaturationFromPressureSI Pa},
    enthalpy scale=1,pressure scale=1,
    name=r1234yf-da,export coordinates=RHeatPumpDA,
    coord digits=10,log coordinates=true,
    color=RHeatPumpCycle,
    style={line width=1.65pt,-{Latex[length=2.6mm]}}
  ]%
  % A -> B: evaporation at -10 C followed by 10 K isobaric superheating.
  \LCPAddPHProcess[
    fluid=R1234yf,type=isobar,
    pressure=\RHeatPumpLowSaturationFromPressureSI Pa,
    from={enthalpy=\RHeatPumpDAToEnthalpySI Jkg},
    to={temperature=0C},
    enthalpy scale=1,pressure scale=1,
    name=r1234yf-ab,export coordinates=RHeatPumpAB,
    coord digits=10,log coordinates=true,
    color=RHeatPumpCycle,
    style={line width=1.65pt,-{Latex[length=2.6mm]}}
  ]%
}

\newcommand{\RHeatPumpStateIntersections}{%
  \path[name intersections={of=r1234yf-da and r1234yf-ab,
    by=RHeatPumpA}];%
  \path[name intersections={of=r1234yf-ab and r1234yf-bc,
    by=RHeatPumpB}];%
  \path[name intersections={of=r1234yf-bc and r1234yf-cd,
    by=RHeatPumpC}];%
  \path[name intersections={of=r1234yf-cd and r1234yf-da,
    by=RHeatPumpD}];%
}

\newcommand{\RHeatPumpStateMarker}[3]{%
  \filldraw[draw=RHeatPumpCycle,fill=white,line width=1pt]
    (#1) circle[radius=2.1pt];%
  \node[r1234yf state,#3] at (#1) {#2};%
}


\newcommand{\RHeatPumpDiagram}{%
\LCPSetup{fluid=R1234yf}
\begin{tikzpicture}
\begin{axis}[
  lcp fluid=R1234yf,
  width=24cm,height=15cm,
  xmin=100000,xmax=620000,
  ymin=100000,ymax=8000000,ymode=log,
  xlabel={Specific enthalpy $h$ (\si{\joule\per\kilogram})},
  ylabel={Pressure $p$ (\si{\pascal})},
  title={R1234yf pressure-enthalpy diagram},
  grid=both,
  major grid style={black!28},minor grid style={black!10},
  tick label style={font=\small},label style={font=\small},
  title style={font=\sffamily\large},
  clip mode=individual,
  auto node placement,
  % This background contains many labelled curves.  The greedy algorithm is
```

```
  % deterministic and keeps worksheet builds fast while still allowing the
  % autonomous pgfplots library to choose a readable position on each curve.
  auto node algorithm=greedy,
  auto node candidates=21,
  auto node bbox mode=oriented,
  auto node failure mode=hide-low-priority
]
  \RHeatPumpBackground

  \ifRHeatPumpSolution
    \RHeatPumpOperatingPressures
    \RHeatPumpCyclePaths
    \RHeatPumpStateIntersections

    \coordinate (RHeatPumpAxisBottom) at (axis cs:100000,100000);
    \path (RHeatPumpC |- RHeatPumpAxisBottom)
      coordinate (RHeatPumpCBottom);
    \path (RHeatPumpD |- RHeatPumpAxisBottom)
      coordinate (RHeatPumpDBottom);
    \draw[RHeatPumpGuide,densely dashed]
      (RHeatPumpCBottom)-(RHeatPumpC);
    \draw[RHeatPumpGuide,densely dashed]
      (RHeatPumpDBottom)-(RHeatPumpD);

    \RHeatPumpStateMarker{RHeatPumpA}{A}{anchor=north east}
    \RHeatPumpStateMarker{RHeatPumpB}{B}{anchor=south east}
    \RHeatPumpStateMarker{RHeatPumpC}{C}{anchor=south west}
    \RHeatPumpStateMarker{RHeatPumpD}{D}{anchor=south east}

    % LaTeX's expandable floating-point evaluator accepts the full SI values
    % without TeX's fixed-point dimension limit.  Division by 1000 returns kW.
    \edef\RHeatPumpCondenserPowerKW{\fpeval{
      5*(\RHeatPumpCDToEnthalpySI
        -\RHeatPumpCDFromEnthalpySI)/1000}}
    \edef\RHeatPumpEvaporatorPowerKW{\fpeval{
      5*(\RHeatPumpABToEnthalpySI
        -\RHeatPumpABFromEnthalpySI)/1000}}
    \edef\RHeatPumpCompressorPowerKW{\fpeval{
      5*(\RHeatPumpBCToEnthalpySI
        -\RHeatPumpBCFromEnthalpySI)/1000}}

    \node[r1234yf note,anchor=north west]
      at (rel axis cs:.012,.985) {%
      $p_e=\qty[round-mode=figures,round-precision=4]
        {\RHeatPumpLowSaturationFromPressureSI}{\pascal}$,\quad
      $p_c=\qty[round-mode=figures,round-precision=5]
        {\RHeatPumpHighSaturationFromPressureSI}{\pascal}$\\
      $h_C=\qty[round-mode=figures,round-precision=6]
        {\RHeatPumpCDFromEnthalpySI}{\joule\per\kilogram}$,\quad
      $h_D=\qty[round-mode=figures,round-precision=6]
        {\RHeatPumpCDToEnthalpySI}{\joule\per\kilogram}$\\
      $\mathcal P_1=\dot m(h_D-h_C)
        =\qty[round-mode=figures,round-precision=4]
          {\RHeatPumpCondenserPowerKW}{\kilo\watt}$
    };

    \node[r1234yf note,anchor=south west,text=RHeatPumpCycle]
      at (RHeatPumpA) {throttle:\\$h_A=h_D$};
    \node[r1234yf note,anchor=north,text=RHeatPumpCycle]
      at ($(RHeatPumpA)!.55!(RHeatPumpB)$) {evaporator};
    \node[r1234yf note,anchor=south east,text=RHeatPumpCycle]
      at ($(RHeatPumpB)!.55!(RHeatPumpC)$) {compressor:\\$s_C=s_B$};
    \node[r1234yf note,anchor=south,text=RHeatPumpCycle]
      at ($(RHeatPumpC)!.5!(RHeatPumpD)$) {condenser};
```

```
    \typeout{R1234yf heat-pump low pressure (Pa):
      \RHeatPumpLowSaturationFromPressureSI}
    \typeout{R1234yf heat-pump high pressure (Pa):
      \RHeatPumpHighSaturationFromPressureSI}
    \typeout{R1234yf heat-pump h_C (J/kg):
      \RHeatPumpCDFromEnthalpySI}
    \typeout{R1234yf heat-pump h_D (J/kg):
      \RHeatPumpCDToEnthalpySI}
    \typeout{R1234yf heat-pump condenser power received (kW):
      \RHeatPumpCondenserPowerKW}
  \fi
\end{axis}
\end{tikzpicture}
}
```

A.5 Methane Linde process

```
% Shared worksheet/solution source for methane liquefaction by the Linde
% process. All curves, states, paths, mass-flow rates, and powers are generated.


\usetikzlibrary{arrows.meta,calc}
\pgfplotsset{compat=1.18}

\definecolor{MLQuality}{RGB}{50,105,225}
\definecolor{MLTemperature}{RGB}{220,55,40}
\definecolor{MLEntropy}{RGB}{235,135,15}
\definecolor{MLVolume}{RGB}{30,145,75}
\definecolor{MLCompression}{RGB}{190,55,35}
\definecolor{MLCooling}{RGB}{30,105,185}
\definecolor{MLRegenerator}{RGB}{130,60,160}
\definecolor{MLSeparation}{RGB}{15,125,110}
\definecolor{MLGuide}{RGB}{75,82,95}

\tikzset{
  methane linde state/.style={
    circle,draw=black,fill=white,very thick,inner sep=1.4pt,
    font=\sffamily\bfseries\scriptsize
  },
  methane linde note/.style={
    fill=white,fill opacity=.94,text opacity=1,rounded corners=1pt,
    inner sep=2.5pt,font=\sffamily\scriptsize,align=left
  }
}

% Four colour-coded curve families are computed independently by CoolProp.
\newcommand{\MethaneLindeDenseBackground}{%
  \LCPAddPHQuality[
    fluid=Methane,pressure min=20000,pressure max=20000000,
    quality values={0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1},
    quality symbol=x,quality color=MLQuality,
    quality boundary color=MLQuality,
    interior style={line width=.36pt},boundary style={line width=.85pt},
    labels=true,quality label every=1,quality label pos=.13,
    quality label sloped=true,
    initial intervals=24,max depth=9,tolerance=.12
  ]%
  \LCPAddPHIsotherms[
    fluid=Methane,pressure min=20000,pressure max=20000000,
    temperature unit=K,
    temperature values={103,113,123,133,143,153,163,173,183,193,
```

```
      203,213,223,233,243,253,263,273,283,293,303,313,323,333,
      343,353,363,373},
    temperature symbol=T,isotherm color=MLTemperature,
    isotherm style={line width=.38pt},labels=true,
    % Retain the complete 10 K grid but label alternate curves.  This keeps
    % interpolation accurate without crowding entropy and volume labels.
    isotherm label every=2,isotherm label pos=.68,
    isotherm label sloped=true,
    initial intervals=24,max depth=9,tolerance=.12
  ]%
  \LCPAddPHIsentropes[
    fluid=Methane,pressure min=20000,pressure max=20000000,
    entropy unit=kjkgk,
    entropy values={1,1.5,2,2.5,3,3.5,4,4.5,5,5.5,6,6.5},
    entropy symbol=s,isentrope color=MLEntropy,
    isentrope style={line width=.42pt},labels=true,
    isentrope label every=2,isentrope label pos=.79,
    isentrope label sloped=true,
    initial intervals=22,max depth=8,tolerance=.14
  ]%
  \LCPAddPHIsochores[
    fluid=Methane,pressure min=20000,pressure max=20000000,
    specific volume unit=m3kg,
    specific volume values={0.005,0.01,0.02,0.05,0.1,0.2,0.5,1,2,5},
    specific volume symbol=v,isochore color=MLVolume,
    isochore style={line width=.42pt},labels=true,
    isochore label every=2,isochore label pos=.38,
    isochore label sloped=true,
    isochore initial intervals=22,isochore max depth=8,
    isochore tolerance=.14
  ]%
}

% Intermediate solution pages retain the phase envelope and quality grid, but
% avoid repeating the sixty-one-curve reference network on every page.
\newcommand{\MethaneLindeSparseBackground}{%
  \LCPAddPHQuality[
    fluid=Methane,pressure min=20000,pressure max=20000000,
    quality values={0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1},
    quality color=MLQuality,quality boundary color=MLQuality,
    interior style={line width=.3pt},boundary style={line width=.8pt},
    labels=false,initial intervals=20,max depth=8,tolerance=.14
  ]%
}

% Resolve the primitive states supplied by the exercise, then close the two
% coupled balances before calculating the three compression stages.  Helper
% paths are invisible; their globally exported SI values are the numerical API
% used by both the drawing and the correction.
\newcommand{\MethaneLindeCompute}{%
  \LCPAddPHProcess[
    fluid=Methane,type=isobar,pressure=1bar,
    from={temperature=7C},to={quality=1},
    name=ml-helper-feed-vapour,style={draw=none},
    export coordinates=MethaneLindeFeedVapour,coord digits=12
  ]%
  \LCPAddPHProcess[
    fluid=Methane,type=isobar,pressure=1bar,
    from={quality=0},to={quality=1},
    name=ml-helper-saturation,style={draw=none},
    export coordinates=MethaneLindeSaturation,coord digits=12
  ]%
  \LCPAddPHProcess[
    fluid=Methane,type=isobar,pressure=100bar,
```

```
  from={temperature=-63C},to={temperature=-82C},
  name=ml-helper-high-regenerator,style={draw=none},
  export coordinates=MethaneLindeHighRegenerator,coord digits=12
]%
\LCPAddPHProcess[
  fluid=Methane,type=isenthalp,
  from={pressure=100bar,temperature=-82C},to={pressure=1bar},
  name=ml-helper-throttle,style={draw=none},
  export coordinates=MethaneLindeThrottle,coord digits=12,
  log coordinates=true
]%

\edef\MethaneLindeQualityEight{\fpeval{
  \MethaneLindeThrottleToQuality}}%
\edef\MethaneLindeMassFlowOne{\fpeval{
  1/(1-\MethaneLindeQualityEight)}}%
\edef\MethaneLindeMassFlowOneBis{\fpeval{
  \MethaneLindeQualityEight*\MethaneLindeMassFlowOne}}%
\edef\MethaneLindeEnthalpyOneBisSI{\fpeval{
  \MethaneLindeSaturationToEnthalpySI
  -(\MethaneLindeMassFlowOne/\MethaneLindeMassFlowOneBis)
  *(\MethaneLindeHighRegeneratorToEnthalpySI
    -\MethaneLindeHighRegeneratorFromEnthalpySI)}}%
\edef\MethaneLindeEnthalpyOneSI{\fpeval{
  (\MethaneLindeFeedVapourFromEnthalpySI
    +\MethaneLindeMassFlowOneBis*\MethaneLindeEnthalpyOneBisSI)
  /\MethaneLindeMassFlowOne}}%

\LCPAddPHProcess[
  fluid=Methane,type=isentrope,
  from={pressure=1bar,enthalpy=\MethaneLindeEnthalpyOneSI Jkg},
  to={pressure=5bar},name=ml-helper-c-one,
  style={draw=none},export coordinates=MethaneLindeCOne,
  coord digits=12,log coordinates=true
]%
\LCPAddPHProcess[
  fluid=Methane,type=isobar,pressure=5bar,
  from={entropy=\MethaneLindeCOneToEntropySI JkgK},
  to={enthalpy=866kJkg},name=ml-helper-e-one,
  style={draw=none},export coordinates=MethaneLindeEOne,
  coord digits=12
]%
\LCPAddPHProcess[
  fluid=Methane,type=isentrope,
  from={pressure=5bar,enthalpy=866kJkg},to={pressure=25bar},
  name=ml-helper-c-two,style={draw=none},
  export coordinates=MethaneLindeCTwo,coord digits=12,
  log coordinates=true
]%
\LCPAddPHProcess[
  fluid=Methane,type=isobar,pressure=25bar,
  from={entropy=\MethaneLindeCTwoToEntropySI JkgK},
  to={enthalpy=840kJkg},name=ml-helper-e-two,
  style={draw=none},export coordinates=MethaneLindeETwo,
  coord digits=12
]%
\LCPAddPHProcess[
  fluid=Methane,type=isentrope,
  from={pressure=25bar,enthalpy=840kJkg},to={pressure=100bar},
  name=ml-helper-c-three,style={draw=none},
  export coordinates=MethaneLindeCThree,coord digits=12,
  log coordinates=true
]%
\LCPAddPHProcess[
```

```
    fluid=Methane,type=isobar,pressure=100bar,
    from={entropy=\MethaneLindeCThreeToEntropySI JkgK},
    to={temperature=-63C},name=ml-helper-e-three,
    style={draw=none},export coordinates=MethaneLindeEThree,
    coord digits=12
  ]%

  \edef\MethaneLindeThermalPowerOneKW{\fpeval{
    \MethaneLindeMassFlowOne*(866000-\MethaneLindeCOneToEnthalpySI)/1000}}%
  \edef\MethaneLindeThermalPowerTwoKW{\fpeval{
    \MethaneLindeMassFlowOne*(840000-\MethaneLindeCTwoToEnthalpySI)/1000}}%
  \edef\MethaneLindeThermalPowerThreeKW{\fpeval{
    \MethaneLindeMassFlowOne*(\MethaneLindeEThreeToEnthalpySI
      -\MethaneLindeCThreeToEnthalpySI)/1000}}%
  \edef\MethaneLindeCompressorOnePowerKW{\fpeval{
    \MethaneLindeMassFlowOne*(\MethaneLindeCOneToEnthalpySI
      -\MethaneLindeEnthalpyOneSI)/1000}}%
  \edef\MethaneLindeCompressorTwoPowerKW{\fpeval{
    \MethaneLindeMassFlowOne*(\MethaneLindeCTwoToEnthalpySI-866000)/1000}}%
  \edef\MethaneLindeCompressorThreePowerKW{\fpeval{
    \MethaneLindeMassFlowOne*(\MethaneLindeCThreeToEnthalpySI-840000)/1000}}%
  \edef\MethaneLindeIdealCompressionPowerKW{\fpeval{
    \MethaneLindeCompressorOnePowerKW
    +\MethaneLindeCompressorTwoPowerKW
    +\MethaneLindeCompressorThreePowerKW}}%
  \edef\MethaneLindeCoolingEffectiveness{\fpeval{
    -(\MethaneLindeThermalPowerOneKW
      +\MethaneLindeThermalPowerTwoKW
      +\MethaneLindeThermalPowerThreeKW)/1800}}%
}

\newcommand{\MethaneLindeCoordinates}{%
  \coordinate (MLZero) at
    (axis cs:\MethaneLindeFeedVapourFromX,\MethaneLindeFeedVapourFromY);%
  \coordinate (MLOne) at
    (axis cs:\fpeval{\MethaneLindeEnthalpyOneSI/1000},1);%
  \coordinate (MLOneBis) at
    (axis cs:\fpeval{\MethaneLindeEnthalpyOneBisSI/1000},1);%
  \coordinate (MLTwo) at
    (axis cs:\MethaneLindeCOneToX,\MethaneLindeCOneToY);%
  \coordinate (MLThree) at
    (axis cs:\MethaneLindeEOneToX,\MethaneLindeEOneToY);%
  \coordinate (MLFour) at
    (axis cs:\MethaneLindeCTwoToX,\MethaneLindeCTwoToY);%
  \coordinate (MLFive) at
    (axis cs:\MethaneLindeETwoToX,\MethaneLindeETwoToY);%
  \coordinate (MLSix) at
    (axis cs:\MethaneLindeCThreeToX,\MethaneLindeCThreeToY);%
  \coordinate (MLSeven) at
    (axis cs:\MethaneLindeHighRegeneratorFromX,
      \MethaneLindeHighRegeneratorFromY);%
  \coordinate (MLSevenBis) at
    (axis cs:\MethaneLindeHighRegeneratorToX,
      \MethaneLindeHighRegeneratorToY);%
  \coordinate (MLEight) at
    (axis cs:\MethaneLindeThrottleToX,\MethaneLindeThrottleToY);%
  \coordinate (MLNine) at
    (axis cs:\MethaneLindeSaturationToX,\MethaneLindeSaturationToY);%
  \coordinate (MLTen) at
    (axis cs:\MethaneLindeSaturationFromX,\MethaneLindeSaturationFromY);%
}

\def\MethaneLindeProcess[#1]{%
  \LCPAddPHProcess[#1,fluid=Methane,coord digits=10,
```

```
    style={line width=1.55pt,-{Latex[length=2.4mm]}}]}

\newcommand{\MethaneLindeDrawSeparation}{%
  \MethaneLindeProcess[type=isobar,pressure=100bar,
    from={temperature=-63C},to={temperature=-82C},
    name=ml-seven-sevenbis,color=MLRegenerator]%
  \MethaneLindeProcess[type=isenthalp,
    from={pressure=100bar,temperature=-82C},to={pressure=1bar},
    name=ml-sevenbis-eight,color=MLRegenerator,
    log coordinates=true]%
  \MethaneLindeProcess[type=isobar,pressure=1bar,
    from={enthalpy=\MethaneLindeThrottleToEnthalpySI Jkg},to={quality=1},
    name=ml-eight-nine,color=MLSeparation]%
  \MethaneLindeProcess[type=isobar,pressure=1bar,
    from={enthalpy=\MethaneLindeThrottleToEnthalpySI Jkg},to={quality=0},
    name=ml-eight-ten,color=MLSeparation]%
  \MethaneLindeProcess[type=isobar,pressure=1bar,
    from={quality=1},to={enthalpy=\MethaneLindeEnthalpyOneBisSI Jkg},
    name=ml-nine-onebis,color=MLRegenerator]%
}

\newcommand{\MethaneLindeDrawMixer}{%
  \MethaneLindeProcess[type=isobar,pressure=1bar,
    from={temperature=7C},to={enthalpy=\MethaneLindeEnthalpyOneSI Jkg},
    name=ml-zero-one,color=MLSeparation]%
  \MethaneLindeProcess[type=isobar,pressure=1bar,
    from={enthalpy=\MethaneLindeEnthalpyOneBisSI Jkg},
    to={enthalpy=\MethaneLindeEnthalpyOneSI Jkg},
    name=ml-onebis-one,color=MLSeparation]%
}

\newcommand{\MethaneLindeDrawStageOne}{%
  \MethaneLindeProcess[type=isentrope,
    from={pressure=1bar,enthalpy=\MethaneLindeEnthalpyOneSI Jkg},
    to={pressure=5bar},name=ml-one-two,color=MLCompression,
    log coordinates=true]%
  \MethaneLindeProcess[type=isobar,pressure=5bar,
    from={entropy=\MethaneLindeCOneToEntropySI JkgK},
    to={enthalpy=866kJkg},name=ml-two-three,color=MLCooling]%
}

\newcommand{\MethaneLindeDrawStageTwo}{%
  \MethaneLindeProcess[type=isentrope,
    from={pressure=5bar,enthalpy=866kJkg},to={pressure=25bar},
    name=ml-three-four,color=MLCompression,log coordinates=true]%
  \MethaneLindeProcess[type=isobar,pressure=25bar,
    from={entropy=\MethaneLindeCTwoToEntropySI JkgK},
    to={enthalpy=840kJkg},name=ml-four-five,color=MLCooling]%
}

\newcommand{\MethaneLindeDrawStageThree}{%
  \MethaneLindeProcess[type=isentrope,
    from={pressure=25bar,enthalpy=840kJkg},to={pressure=100bar},
    name=ml-five-six,color=MLCompression,log coordinates=true]%
  \MethaneLindeProcess[type=isobar,pressure=100bar,
    from={entropy=\MethaneLindeCThreeToEntropySI JkgK},
    to={temperature=-63C},name=ml-six-seven,color=MLCooling]%
}

\newcommand{\MethaneLindeMark}[3]{%
  \node[methane linde state,#3] at (#1) {#2};}

\newcommand{\MethaneLindeMarkAnchors}{%
  \MethaneLindeMark{MLZero}{0}{anchor=north west}%
```

```
  \MethaneLindeMark{MLSeven}{7}{anchor=south west}%
  \MethaneLindeMark{MLTen}{10}{anchor=north east}%
}

\newcommand{\MethaneLindeMarkSeparation}{%
  \MethaneLindeMark{MLSevenBis}{$7'$}{anchor=south}%
  \MethaneLindeMark{MLEight}{8}{anchor=north}%
  \MethaneLindeMark{MLNine}{9}{anchor=north west}%
  \MethaneLindeMark{MLOneBis}{$1'$}{anchor=north}%
}

\newcommand{\MethaneLindeMarkStageOne}{%
  \MethaneLindeMark{MLOne}{1}{anchor=north}%
  \MethaneLindeMark{MLTwo}{2}{anchor=south east}%
  \MethaneLindeMark{MLThree}{3}{anchor=north west}%
}

\newcommand{\MethaneLindeMarkStageTwo}{%
  \MethaneLindeMark{MLFour}{4}{anchor=south west}%
  \MethaneLindeMark{MLFive}{5}{anchor=north west}%
}

\newcommand{\MethaneLindeMarkStageThree}{%
  \MethaneLindeMark{MLSix}{6}{anchor=south west}%
}

\newcommand{\MethaneLindeNumericPanel}{%
  \node[methane linde note,anchor=north west]
    at (rel axis cs:.012,.985) {%
    $x_8=\num[round-mode=places,round-precision=4]
      {\MethaneLindeQualityEight}$,\quad
    $\dot m_1=\qty[round-mode=places,round-precision=3]
      {\MethaneLindeMassFlowOne}{\kilogram\per\second}$\\
    $h_{1'}=\qty[round-mode=places,round-precision=1]
      {\fpeval{\MethaneLindeEnthalpyOneBisSI/1000}}
      {\kilo\joule\per\kilogram}$,\quad
    $h_1=\qty[round-mode=places,round-precision=1]
      {\fpeval{\MethaneLindeEnthalpyOneSI/1000}}
      {\kilo\joule\per\kilogram}$\\
    $\dot Q_{E1},\dot Q_{E2},\dot Q_{E3}
      =\qty[round-mode=places,round-precision=0]
        {\MethaneLindeThermalPowerOneKW}{\kilo\watt},
       \qty[round-mode=places,round-precision=0]
        {\MethaneLindeThermalPowerTwoKW}{\kilo\watt},
       \qty[round-mode=places,round-precision=0]
        {\MethaneLindeThermalPowerThreeKW}{\kilo\watt}$\\
    $\dot W_{C1}=\qty[round-mode=places,round-precision=0]
      {\MethaneLindeCompressorOnePowerKW}{\kilo\watt}$,\quad
    $e_T=\num[round-mode=places,round-precision=3]
      {\MethaneLindeCoolingEffectiveness}$
  };%
}

\newcommand{\MethaneLindeLogResults}{%
  \typeout{Methane Linde x8: \MethaneLindeQualityEight}%
  \typeout{Methane Linde mass flow 1 (kg/s): \MethaneLindeMassFlowOne}%
  \typeout{Methane Linde mass flow 1bis (kg/s): \MethaneLindeMassFlowOneBis}%
  \typeout{Methane Linde h1bis (J/kg): \MethaneLindeEnthalpyOneBisSI}%
  \typeout{Methane Linde h1 (J/kg): \MethaneLindeEnthalpyOneSI}%
  \typeout{Methane Linde QE1 (kW): \MethaneLindeThermalPowerOneKW}%
  \typeout{Methane Linde QE2 (kW): \MethaneLindeThermalPowerTwoKW}%
  \typeout{Methane Linde QE3 (kW): \MethaneLindeThermalPowerThreeKW}%
  \typeout{Methane Linde WC1 (kW): \MethaneLindeCompressorOnePowerKW}%
  \typeout{Methane Linde ideal compressor total (kW):
```

```
    \MethaneLindeIdealCompressionPowerKW}%
  \typeout{Methane Linde cooling effectiveness:
    \MethaneLindeCoolingEffectiveness}%
}

\newcommand{\MethaneLindeStageTitle}[1]{%
  \def\MethaneLindeTitle{Blank methane PH worksheet}%
  \ifcase#1\relax
  \or\def\MethaneLindeTitle{Step 1: locate the three directly specified states}%
  \or\def\MethaneLindeTitle{Step 2: regenerator, throttle, and phase separator}%
  \or\def\MethaneLindeTitle{Step 3: close the recirculation balance and draw the first stage}%
  \or\def\MethaneLindeTitle{Step 4: add the second compression and intercooler}%
  \or\def\MethaneLindeTitle{Step 5: complete the high-pressure train}%
  \or\def\MethaneLindeTitle{Step 6: complete Linde process and live numerical audit}%
  \fi
}

\newcommand{\MethaneLindeDiagram}[1]{%
  \MethaneLindeStageTitle{#1}%

  \begin{page}
  \begin{tikzpicture}
  \begin{axis}[
    lcp fluid=Methane,width=24cm,height=15.8cm,
    xmin=-100,xmax=1200,ymin=.2,ymax=200,ymode=log,
    xlabel={Specific enthalpy $h$ (\si{\kilo\joule\per\kilogram})},
    ylabel={Pressure $p$ (bar)},title={\MethaneLindeTitle},
    grid=both,major grid style={black!28},minor grid style={black!11},
    tick label style={font=\small},label style={font=\small},
    title style={font=\sffamily\large},clip mode=individual,
    auto node placement,auto node algorithm=repair,
    auto node candidates=19,auto node bbox mode=oriented,
    auto node failure mode=hide-low-priority
  ]
    \ifnum#1=0\relax
      \MethaneLindeDenseBackground
    \else
      \ifnum#1=6\relax
        \MethaneLindeDenseBackground
      \else
        \MethaneLindeSparseBackground
      \fi
      \MethaneLindeCompute
      \MethaneLindeCoordinates

      \addplot[MLGuide,densely dashed,line width=.55pt,forget plot]
        coordinates {(-100,1) (1200,1)};
      \addplot[MLGuide,densely dashed,line width=.55pt,forget plot]
        coordinates {(-100,5) (1200,5)};
      \addplot[MLGuide,densely dashed,line width=.55pt,forget plot]
        coordinates {(-100,25) (1200,25)};
      \addplot[MLGuide,densely dashed,line width=.55pt,forget plot]
        coordinates {(-100,100) (1200,100)};

      \MethaneLindeMarkAnchors
      \ifnum#1>1\relax
        \MethaneLindeDrawSeparation
        \MethaneLindeMarkSeparation
      \fi
      \ifnum#1>2\relax
        \MethaneLindeDrawMixer
        \MethaneLindeDrawStageOne
        \MethaneLindeMarkStageOne
      \fi
```

```
      \ifnum#1>3\relax
        \MethaneLindeDrawStageTwo
        \MethaneLindeMarkStageTwo
      \fi
      \ifnum#1>4\relax
        \MethaneLindeDrawStageThree
        \MethaneLindeMarkStageThree
      \fi
      \ifnum#1=6\relax
        \MethaneLindeNumericPanel
        \MethaneLindeLogResults
      \fi
    \fi
  \end{axis}
  \end{tikzpicture}
  \end{page}
  %
}
```

Command Index

`axis` environment, 78

Environments
`axis`, 78

`\LCPAddCycle`, 176
`\LCPAddDiagramFamily`, 97
`\LCPAddDiagramPlots`, 96
`\LCPAddDiagramProcess`, 97
`\LCPAddHSIsobars`, 152
`\LCPAddHSIsochores`, 151
`\LCPAddHSIsoQuality`, 153
`\LCPAddHSIsotherms`, 152
`\LCPAddHSPlots`, 150
`\LCPAddHSProcess`, 153
`\LCPAddHSQuality`, 151
`\LCPAddModelProcess`, 176
`\LCPAddPHIsentropes`, 101
`\LCPAddPhIsentropes` (alias), 103
`\LCPAddPHIsochores`, 102
`\LCPAddPhIsochores` (alias), 103
`\LCPAddPHIsoQuality` (alias), 103
`\LCPAddPhIsoQuality` (alias), 103
`\LCPAddPHIsotherms`, 101
`\LCPAddPhIsotherms` (alias), 103
`\LCPAddPHPlots`, 100
`\LCPAddPhPlots` (alias), 103
`\LCPAddPHProcess`, 102
`\LCPAddPhProcess` (alias), 103
`\LCPAddPHQuality`, 100
`\LCPAddPTIsenthalps`, 165
`\LCPAddPTIsentropes`, 164
`\LCPAddPTIsochores`, 165
`\LCPAddPTPhaseEnvelope`, 164
`\LCPAddPTPlots`, 163
`\LCPAddPTProcess`, 166
`\LCPAddPTSaturation` (alias), 164
`\LCPAddPVIsentropes`, 134
`\LCPAddPVIsotherms`, 134
`\LCPAddPVPlots`, 133
`\LCPAddPVProcess`, 134
`\LCPAddPVQuality`, 133
`\LCPAddTSIsenthalps`, 141
`\LCPAddTSIsoQuality`, 142
`\LCPAddTSPlots`, 140
`\LCPAddTSProcess`, 142
`\LCPAddTSQuality`, 141
`\LCPBalanceSetup`, 183
`\LCPComponentValue`, 183
`\LCPCycleValue`, 183
`\LCPDeclareFluid`, 88
`\LCPDeclareModel`, 176
`\LCPDefineComponent`, 183
`\LCPDefineCycle`, 176
`\LCPDefineProcess`, 176
`\LCPDefineState`, 176
`\LCPDiagram`, 96
`\LCPExportBalance`, 184
`\LCPExportCycleTable`, 177
`\LCPExportDiagramTable`, 94
`\LCPExportProcessTable`, 94
`\LCPExportStates`, 177
`\LCPFluidName`, 88
`\LCPFluidPcritBar`, 88
`\LCPFluidPcritPlot`, 88
`\LCPFluidPcritSI`, 88
`\LCPFluidPtripleBar`, 88
`\LCPFluidPtripleSI`, 88
`\LCPFluidReferenceState`, 88
`\LCPFluidRhocritSI`, 88
`\LCPFluidTcritC`, 88
`\LCPFluidTcritK`, 88
`\LCPFluidTtripleC`, 88
`\LCPFluidTtripleK`, 88
`\LCPHSDiagram`, 150
`\LCPLogBalance`, 184
`\LCPLogState`, 177
`\LCPPHDiagram`, 100
`\LCPPhDiagram` (alias), 102
`\LCPPhIsoQualityDiagram` (alias), 103
`\LCPPTDiagram`, 163
`\LCPPVDiagram`, 132
`\LCPSetFluid`, 87
`\LCPSetLibrary`, 87
`\LCPSetReferenceState`, 88
`\LCPSetup`, 87
`\LCPStateCoordinate`, 177
`\LCPStateValue`, 177
`\LCPTSDiagram`, 140
`\LCPUpdateFluidConstants`, 88

Key and Value Index

automatic labels (alias), 119
autonode candidate strategy=around-preferred, 122
autonode candidate strategy=uniform, 122
autonode labels (alias), 120

boundary color (alias), 115

cache key, 89, 179
cache directory key, 89, 179
color key, 178
common/autonode candidate strategy key, 122
common/autonode candidates key, 121
common/autonode clearance key, 122
common/autonode inner sep key, 122
common/autonode normal shift key, 122
common/autonode overlap weight key, 122
common/autonode preferred weight key, 122
common/axis options key, 105
common/boundary style key, 114
common/cache key, 91
common/cache directory key, 91
common/color key, 128
common/coord digits key, 112, 132
common/curve style key, 113
common/curve style for key, 121
common/domain policy key, 99, 125
common/enthalpy key, 127
common/enthalpy scale key, 104, 131
common/enthalpy symbol key, 105
common/entropy key, 127
common/entropy count key, 109
common/entropy max key, 109
common/entropy min key, 108
common/entropy mode key, 109
common/entropy preset key, 109
common/entropy step key, 109
common/entropy symbol key, 115
common/entropy unit key, 108
common/entropy values key, 109
common/export coordinates key, 130
common/fluid key, 103, 126
common/forget plot key, 114
common/from key, 126
common/initial intervals key, 111, 131
common/interior style key, 114
common/isentrope key, 104
common/isentrope color key, 114
common/isentrope initial intervals key, 112
common/isentrope label allow upside down key, 118
common/isentrope label every key, 117
common/isentrope label node style key, 119
common/isentrope label pos key, 116
common/isentrope label sloped key, 118
common/isentrope label style key, 119
common/isentrope labels key, 116
common/isentrope max depth key, 112
common/isentrope style key, 114
common/isentrope tolerance key, 112
common/isochore key, 104
common/isochore color key, 114
common/isochore initial intervals key, 113
common/isochore label allow upside down key, 118
common/isochore label every key, 117
common/isochore label node style key, 119
common/isochore label pos key, 117
common/isochore label sloped key, 118
common/isochore label style key, 119
common/isochore labels key, 116
common/isochore max depth key, 113
common/isochore style key, 114
common/isochore tolerance key, 113
common/isotherm key, 104
common/isotherm color key, 113
common/isotherm initial intervals key, 112
common/isotherm label allow upside down key, 118
common/isotherm label every key, 117
common/isotherm label node style key, 119
common/isotherm label pos key, 116
common/isotherm label sloped key, 117
common/isotherm label style key, 119
common/isotherm labels key, 116
common/isotherm max depth key, 112
common/isotherm style key, 114
common/isotherm tolerance key, 112
common/label key, 129
common/label allow upside down key, 118, 129
common/label allow upside down for key, 121
common/label box base width key, 123
common/label box char width key, 124
common/label box height key, 124
common/label box sloped multiplier key, 124
common/label candidate count key, 123
common/label collision policy key, 123
common/label distance penalty key, 123
common/label edge penalty key, 123
common/label every key, 117
common/label fixed for key, 121
common/label hide edge threshold key, 123
common/label hide overlap threshold key, 123
common/label max pos key, 122
common/label min pos key, 122
common/label node style key, 119, 129
common/label overlap penalty key, 123
common/label placement key, 116

common/label pos key, 116, 129
common/label pos for key, 120
common/label separation key, 123
common/label show for key, 121
common/label sloped key, 117, 129
common/label sloped for key, 121
common/label style key, 118, 129
common/label style for key, 120
common/label text for key, 120
common/labels key, 115
common/legend key, 114
common/library key, 103, 126
common/log coordinates key, 130
common/log weight key, 112, 132
common/mark endpoints key, 129
common/marker style key, 129
common/max depth key, 111, 132
common/name key, 128
common/pressure key, 127
common/pressure max key, 104
common/pressure max factor key, 104
common/pressure min key, 104
common/pressure scale key, 104, 131
common/pressure symbol key, 105
common/quality key, 104, 127
common/quality boundary color key, 113
common/quality color key, 113
common/quality count key, 106
common/quality label allow upside down key, 118
common/quality label every key, 117
common/quality label node style key, 119
common/quality label pos key, 116
common/quality label sloped key, 117
common/quality label style key, 118
common/quality labels key, 116
common/quality max key, 106
common/quality min key, 106
common/quality mode key, 106
common/quality preset key, 106
common/quality step key, 106
common/quality symbol key, 115
common/quality values key, 106
common/reference state key, 99, 124
common/saturation pressure epsilon key, 113
common/specific volume key, 127
common/specific volume count key, 110
common/specific volume max key, 110
common/specific volume min key, 110
common/specific volume mode key, 110
common/specific volume preset key, 110
common/specific volume step key, 110
common/specific volume symbol key, 115
common/specific volume unit key, 109
common/specific volume values key, 110
common/style key, 128
common/temperature key, 127
common/temperature count key, 108
common/temperature max key, 107
common/temperature min key, 107
common/temperature mode key, 108
common/temperature preset key, 108
common/temperature step key, 108
common/temperature symbol key, 115
common/temperature unit key, 107
common/temperature values key, 108
common/to key, 126
common/tolerance key, 112, 132
common/type key, 126
common/value key, 126
coord digits key, 179

discontinuity policy (alias), 99, 125
domain policy key, 179

efficiency key, 185
enthalpy axis unit key, 89
enthalpy curves (alias), 144, 167
enthalpy scale key, 90
entropy axis unit key, 90
entropy mode=auto, 109
entropy mode=explicit, 109
entropy mode=linear, 109
entropy mode=list, 109
entropy mode=log, 109
entropy mode=logarithmic, 109
entropy mode=predefined, 109
entropy mode=preset, 109
entropy preset=default, 109
entropy preset=dense, 109
entropy preset=refrigeration, 109
entropy preset=sparse, 109
entropy scale key, 90
entropy unit=j/kg/k, 108
entropy unit=jkgk, 108
entropy unit=kjkgk, 108
entropy unit=si, 108
equilibrium curve (alias), 167

fluid key, 89, 177
from key, 178, 184

h (alias), 128
h count (alias), 145, 170
h max (alias), 145, 170
h min (alias), 145, 170
h mode (alias), 145, 170
h scale (alias), 90, 105, 132
h step (alias), 145, 170
h unit (alias), 144, 169
h values (alias), 145, 170
heat key, 185
HS/enthalpy axis max key, 161
HS/enthalpy axis min key, 160
HS/enthalpy axis unit key, 125, 156
HS/enthalpy weight key, 156, 161
HS/entropy axis max key, 160
HS/entropy axis min key, 160

HS/entropy axis unit key, 125, 156
HS/entropy scale key, 155, 161
HS/isobar key, 155
HS/isobar color key, 159
HS/isobar count key, 157
HS/isobar initial intervals key, 160
HS/isobar label allow upside down key, 159
HS/isobar label every key, 158
HS/isobar label node style key, 159
HS/isobar label pos key, 158
HS/isobar label sloped key, 158
HS/isobar label style key, 159
HS/isobar labels key, 158
HS/isobar max key, 157
HS/isobar max depth key, 160
HS/isobar min key, 157
HS/isobar mode key, 157
HS/isobar step key, 157
HS/isobar style key, 159
HS/isobar temperature max key, 158
HS/isobar temperature min key, 157
HS/isobar tolerance key, 160
HS/isobar unit key, 157
HS/isobar values key, 157

id (alias), 129
initial intervals key, 179
isenthalps (alias), 144, 167
isentropes (alias), 105, 167
iso quality (alias), 105
isobars (alias), 155
isochores (alias), 105, 167
isoquality (alias), 105
isotherms (alias), 105

kind (alias), 128

label collision (alias), 123
label text (alias), 129
lcp fluid (alias), 175
library key, 89, 177
/luacoolprop/
    balance/
      mass flow, 185
    cache, 89
    cache directory, 89
    component/
      efficiency, 185
      from, 184
      heat, 185
      pressure drop, 185
      pressure ratio, 185
      thermal model, 185
      to, 184
      to state, 184
      type, 184
    diagram/
      common/autonode candidate strategy, 122
      common/autonode candidates, 121
      common/autonode clearance, 122
      common/autonode inner sep, 122
      common/autonode normal shift, 122
      common/autonode overlap weight, 122
      common/autonode preferred weight, 122
      common/axis options, 105
      common/boundary style, 114
      common/cache, 91
      common/cache directory, 91
      common/coord digits, 112
      common/curve style, 113
      common/curve style for, 121
      common/domain policy, 99
      common/enthalpy scale, 104
      common/enthalpy symbol, 105
      common/entropy count, 109
      common/entropy max, 109
      common/entropy min, 108
      common/entropy mode, 109
      common/entropy preset, 109
      common/entropy step, 109
      common/entropy symbol, 115
      common/entropy unit, 108
      common/entropy values, 109
      common/fluid, 103
      common/forget plot, 114
      common/initial intervals, 111
      common/interior style, 114
      common/isentrope, 104
      common/isentrope color, 114
      common/isentrope initial intervals, 112
      common/isentrope label allow upside down, 118
      common/isentrope label every, 117
      common/isentrope label node style, 119
      common/isentrope label pos, 116
      common/isentrope label sloped, 118
      common/isentrope label style, 119
      common/isentrope labels, 116
      common/isentrope max depth, 112
      common/isentrope style, 114
      common/isentrope tolerance, 112
      common/isochore, 104
      common/isochore color, 114
      common/isochore initial intervals, 113
      common/isochore label allow upside down, 118
      common/isochore label every, 117
      common/isochore label node style, 119
      common/isochore label pos, 117
      common/isochore label sloped, 118
      common/isochore label style, 119
      common/isochore labels, 116

common/isochore max depth, 113
common/isochore style, 114
common/isochore tolerance, 113
common/isotherm, 104
common/isotherm color, 113
common/isotherm initial intervals, 112
common/isotherm label allow upside down, 118
common/isotherm label every, 117
common/isotherm label node style, 119
common/isotherm label pos, 116
common/isotherm label sloped, 117
common/isotherm label style, 119
common/isotherm labels, 116
common/isotherm max depth, 112
common/isotherm style, 114
common/isotherm tolerance, 112
common/label allow upside down, 118
common/label allow upside down for, 121
common/label box base width, 123
common/label box char width, 124
common/label box height, 124
common/label box sloped multiplier, 124
common/label candidate count, 123
common/label collision policy, 123
common/label distance penalty, 123
common/label edge penalty, 123
common/label every, 117
common/label fixed for, 121
common/label hide edge threshold, 123
common/label hide overlap threshold, 123
common/label max pos, 122
common/label min pos, 122
common/label node style, 119
common/label overlap penalty, 123
common/label placement, 116
common/label pos, 116
common/label pos for, 120
common/label separation, 123
common/label show for, 121
common/label sloped, 117
common/label sloped for, 121
common/label style, 118
common/label style for, 120
common/label text for, 120
common/labels, 115
common/legend, 114
common/library, 103
common/log weight, 112
common/max depth, 111
common/pressure max, 104
common/pressure max factor, 104
common/pressure min, 104
common/pressure scale, 104
common/pressure symbol, 105
common/quality, 104
common/quality boundary color, 113
common/quality color, 113
common/quality count, 106
common/quality label allow upside down, 118
common/quality label every, 117
common/quality label node style, 119
common/quality label pos, 116
common/quality label sloped, 117
common/quality label style, 118
common/quality labels, 116
common/quality max, 106
common/quality min, 106
common/quality mode, 106
common/quality preset, 106
common/quality step, 106
common/quality symbol, 115
common/quality values, 106
common/reference state, 99
common/saturation pressure epsilon, 113
common/specific volume count, 110
common/specific volume max, 110
common/specific volume min, 110
common/specific volume mode, 110
common/specific volume preset, 110
common/specific volume step, 110
common/specific volume symbol, 115
common/specific volume unit, 109
common/specific volume values, 110
common/temperature count, 108
common/temperature max, 107
common/temperature min, 107
common/temperature mode, 108
common/temperature preset, 108
common/temperature step, 108
common/temperature symbol, 115
common/temperature unit, 107
common/temperature values, 108
common/tolerance, 112
HS/enthalpy axis max, 161
HS/enthalpy axis min, 160
HS/enthalpy axis unit, 156
HS/enthalpy weight, 156
HS/entropy axis max, 160
HS/entropy axis min, 160
HS/entropy axis unit, 156
HS/entropy scale, 155
HS/isobar, 155
HS/isobar color, 159
HS/isobar count, 157
HS/isobar initial intervals, 160
HS/isobar label allow upside down, 159
HS/isobar label every, 158
HS/isobar label node style, 159

HS/isobar label pos, 158
HS/isobar label sloped, 158
HS/isobar label style, 159
HS/isobar labels, 158
HS/isobar max, 157
HS/isobar max depth, 160
HS/isobar min, 157
HS/isobar mode, 157
HS/isobar step, 157
HS/isobar style, 159
HS/isobar temperature max, 158
HS/isobar temperature min, 157
HS/isobar tolerance, 160
HS/isobar unit, 157
HS/isobar values, 157
PH/enthalpy axis unit, 105
PH/pressure axis unit, 105
PT/domain policy, 173
PT/enthalpy count, 170
PT/enthalpy max, 169
PT/enthalpy min, 169
PT/enthalpy mode, 170
PT/enthalpy step, 169
PT/enthalpy symbol, 169
PT/enthalpy unit, 169
PT/enthalpy values, 170
PT/entropy symbol, 169
PT/fluid, 166
PT/initial intervals, 173
PT/isenthalp, 167
PT/isenthalp color, 171
PT/isenthalp initial intervals, 173
PT/isenthalp label allow upside down, 170
PT/isenthalp label every, 170
PT/isenthalp label node style, 171
PT/isenthalp label pos, 170
PT/isenthalp label sloped, 170
PT/isenthalp label style, 171
PT/isenthalp labels, 170
PT/isenthalp max depth, 173
PT/isenthalp style, 171
PT/isenthalp tolerance, 173
PT/isentrope, 167
PT/isochore, 167
PT/library, 166
PT/log weight, 173
PT/max depth, 173
PT/phase envelope, 167
PT/phase envelope color, 172
PT/phase envelope initial intervals, 172
PT/phase envelope label allow upside down, 171
PT/phase envelope label node style, 172
PT/phase envelope label pos, 171
PT/phase envelope label sloped, 171
PT/phase envelope label style, 172
PT/phase envelope label text, 171
PT/phase envelope labels, 171
PT/phase envelope max depth, 172
PT/phase envelope style, 172
PT/phase envelope tolerance, 172
PT/pressure axis unit, 168
PT/pressure max, 167
PT/pressure max factor, 168
PT/pressure min, 167
PT/pressure scale, 168
PT/pressure symbol, 169
PT/reference state, 167
PT/specific volume symbol, 169
PT/temperature axis max, 168
PT/temperature axis min, 168
PT/temperature axis unit, 168
PT/temperature scale, 168
PT/temperature symbol, 168
PT/tolerance, 173
PV/log x weight, 136
PV/pressure axis unit, 136
PV/specific volume axis unit, 136
PV/specific volume scale, 136
TS/enthalpy count, 145
TS/enthalpy max, 145
TS/enthalpy min, 145
TS/enthalpy mode, 145
TS/enthalpy step, 145
TS/enthalpy unit, 144
TS/enthalpy values, 145
TS/entropy axis max, 147
TS/entropy axis min, 147
TS/entropy axis unit, 144
TS/entropy scale, 144
TS/isenthalp, 144
TS/isenthalp color, 146
TS/isenthalp initial intervals, 147
TS/isenthalp label allow upside down, 146
TS/isenthalp label every, 146
TS/isenthalp label node style, 146
TS/isenthalp label pos, 146
TS/isenthalp label sloped, 146
TS/isenthalp label style, 146
TS/isenthalp labels, 145
TS/isenthalp max depth, 147
TS/isenthalp style, 146
TS/isenthalp tolerance, 147
TS/temperature axis max, 147
TS/temperature axis min, 147
TS/temperature axis unit, 144
TS/temperature scale, 144
TS/temperature weight, 147
enthalpy axis unit, 89
enthalpy scale, 90
entropy axis unit, 90
entropy scale, 90
fluid, 89
library, 89

model/
  fluid, 177
  library, 177
  reference state, 178
model process/
  from, 178
  to, 178
  to state, 178
  type, 178
model view/
  cache, 179
  cache directory, 179
  color, 178
  coord digits, 179
  domain policy, 179
  initial intervals, 179
  mark endpoints, 178
  marker style, 179
  max depth, 179
  style, 178
  tolerance, 179
pressure axis unit, 89
pressure scale, 90
process/
  common/cache, 91
  common/cache directory, 91
  common/color, 128
  common/coord digits, 132
  common/domain policy, 125
  common/enthalpy, 127
  common/enthalpy scale, 131
  common/entropy, 127
  common/export coordinates, 130
  common/fluid, 126
  common/from, 126
  common/initial intervals, 131
  common/label, 129
  common/label allow upside down, 129
  common/label node style, 129
  common/label pos, 129
  common/label sloped, 129
  common/label style, 129
  common/library, 126
  common/log coordinates, 130
  common/log weight, 132
  common/mark endpoints, 129
  common/marker style, 129
  common/max depth, 132
  common/name, 128
  common/pressure, 127
  common/pressure scale, 131
  common/quality, 127
  common/reference state, 124
  common/specific volume, 127
  common/style, 128
  common/temperature, 127
  common/to, 126
  common/tolerance, 132
  common/type, 126
  common/value, 126
  HS/enthalpy axis unit, 125
  HS/enthalpy weight, 161
  HS/entropy axis unit, 125
  HS/entropy scale, 161
  PH/enthalpy axis unit, 125
  PH/pressure axis unit, 125
  PT/pressure axis unit, 174
  PT/temperature axis unit, 174
  PT/temperature scale, 174
  PV/log x weight, 138
  PV/pressure axis unit, 125
  PV/specific volume axis unit, 125
  PV/specific volume scale, 138
  TS/entropy axis unit, 125
  TS/entropy scale, 148
  TS/temperature axis unit, 125
  TS/temperature scale, 148
  TS/temperature weight, 148
reference state, 89
specific volume axis unit, 89
specific volume scale, 90
temperature axis unit, 90
temperature scale, 90
luacoolprop fluid key, 175

mark endpoints key, 178
marker style key, 179
markers (alias), 129
mass flow key, 185
max depth key, 179

p (alias), 128
p max (alias), 105
p max factor (alias), 105
p min (alias), 105
p scale (alias), 90, 105, 132, 168
/pgfplots/
    luacoolprop fluid, 175
pgfplots autonode labels (alias), 120
PH/enthalpy axis unit key, 105, 125
PH/pressure axis unit key, 105, 125
pressure axis unit key, 89
pressure drop key, 185
pressure ratio key, 185
pressure scale key, 90
process (alias), 128
process color (alias), 129
process style (alias), 129
PT/domain policy key, 173
PT/enthalpy count key, 170
PT/enthalpy max key, 169
PT/enthalpy min key, 169
PT/enthalpy mode key, 170
PT/enthalpy step key, 169
PT/enthalpy symbol key, 169
PT/enthalpy unit key, 169
PT/enthalpy values key, 170
PT/entropy symbol key, 169

PT/fluid key, 166
PT/initial intervals key, 173
PT/isenthalp key, 167
PT/isenthalp color key, 171
PT/isenthalp initial intervals key, 173
PT/isenthalp label allow upside down key, 170
PT/isenthalp label every key, 170
PT/isenthalp label node style key, 171
PT/isenthalp label pos key, 170
PT/isenthalp label sloped key, 170
PT/isenthalp label style key, 171
PT/isenthalp labels key, 170
PT/isenthalp max depth key, 173
PT/isenthalp style key, 171
PT/isenthalp tolerance key, 173
PT/isentrope key, 167
PT/isochore key, 167
PT/library key, 166
PT/log weight key, 173
PT/max depth key, 173
PT/phase envelope key, 167
PT/phase envelope color key, 172
PT/phase envelope initial intervals key, 172
PT/phase envelope label allow upside down key, 171
PT/phase envelope label node style key, 172
PT/phase envelope label pos key, 171
PT/phase envelope label sloped key, 171
PT/phase envelope label style key, 172
PT/phase envelope label text key, 171
PT/phase envelope labels key, 171
PT/phase envelope max depth key, 172
PT/phase envelope style key, 172
PT/phase envelope tolerance key, 172
PT/pressure axis unit key, 168, 174
PT/pressure max key, 167
PT/pressure max factor key, 168
PT/pressure min key, 167
PT/pressure scale key, 168
PT/pressure symbol key, 169
PT/reference state key, 167
PT/specific volume symbol key, 169
PT/temperature axis max key, 168
PT/temperature axis min key, 168
PT/temperature axis unit key, 168, 174
PT/temperature scale key, 168, 174
PT/temperature symbol key, 168
PT/tolerance key, 173
PV/log x weight key, 136, 138
PV/pressure axis unit key, 125, 136
PV/specific volume axis unit key, 125, 136
PV/specific volume scale key, 136, 138

Q (alias), 128
q (alias), 128
q count (alias), 107
q max (alias), 107
q min (alias), 107
q mode (alias), 107
q preset (alias), 107
q step (alias), 107
q values (alias), 107
quality curves (alias), 105
quality mode=auto, 106
quality mode=explicit, 106
quality mode=linear, 106
quality mode=list, 106
quality mode=log, 106
quality mode=logarithmic, 106
quality mode=predefined, 106
quality mode=preset, 106
quality preset=boundaries, 107
quality preset=boundary, 107
quality preset=default, 106
quality preset=dense, 107
quality preset=sparse, 107

reference state key, 89, 178

s (alias), 128
s count (alias), 109
s max (alias), 109
s min (alias), 109
s mode (alias), 109
s preset (alias), 109
s scale (alias), 90, 144, 148, 156, 161
s step (alias), 109
s values (alias), 109
saturation (alias), 167
sloped label (alias), 129
sloped labels (alias), 119
smart labels (alias), 119
specific volume axis unit key, 89
specific volume mode=auto, 110
specific volume mode=explicit, 110
specific volume mode=linear, 110
specific volume mode=list, 110
specific volume mode=log, 110
specific volume mode=logarithmic, 110
specific volume mode=predefined, 110
specific volume mode=preset, 110
specific volume preset=default, 110
specific volume preset=dense, 110
specific volume preset=refrigeration, 110
specific volume preset=sparse, 110
specific volume scale key, 90
specific volume unit=dm3/kg, 109
specific volume unit=dm3kg, 109
specific volume unit=l/kg, 109
specific volume unit=lkg, 109
specific volume unit=m3kg, 109
start (alias), 128
stop (alias), 128
style key, 178

T (alias), 128

t (alias), 128
t count (alias), 108
t max (alias), 108
t min (alias), 108
t mode (alias), 108
t preset (alias), 108
t scale (alias), 90, 144, 148, 168, 174
t step (alias), 108
t values (alias), 108
temperature axis unit key, 90
temperature mode=auto, 108
temperature mode=explicit, 108
temperature mode=linear, 108
temperature mode=list, 108
temperature mode=log, 108
temperature mode=logarithmic, 108
temperature mode=predefined, 108
temperature mode=preset, 108
temperature preset=default, 108
temperature preset=dense, 108
temperature preset=refrigeration, 108
temperature preset=sparse, 108
temperature scale key, 90
temperature unit=c, 107
temperature unit=celsius, 107
temperature unit=degc, 107
temperature unit=k, 107
temperature unit=kelvin, 107
thermal model key, 185
to key, 178, 184
to state key, 178, 184
tolerance key, 179
TS/enthalpy count key, 145
TS/enthalpy max key, 145
TS/enthalpy min key, 145
TS/enthalpy mode key, 145
TS/enthalpy step key, 145
TS/enthalpy unit key, 144
TS/enthalpy values key, 145
TS/entropy axis max key, 147
TS/entropy axis min key, 147
TS/entropy axis unit key, 125, 144
TS/entropy scale key, 144, 148
TS/isenthalp key, 144
TS/isenthalp color key, 146
TS/isenthalp initial intervals key, 147
TS/isenthalp label allow upside down key, 146
TS/isenthalp label every key, 146
TS/isenthalp label node style key, 146
TS/isenthalp label pos key, 146
TS/isenthalp label sloped key, 146
TS/isenthalp label style key, 146
TS/isenthalp labels key, 145
TS/isenthalp max depth key, 147
TS/isenthalp style key, 146
TS/isenthalp tolerance key, 147
TS/temperature axis max key, 147
TS/temperature axis min key, 147
TS/temperature axis unit key, 125, 144
TS/temperature scale key, 144, 148
TS/temperature weight key, 147, 148
type key, 178, 184
type=enthalpy, 126
type=entropy, 126
type=h, 126
type=isenthalp, 126
type=isenthalpic, 126
type=isentrope, 126
type=isentropic, 126
type=isobar, 126
type=isobaric, 126
type=isochore, 126
type=isochoric, 126
type=isoquality, 126
type=isotherm, 126
type=isothermal, 126
type=p, 126
type=pressure, 126
type=q, 126
type=quality, 126
type=s, 126
type=t, 126
type=temperature, 126
type=throttle, 126
type=v, 126
type=volume, 126

v (alias), 128
v count (alias), 110
v max (alias), 110
v min (alias), 110
v mode (alias), 110
v preset (alias), 110
v scale (alias), 90, 136, 138
v step (alias), 110
v unit (alias), 110
v values (alias), 110
volume unit (alias), 110