

luacoolprop

LuaTeX access to CoolProp for PGFPlots thermodynamic diagrams

Christophe Jorssen
christophe.jorssen@gmail.com

Version 1.1.1 — 2026-09-20

Contents

I	Tutorials	6
1	A teaching pair for a real-fluid refrigeration machine	6
1.1	The exercise behind the diagram	6
1.2	One shared source and two thin drivers	6
1.3	Step 1: choose the worksheet's reading domain	7
1.4	Step 2: select purposeful isoline families	8
1.5	Result: the student worksheet diagram	9
1.6	Step 3: establish the five thermodynamic states	9
1.7	Step 4: draw each process from its invariant	10
1.8	Step 5: add state names and reading aids	11
1.9	Result: the model-solution diagram	13
1.10	Step 6: turn readings into an energy balance	13
1.11	Embedding the two PDFs in course material	14
1.12	Adapting the tutorial safely	14
1.13	Instructor's final checklist	15
2	Advanced tutorial: reconstructing an R134a air conditioner	16
2.1	Translate the statement into thermodynamic constraints	16
2.2	Use a stable plotting domain	16
2.3	Stage 1: identify the three phase regions	17
2.4	Stage 2: recognize the ideal-gas region	18
2.5	Stage 3: locate state 1 from two measured properties	19
2.6	Stage 4: construct the compressor outlet	20
2.7	Stage 5: cool at constant pressure to state 3	21
2.8	Stage 6: throttle, close the cycle, and locate state 4	22
2.9	Obtain every numerical value from process exports	23
2.10	Files, rebuild, and adaptation checklist	24
3	PV tutorial: butane in a domestic gas lighter	26
3.1	Translate the exercise into thermodynamic data	26
3.2	Keep one executable source for all five stages	26
3.3	Step 1: build a readable log-log worksheet	27
3.4	Step 2: read phase regions and the critical isotherm	29
3.5	Step 3: recognize the ideal-gas region geometrically	30
3.6	Step 4: locate the initial state with TikZ intersections	31
3.7	Step 5: draw the rigid-tank heating process	32

3.8	Rebuild and adapt the tutorial	34
4	Teaching pair: an R1234yf swimming-pool heat pump	35
4.1	Translate the statement into four idealized devices	35
4.2	Establish the steady-flow first law	35
4.3	Use one source and two thin drivers	36
4.4	Step 1: reproduce the blank PH chart	36
4.5	Step 2: derive both operating pressures	38
4.6	Step 3: compute the cycle in dependency order	39
4.7	Step 4: place states by intersections, not copied coordinates	40
4.8	Step 5: calculate and interpret the condenser power	41
4.9	Rebuild and adapt the teaching pair	42
5	Expert tutorial: methane liquefaction by the Linde process	43
5.1	Read the process as a directed network	43
5.2	Use a shared worksheet and progressive solution	43
5.3	Step 1: regenerate the blank methane PH chart	44
5.4	Step 2: calculate the primitive states	46
5.5	Step 3: throttle and separate the two-phase outlet	47
5.6	Step 4: close the separator mass balance	48
5.7	Step 5: close the two-stream regenerator balance	49
5.8	Step 6: close the mixer balance and reveal state 1	49
5.9	Step 7: construct the first compression and cooler	50
5.10	Step 8: repeat the invariant pattern for stages 2 and 3	51
5.11	Step 9: calculate powers with a declared sign convention	52
5.12	Step 10: read the complete process without inventing a cycle	53
5.13	Graphical estimates versus live calculations	54
5.14	Numerical API and reproducible build	55
II	User manual	56
6	Purpose and scope	56
6.1	How to cite CoolProp	56
7	Installation and runtime requirements	57
7.1	Install the TeX files	57
7.2	Security boundary and <code>-shell-escape</code>	57
7.3	Choose a compatible CoolProp binary	58
7.4	macOS: download or compile the <code>.dylib</code>	58
7.5	Linux: download or compile the <code>.so</code>	59
7.6	Windows: download or compile <code>CoolProp.dll</code>	60
7.7	Verify the runtime installation	60
7.8	LaTeX, plain TeX, and ConTeXt	61
7.9	LaTeX number and unit formatting with <code>siunitx</code>	62
7.10	Repository build workflow	63
8	Basic users: quick start	64
8.1	A fully automatic diagram	64
8.2	Another fluid	64
9	Diagram identifiers and stable API	65

10 Advanced users: manual PGFPlots axes	66
10.1 Curve-family macros	67
10.2 Value grids	67
11 Automatic labels with pgfplots-autonode	67
11.1 LuaCoolProp keys forwarded to autonode	69
12 Curve names and manual overrides	69
13 Thermodynamic process paths	70
 III Complete TeX reference	 71
14 How to read this reference	71
15 Loading in the three supported formats	71
16 Global configuration and fluid constants	72
16.1 Global keys: /luacoolprop	74
17 The generic diagram API	76
17.1 A symmetric capability contract	77
17.2 Pure-fluid and reference-state contracts	77
17.3 Disconnected thermodynamic domains	78
18 PH convenience commands	78
18.1 Alternate command names	80
19 PH diagram keys	81
19.1 Fluid, families, ranges, and axis	81
19.2 Quality grid	83
19.3 Temperature, entropy, and specific-volume grids	85
19.3.1 Temperature and isotherms	85
19.3.2 Specific entropy and isentropes	86
19.3.3 Specific volume and isochores	87
19.4 Adaptive sampling and numerical output	89
19.5 Curve styles, colours, and legend participation	91
19.6 Labels and family defaults	93
19.7 Per-curve overrides	98
19.8 Autonode options forwarded by PH diagrams	100
19.9 Alternate label-control keys	101
20 Shared process paths and the PH projection	102
20.1 Thermodynamic definition	103
20.2 Identity, appearance, and labels	106
20.3 Named intersections and numeric endpoint access	107
20.4 Process scaling and sampling	109
21 PV pressure–specific-volume diagrams	109
21.1 PV convenience commands	110
21.2 PV keys and defaults	112
21.3 PV curve names, intersections, and overrides	114
21.4 PV process keys and numerical exports	114

21.5 Plain TeX and ConTeXt	115
22 TS temperature–entropy diagrams	116
22.1 TS commands	116
22.2 Shared diagram vocabulary	118
22.3 TS family, scale, and grid keys	118
22.4 TS isenthalp appearance and labels	120
22.5 TS sampling and axis bounds	121
22.6 TS process keys and exported values	122
22.7 Plain TeX and ConTeXt	123
23 HS enthalpy–entropy diagrams	123
23.1 HS commands	124
23.2 Shared vocabulary and physical ranges	126
23.3 HS switches, scale, and error metric	127
23.4 Isobar grid and units	128
23.5 Isobar appearance and automatic labels	129
23.6 Isobar sampling and HS axis bounds	130
23.7 HS process keys and exported coordinates	131
23.8 Plain TeX and ConTeXt	132
24 PT pressure–temperature diagrams	132
24.1 PT commands	133
24.2 Families, ranges, and coordinate units	135
24.3 Enthalpy grid and isenthalp presentation	138
24.4 Phase-envelope presentation and sampling	140
24.5 Shared styling, adaptive sampling, and stable names	141
24.6 PT process projection and exports	143
24.7 Plain TeX and ConTeXt	143
25 PGFPlots fluid style	144
IV Developer reference	144
26 Architecture	144
27 PGFPlots lifecycle and label emission	144
28 Adaptive curve sampling	145
29 Thermodynamic state resolution	145
30 Naming conventions	145
31 Lua API: scope, contracts, and safe use	146
31.1 Loading the module and identifying both versions	146
31.2 SI-unit contract	146
31.3 Errors and recovery boundaries	147
31.4 Library metadata and direct property wrappers	148
31.5 Managed AbstractState objects	149
31.6 Fluid-constant table	149
31.7 Structured diagram interface	150
31.8 Point-record schema and adaptive sampling	150

31.9 Shared option contract and declared capabilities	152
31.10 Lua process API	153
31.11 The built-in registry and custom diagram types	154
31.12 TeX bridge and direct entry points	154
32 Curve records and the autonode bridge	155
33 Process validation	155

Part I

Tutorials

1 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.

1.1 The exercise behind the diagram

The refrigerant follows this idealized sequence:

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

1.2 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	Owens 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`.

1.3 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 kJ kg⁻¹.

Axis prepared for graphical reading

```
\LCPSsetup{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  $h$  ($\mathrm{kJ},\mathrm{kg}^{-1}$)},
  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.

1.4 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, ..., 1	Locate wet-vapor states and estimate Q_E .
isotherm	-60, -40, ..., 240 °C	Read state temperatures and inspect superheated-vapor behavior.
isentropes	1, 2, ..., 8 kJ kg ⁻¹ K ⁻¹	Construct the ideal compressor path and read entropy.
isochore	0.01, 0.02, ..., 20 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.

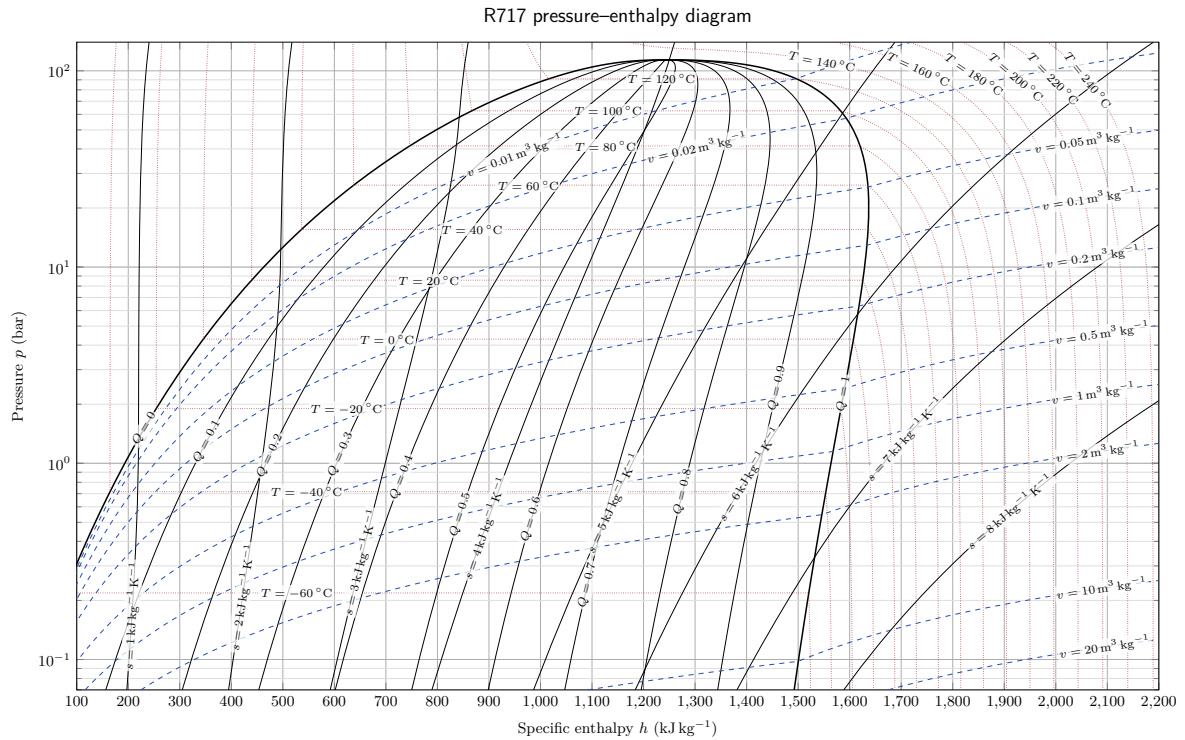
Temperature, entropy, and volume grids

```
\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},
  isentropes color=black,isentropes style={line width=.32pt},
  labels=true,isentropes label every=1,isentropes 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.

1.5 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.



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

Build only the student diagram

```
export LUACOOLOPROP_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.

1.6 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	$T/^\circ\text{C}$	Q	$v/\text{m}^3 \text{kg}^{-1}$	$h/\text{kJ kg}^{-1}$	$s/\text{kJ kg}^{-1} \text{K}^{-1}$
<i>A</i>	3	−9.22	1	0.40597	1596.90	6.22769
<i>B</i>	10	74.39	—	0.15947	1765.42	6.22769
<i>C</i>	10	24.91	1	0.12853	1628.94	5.80290
<i>D</i>	10	24.91	0	0.001658	462.76	1.89037
<i>E</i>	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 $h_D = h_E \simeq 460 \text{ kJ kg}^{-1}$. 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.

1.7 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 $A \rightarrow B$. 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=isentropo,
  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 $B \rightarrow C$. 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 $C \rightarrow D$. Both endpoints are saturation states on the same isobar.

Complete condensation

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

Isenthalpic expansion

```
\LCPAddPHPProcess[
  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 $E \rightarrow A$. 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

```
\LCPAddPHPProcess[
  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.

1.8 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};
```

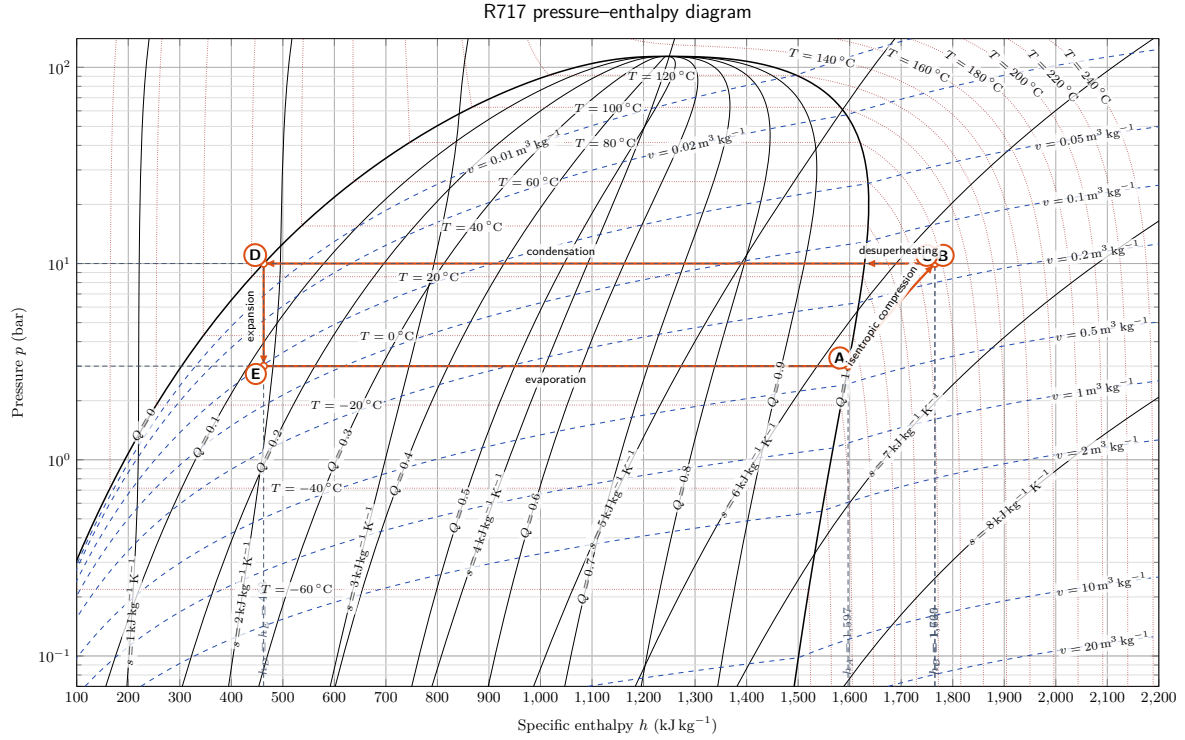
For reading aids, combine the intersection with a coordinate on the axis edge. The TikZ operators `|-` and `-|` copy one canvas component from each coordinate. This remains correct when the axis is resized and when the pressure axis is logarithmic.

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 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`.

1.9 Result: the model-solution diagram



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.

1.10 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

$$w_{A \rightarrow B} = h_B - h_A = 1765.42 - 1596.90 = 168.52 \text{ kJ kg}^{-1}.$$

The heat received by the fluid in the evaporator is

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

and the ideal coefficient of performance is therefore

$$\text{COP} = \frac{q_{E \rightarrow A}}{w_{A \rightarrow B}} \simeq 6.73.$$

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

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

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

Then

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

and

$$\dot{W}_{A \rightarrow B} = \dot{m} w_{A \rightarrow B} \simeq 3.71 \text{ kW}.$$

A useful dimensional check

A result of 22 kg s^{-1} is three orders of magnitude too large for a 25 kW evaporator in this example. The intended value is about 22 g s^{-1} , or $2.2 \times 10^{-2} \text{ kg s}^{-1}$. 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 $1.3 \times 10^3 \text{ kJ kg}^{-1}$.

1.11 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 already enforces that order before compiling this manual.

1.12 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.

1.13 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;
- $A \rightarrow B$ follows an isentrope, $D \rightarrow E$ 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/test.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.

2 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.

2.1 Translate the statement into thermodynamic constraints

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

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

2.2 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  $h$  ( $\text{kJ}\cdot\text{kg}^{-1}$ )},
```

```

    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.

2.3 Stage 1: identify the three phase regions

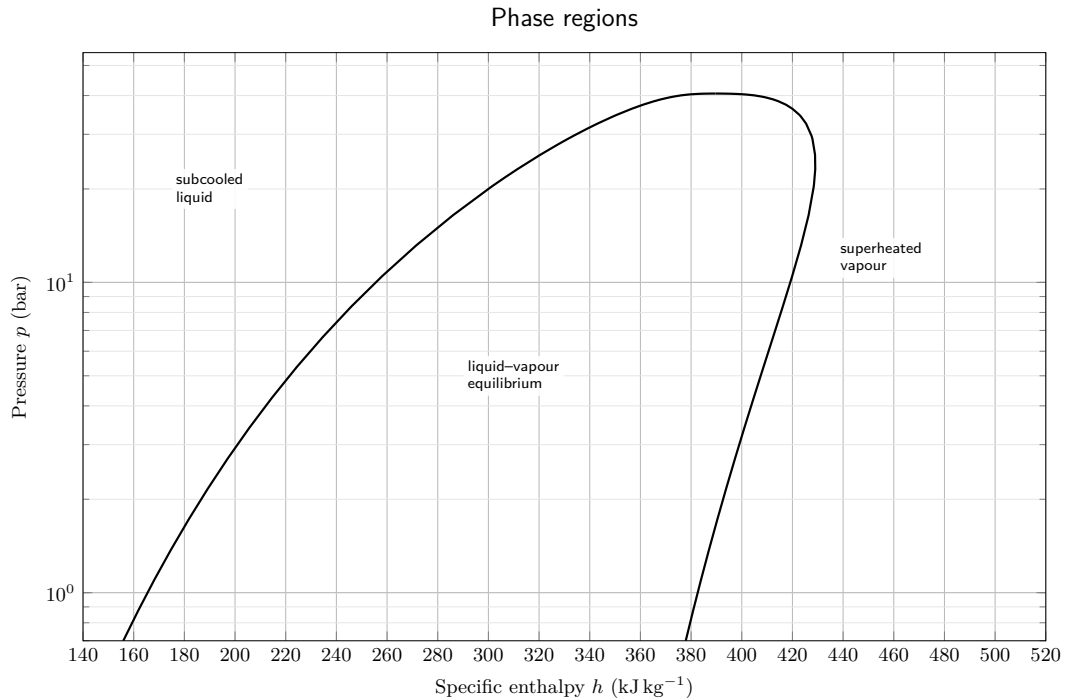
The saturated-liquid and saturated-vapour boundaries are the two curves generated by qualities 0 and 1. They delimit subcooled liquid on the left, liquid–vapour equilibrium below the dome, and superheated vapour on the right. The two branches remain distinct at the low-pressure triple point and converge 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

```

\LCAddPHQuality[
  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};

```



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`.

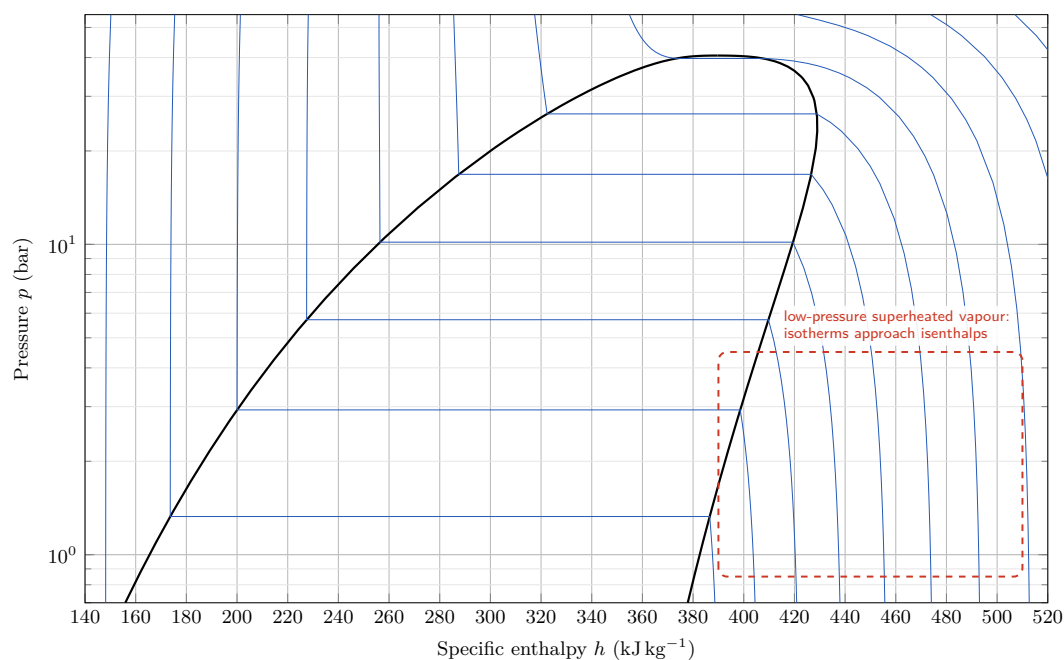
2.4 Stage 2: recognize the ideal-gas region

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
]
```

Where the ideal-gas approximation becomes plausible



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.

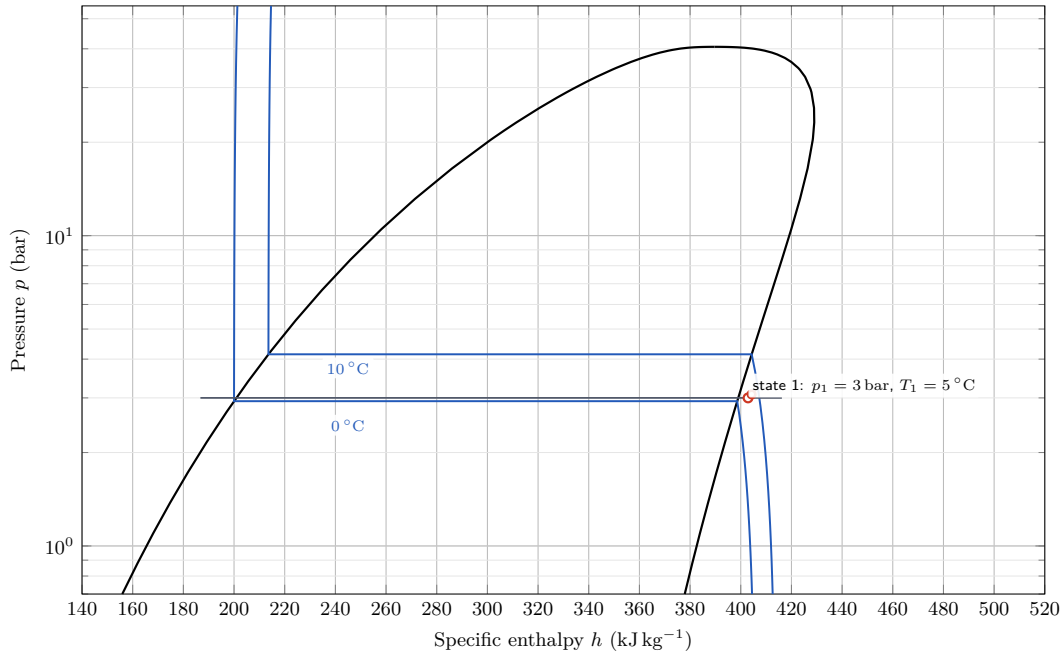
2.5 Stage 3: locate state 1 from two measured properties

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};
```

Locating state 1 from pressure and temperature



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

2.6 Stage 4: construct the compressor outlet

The compression ratio gives

$$p_2 = 6p_1 = 18 \text{ bar}.$$

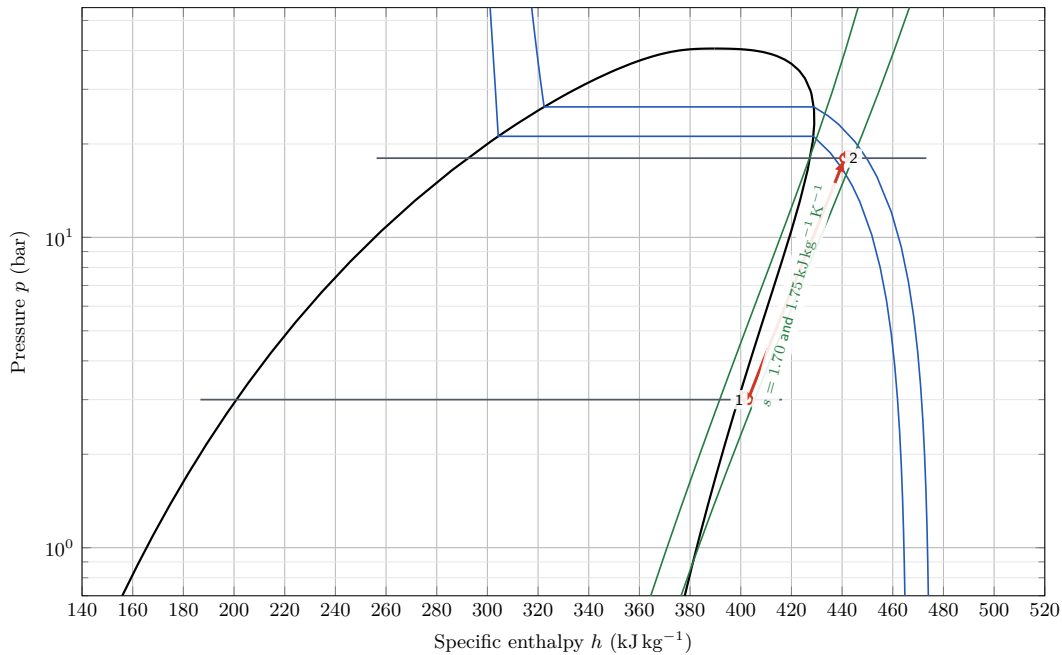
The compressor is adiabatic and reversible in the model, so the path is an isentrope. The process command can compute its conserved entropy directly from state 1:

Isentropic compression from state 1 to state 2

```
\LCPAddPHPProcess[
  type=isentropo,
  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 $1.75 \text{ kJ kg}^{-1} \text{ K}^{-1}$ 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.

Following the isentropic compression to state 2



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

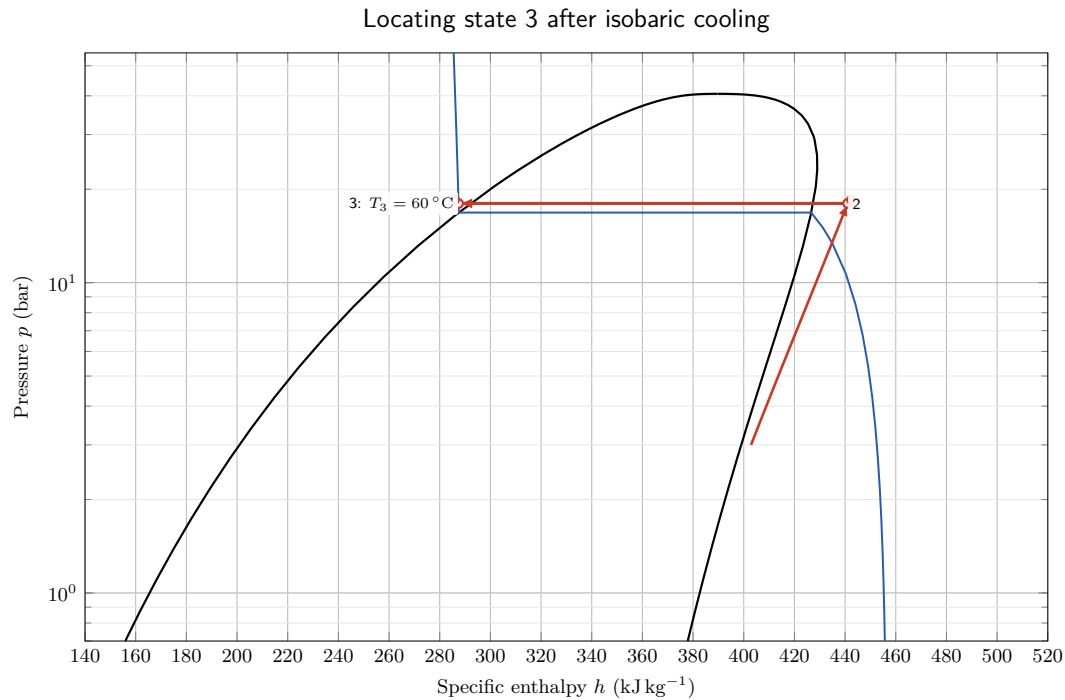
2.7 Stage 5: cool at constant pressure to state 3

State 3 is fixed by $p_3 = p_2 = 18 \text{ bar}$ and $T_3 = 60^\circ\text{C}$. The complete $2 \rightarrow 3$ 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 $1 \rightarrow 2$ and $2 \rightarrow 3$ paths meet exactly. The TikZ intersection remains the sole source of the displayed point position.



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

2.8 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

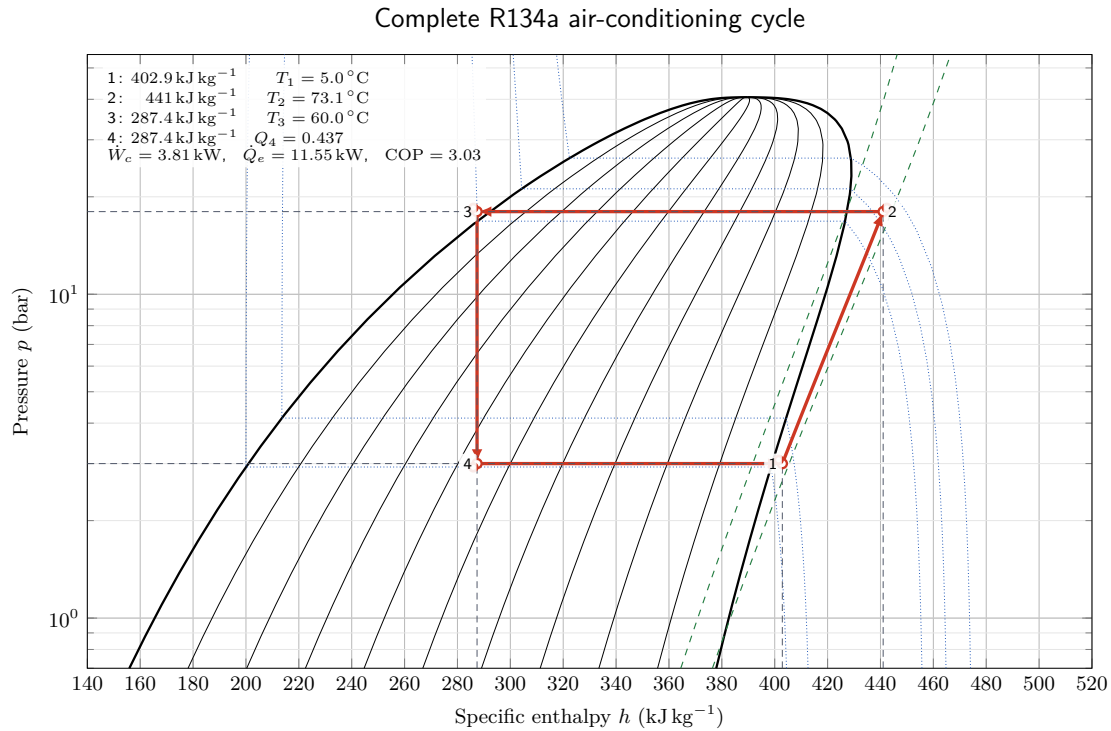
```
\path[name intersections={of=r134a-four-one and r134a-one-two,
  by=RAirOne}];
```

```

\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 $T_4 = 273.822 \text{ K} = 0.672^\circ\text{C}$ and $Q_4 = 0.4367$, compared with the correction's graphical estimates of 1°C and 0.43.



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.

2.9 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 kJ kg^{-1} and mass flow is in kg s^{-1} , their product is directly in kilowatts:

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

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

and

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

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

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

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.

2.10 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 LUACOOOLPROP_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.

3 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.

3.1 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

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

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.

3.2 Keep one executable source for all five stages

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

File	Responsibility
<code>tutorial/butane-pv.tex</code>	Selects the standalone multi-picture document.
<code>tutorial/source/butane-pv-diagrams.tex</code>	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.

3.3 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 $\text{m}^3 \text{kg}^{-1}$, 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  $v$  ( $\text{m}^3/\text{kg}$ )},
  ylabel={Pressure  $p$  ( $\text{Pa}$ )},
  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,
```

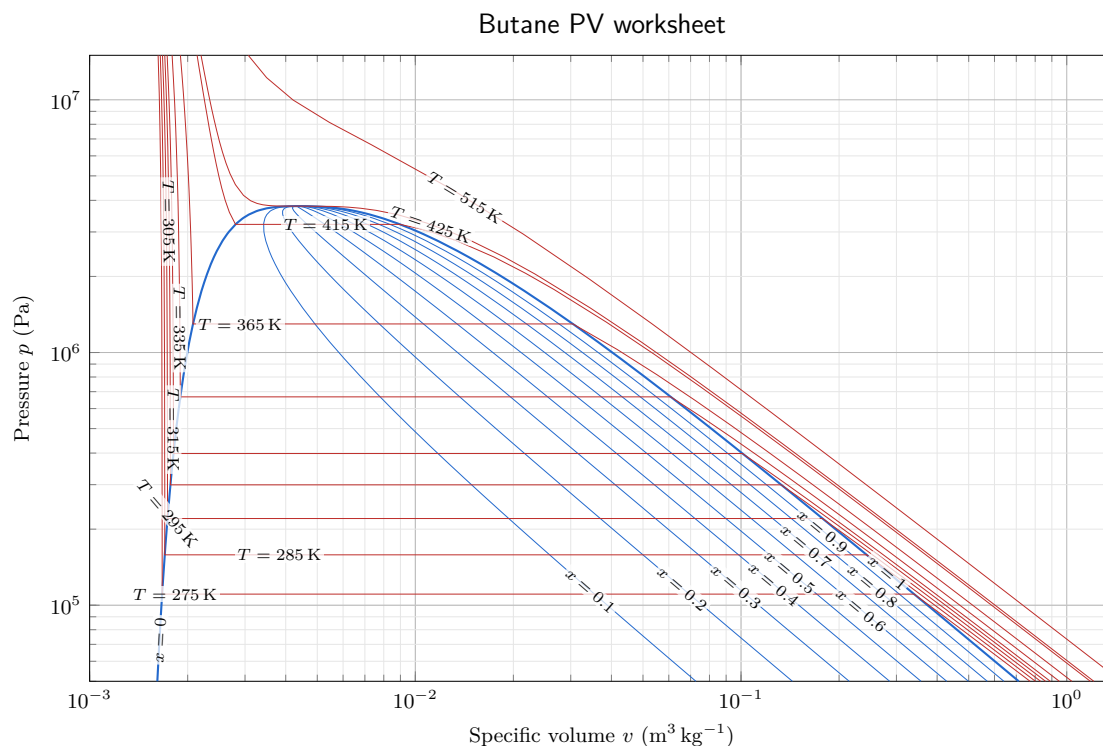
```
quality label sloped=true
]
```

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`.



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.

3.4 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:

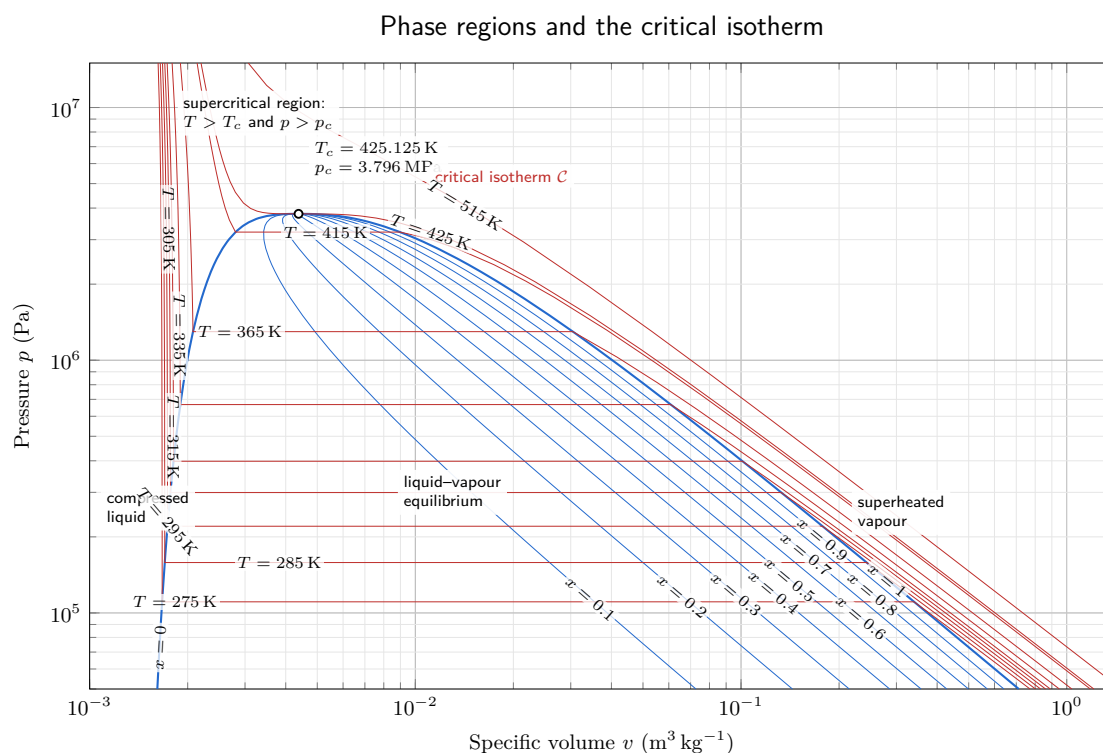
Critical coordinates from the fluid macros

```
\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,

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

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



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.

3.5 Step 3: recognize the ideal-gas region geometrically

For a perfect gas,

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

so a fixed-temperature line satisfies

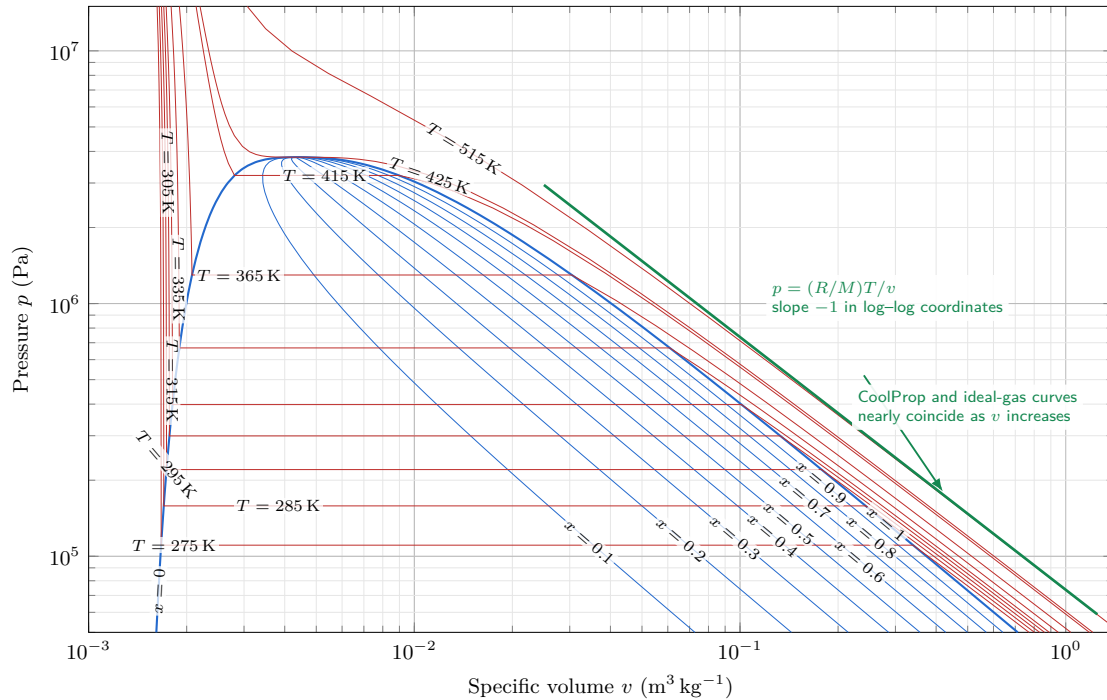
$$pv = \frac{R}{M}T, \quad \log p = -\log v + \log\left(\frac{R}{M}T\right).$$

An ideal-gas isotherm is therefore a straight line of slope -1 in log-log PV coordinates. For butane, with $M = 0.05812 \text{ kg mol}^{-1}$, 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};
```

Ideal-gas limit of the 515 K isotherm



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 $v = 0.2 \text{ m}^3 \text{ kg}^{-1}$ and $Z = 0.9934$ at $v = 0.5 \text{ m}^3 \text{ kg}^{-1}$. The phrase “ideal-gas region” therefore expresses an accuracy decision, not an exact boundary.

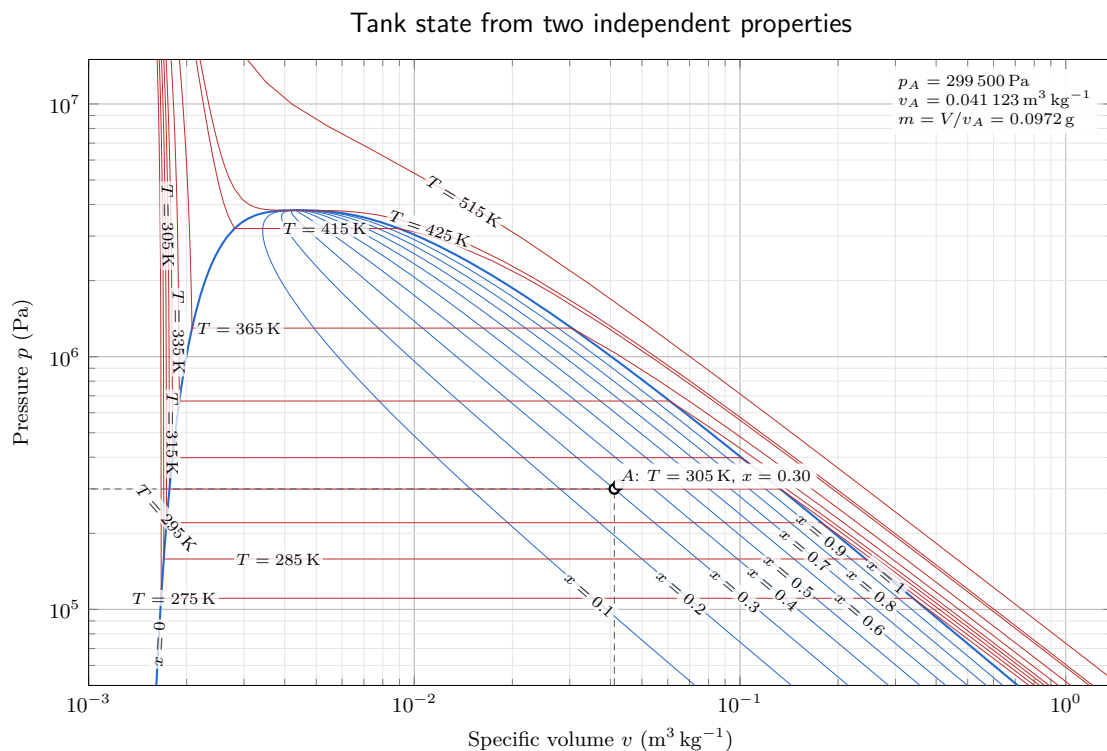
3.6 Step 4: locate the initial state with TikZ intersections

The initial state must satisfy two independent constraints: $T_A = 305\text{ K}$ 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.



The equation-of-state evaluation gives

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

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

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

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

3.7 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

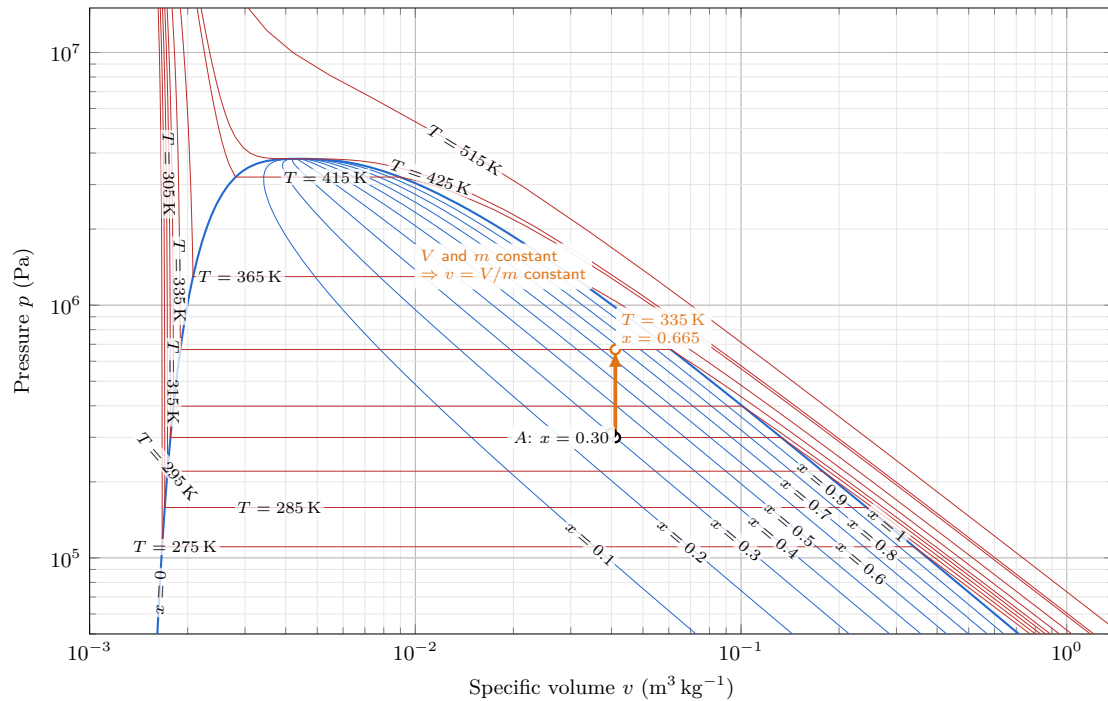
$$v = \frac{V}{m} = \text{constant}.$$

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
]
```

Heating a closed rigid butane tank



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

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

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.

A regression exposed by this tutorial

This example also tests an important API case. 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. The numerical test suite now exercises this butane path explicitly.

3.8 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 LUACOOLOPROP_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 `\LCPAddPVPProcess` 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.

4 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 $\dot{m} = 5.0 \text{ kg s}^{-1}$.

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.

4.1 Translate the statement into four idealized devices

Begin by separating saturation temperatures from inlet and outlet temperatures. The evaporator operates at -10°C and its outlet receives 10 K of superheating, so

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

The condenser operates at 50°C and its outlet receives 10 K of subcooling, so

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

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

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

4.2 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

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

Neglecting kinetic- and potential-energy changes gives

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

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.

4.3 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	Owens 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.

4.4 Step 1: reproduce the blank PH chart

The worksheet uses SI coordinates, so the tutorial overrides the usual LuaCoolProp scales: h remains in 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  $h$  ( $\text{J kg}^{-1}$ )},
  ylabel={Pressure  $p$  ( $\text{Pa}$ )},
  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

```

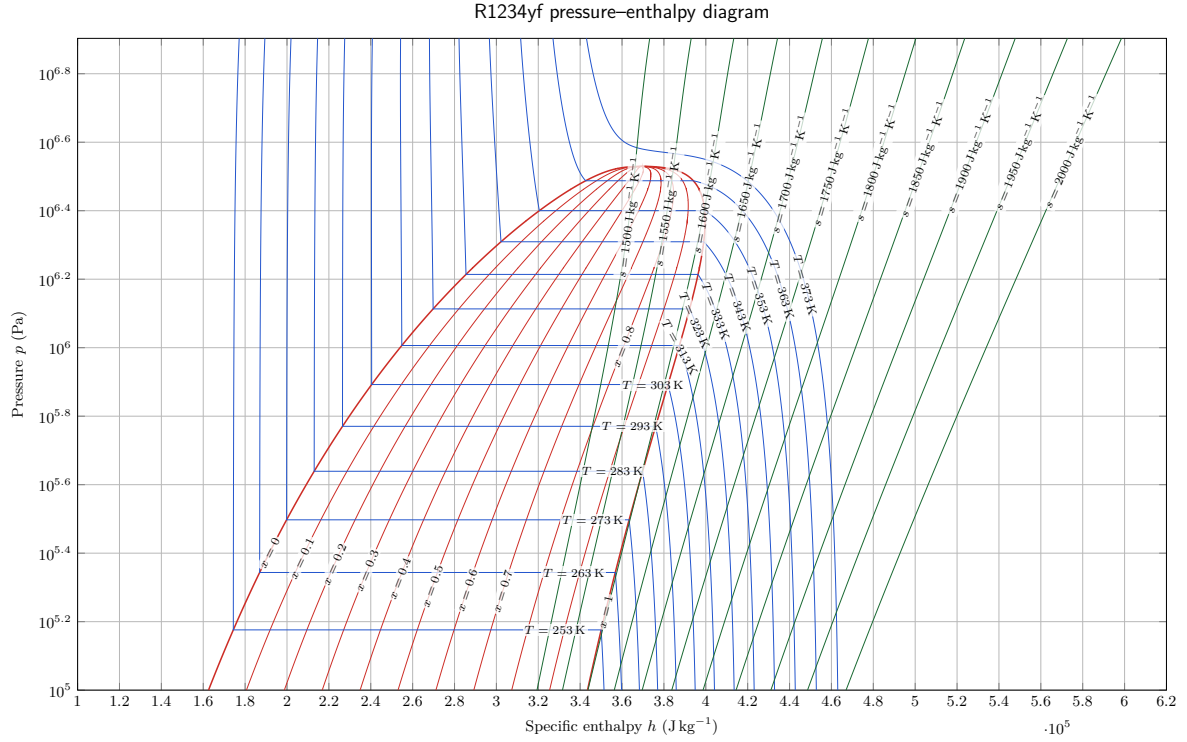
\LCAddPHQuality[
  fluid=R1234yf,
  pressure min=100000,pressure max=8000000,
  enthalpy scale=1,pressure scale=1,
  quality values={0,0.1,0.2,0.3,0.4,0.5,
    0.6,0.7,0.8,0.9,1},
  quality color=red,quality boundary color=red,
  labels=true,quality label sloped=true]

\LCAddPHIsotherms[
  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]

\LCAddPHIsentropes[
  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.



4.5 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°C ; the high-pressure helper starts at saturated liquid at 50°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:

- $\backslash\text{RHeatPumpLowSaturationFromPressureSI}$ stores p_e ;
- $\backslash\text{RHeatPumpHighSaturationFromPressureSI}$ stores p_c .

They give

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

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

4.6 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=isentropo,
  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.

4.7 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

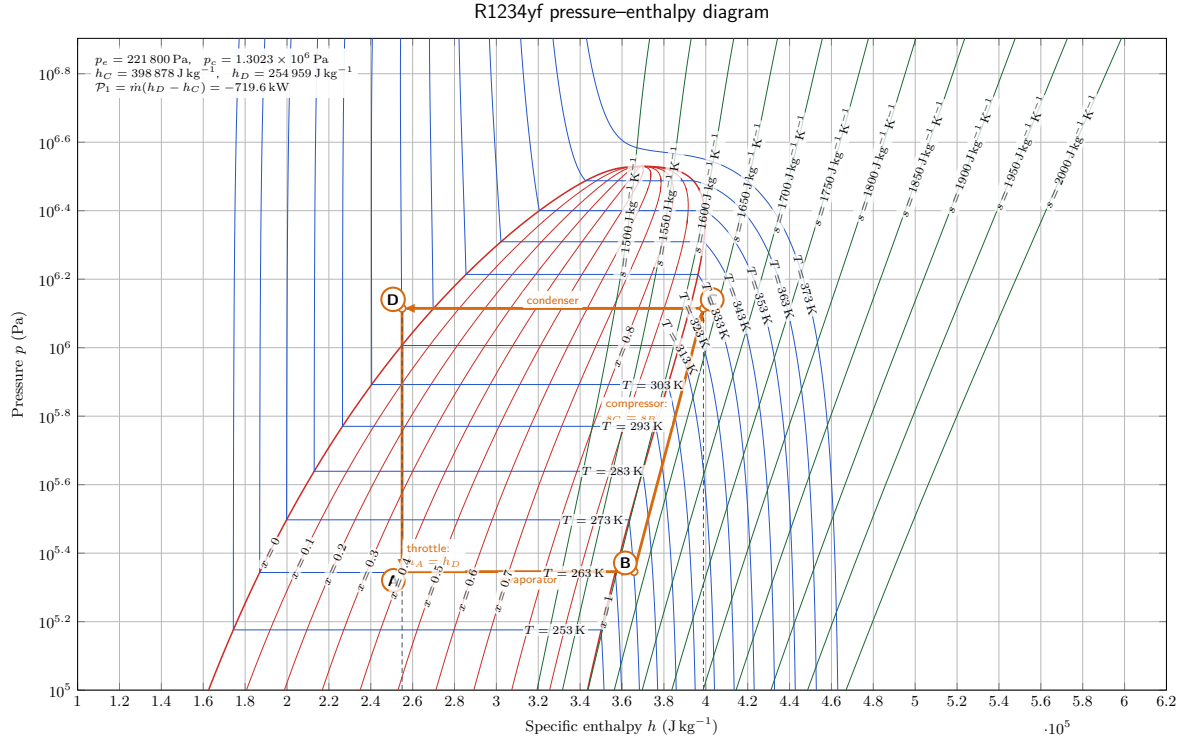
```

\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.



The four exact states generated by CoolProp are:

State	p/bar	$T/^\circ\text{C}$	x	$h/\text{kJ kg}^{-1}$	$s/\text{kJ kg}^{-1} \text{K}^{-1}$
<i>A</i>	2.21785	−10.000	0.39960	254.959	1.21006
<i>B</i>	2.21785	0.000	—	365.744	1.63042
<i>C</i>	13.0235	55.318	—	398.878	1.63042
<i>D</i>	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.

4.8 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 $h_D = 254.959 \text{ kJ kg}^{-1}$,

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

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 kW.

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

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

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.

4.9 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.

5 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.

5.1 Read the process as a directed network

The fresh methane flow rate is $\dot{m} = 1.0 \text{ kg s}^{-1}$ at state 0, with $p_0 = 1 \text{ bar}$ and $T_0 = 7^\circ\text{C}$. 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

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

State 7 is fixed by $p_7 = 100 \text{ bar}$ and $T_7 = -63^\circ\text{C}$. The hot side of the regenerator cools this stream isobarically to $T_{7'} = -82^\circ\text{C}$. 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:

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

5.2 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
```

5.3 Step 1: regenerate the blank methane PH chart

The worksheet covers approximately $-100 \leq h \leq 1200 \text{ kJ kg}^{-1}$ and $0.2 \text{ bar} \leq p \leq 200 \text{ bar}$. 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 $x = 0, 0.1, \dots, 1$ define the two-phase region;
2. isotherms from 103 K to 373 K locate the directly specified temperatures and support interpolation;
3. isentropes from $1 \text{ kJ kg}^{-1} \text{ K}^{-1}$ to $6.5 \text{ kJ kg}^{-1} \text{ K}^{-1}$ construct the ideal compressor outlets; and
4. isochores from $0.005 \text{ m}^3 \text{ kg}^{-1}$ to $5 \text{ m}^3 \text{ kg}^{-1}$ 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

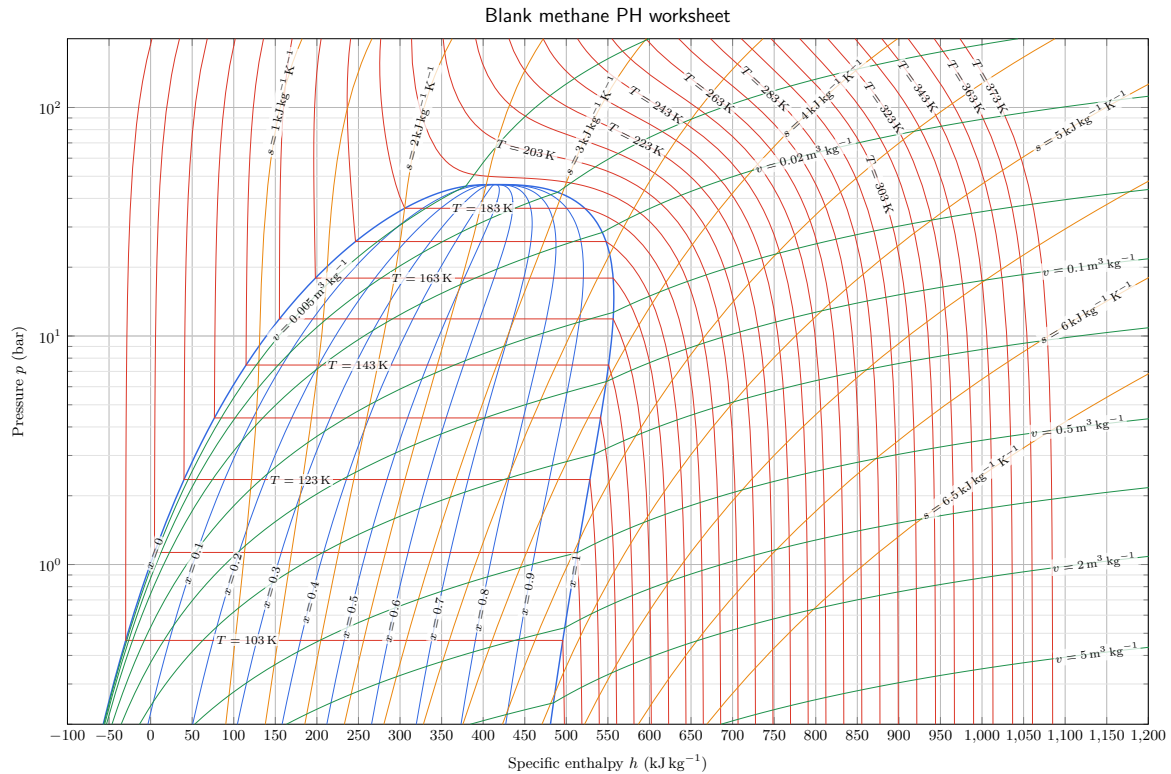
```

\LCAddPHIsentropes[
  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]

\LCAddPHIsochores[
  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`.



5.4 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]

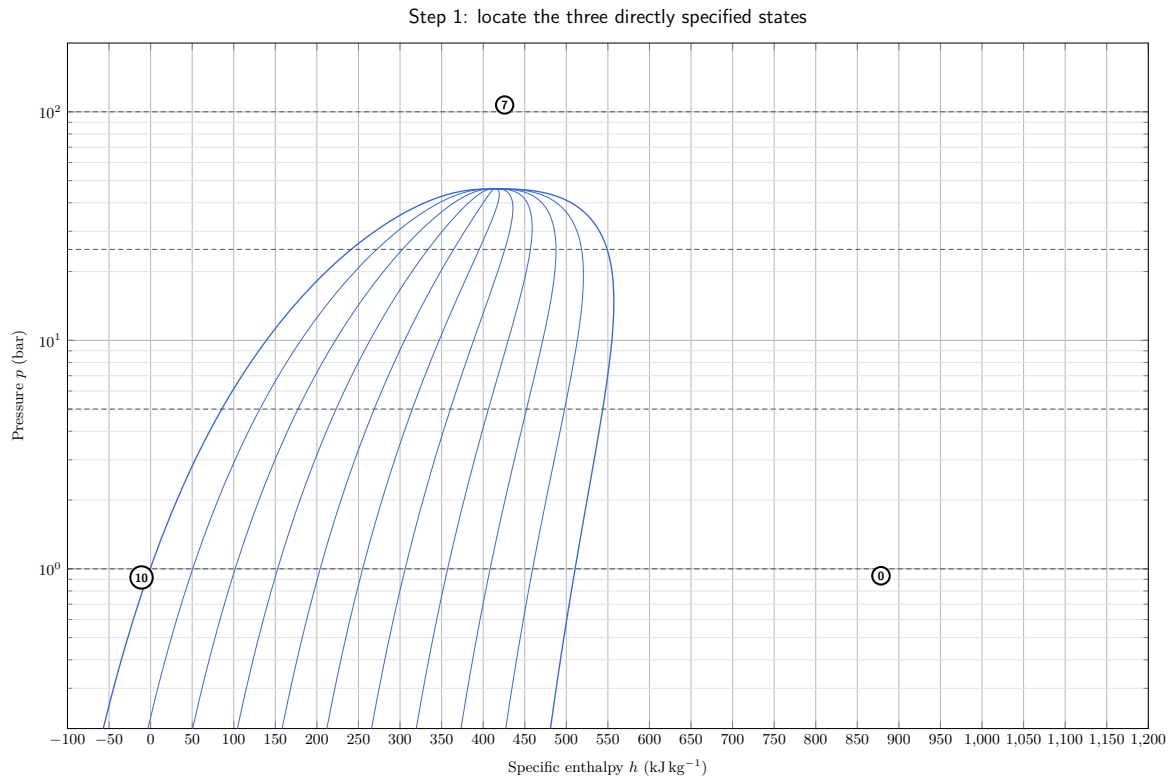
\LCPAddPHProcess[
  fluid=Methane,type=isobar,pressure=1bar,
  from={quality=0},to={quality=1},
  style={draw=none},
  export coordinates=MethaneLindeSaturation]
```

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.



5.5 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

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

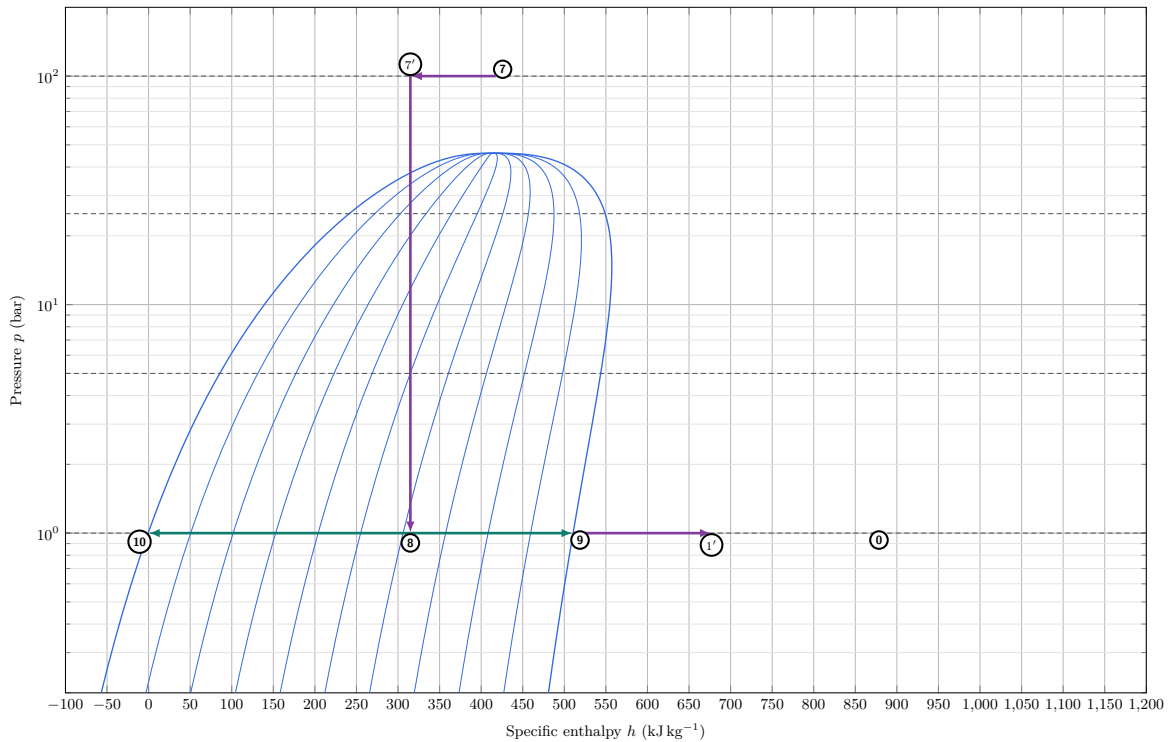
The separator is isobaric. It does not follow a reversible thermodynamic path between 8, 9, and 10; it routes the two phase fractions into two outlet streams. The two horizontal arrows are therefore a flow-topology representation, not equilibrium trajectories undergone by individual fluid particles.

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]
```

Step 2: regenerator, throttle, and phase separator



5.6 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

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

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

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

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

5.7 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 $h_{1'} = 677.122\,\text{kJ kg}^{-1}$. 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.

5.8 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

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

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

5.9 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 $p_2 = 5 \text{ bar}$.

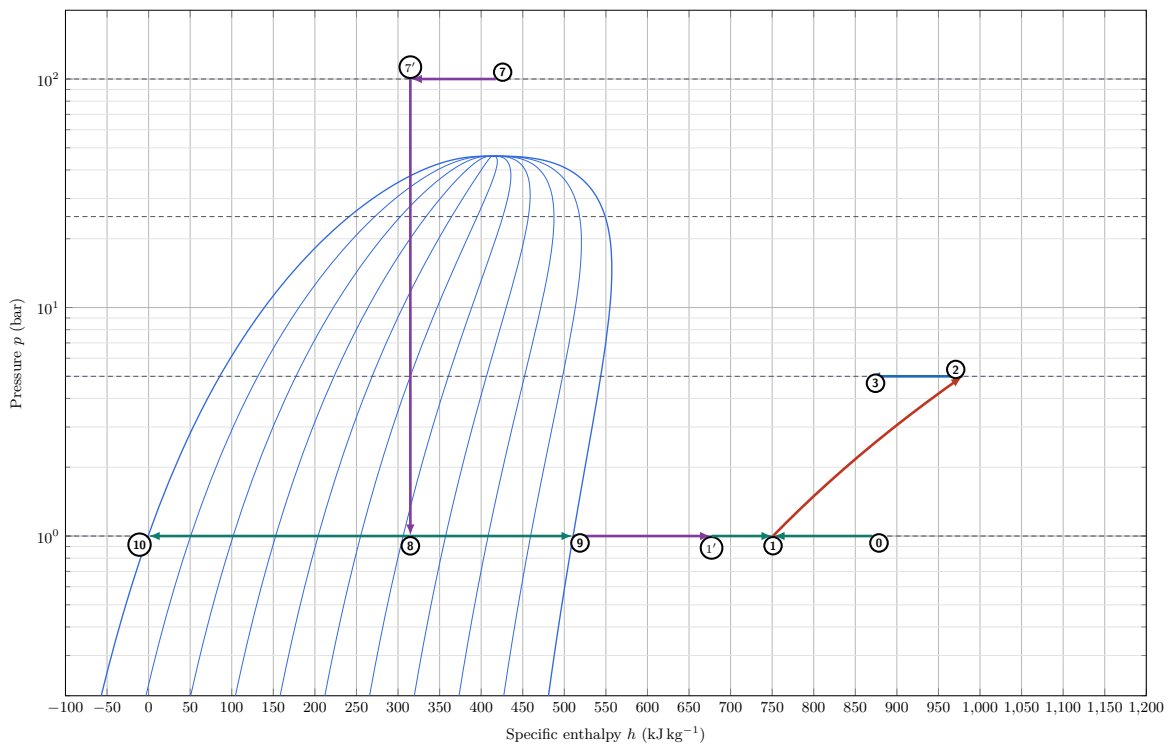
First compressor and first isobaric cooler

```
\LCPAddPHProcess[
  fluid=Methane,type=isentropo,
  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 $h_2 = 979.046 \text{ kJ kg}^{-1}$ and $T_2 = 56.861^\circ\text{C}$. State 3 is not inferred from a temperature curve: its enthalpy is explicit in the statement.

Step 3: close the recirculation balance and draw the first stage



5.10 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=isentropo,
  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]
```

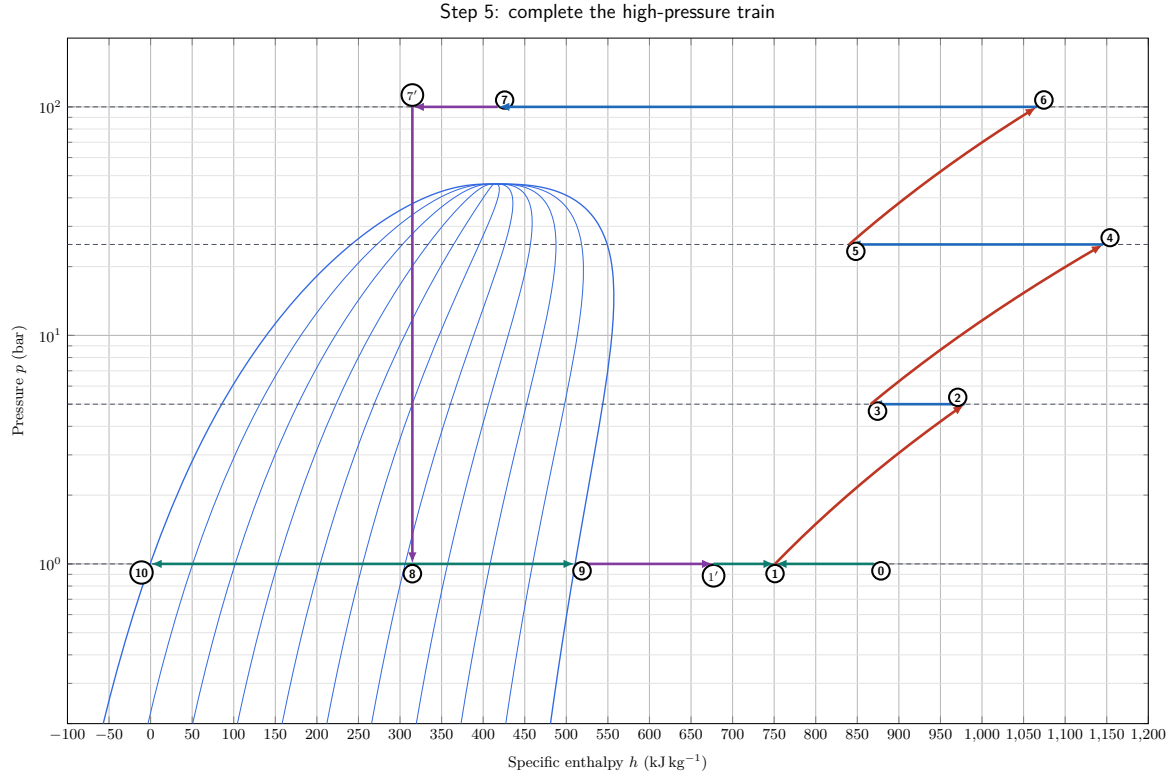
This yields $h_4 = 1145.817 \text{ kJ kg}^{-1}$ and $T_4 = 129.797^\circ\text{C}$. The third stage ends at the given state 7:

Third compression stage

```
\LCPAddPHProcess[
  fluid=Methane,type=isentropo,
  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 $h_6 = 1066.089 \text{ kJ kg}^{-1}$ at $T_6 = 115.236^\circ\text{C}$.



5.11 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

$$\dot{Q}_{E1} = \dot{m}_1(h_3 - h_2),$$

$$\dot{Q}_{E2} = \dot{m}_1(h_5 - h_4),$$

$$\dot{Q}_{E3} = \dot{m}_1(h_7 - h_6).$$

All three must be negative. Their automated values are

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

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}}
```

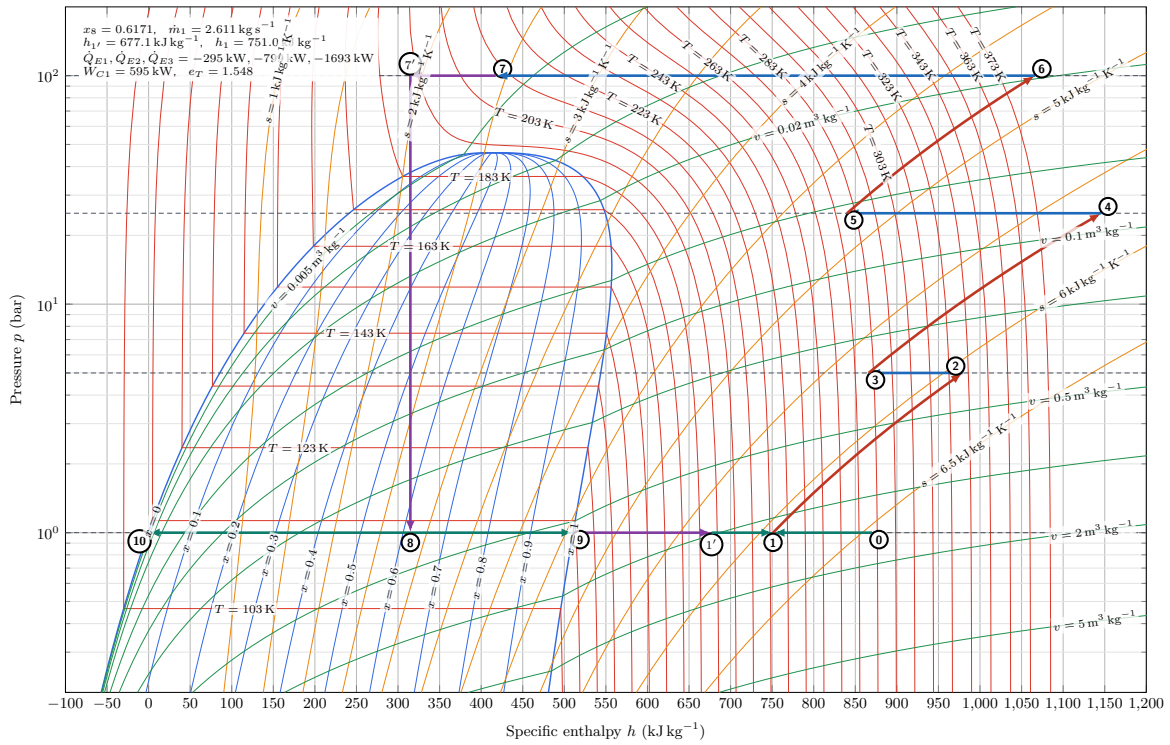
5.12 Step 10: read the complete process without inventing a cycle

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.

Step 6: complete Linde process and live numerical audit



The calculated state table is:

State	p/bar	$T/^\circ\text{C}$	$h/\text{kJ kg}^{-1}$	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.

5.13 Graphical estimates versus live calculations

The approximate column uses values read from a printed chart, with an uncertainty of approximately $\pm 20 \text{ kJ kg}^{-1}$. 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
$h_0/\text{kJ kg}^{-1}$	860	870.145
$h_7/\text{kJ kg}^{-1}$	410	417.612
$h_{7'} = h_8/\text{kJ kg}^{-1}$	300	314.835
$h_9/\text{kJ kg}^{-1}$	440	510.562
x_8	0.60	0.61706
$\dot{m}_1/\text{kg s}^{-1}$	2.5	2.61139
$h_{1'}/\text{kJ kg}^{-1}$	620	677.122
$h_1/\text{kJ kg}^{-1}$	720	751.038
$h_2/\text{kJ kg}^{-1}$	960	979.046
$h_4/\text{kJ kg}^{-1}$	1110	1145.817
$h_6/\text{kJ kg}^{-1}$	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.

5.14 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 LUACOOPLPROP_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 II

User manual

6 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`.

AI-assisted “vibe coding”

This package has been *vibe-coded* with ChatGPT, an artificial-intelligence system from OpenAI. Substantial parts of its design, implementation, documentation, and tests have been generated or revised with AI assistance. AI-generated work can be convincing while still containing incorrect code, incorrect thermodynamic assumptions, incomplete tests, or misleading documentation. Publication on CTAN is not a safety certification. Inspect the source and independently validate every result before relying on it, especially in industrial, regulated, safety-critical, or high-consequence work. The maintainers remain responsible for reviewing releases and welcome reproducible issue reports.

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.

6.1 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](https://doi.org/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.

7 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.

7.1 Install the TeX files

After publication, 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/>.

7.2 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.

7.3 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.

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.

7.4 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 can clone CoolProp’s current default branch for exploratory use:

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. Package maintainers who intentionally adopt a new CoolProp API may use `./scripts/update-coolprop.sh v8.0.0`; unlike the build-only script, that maintainer tool also regenerates the marked FFI declaration block in `luacoolprop.lua`, so end users should not run it as a routine installation step. A release tag may be written with or without its leading `v`.

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.

7.5 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`.

7.6 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"`.

7.7 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

```
% check-luacoolprop.tex
\documentclass{article}
\usepackage{luacoolprop}
\begin{document}
```

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

Run the LaTeX installation check

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

The library path is consumed when the Lua module first loads. 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.

7.8 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

```
\documentclass{article}
\usepackage{luacoolprop}
\begin{document}
\LCPPHDiagram[fluid=R134a]
\end{document}
```

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.

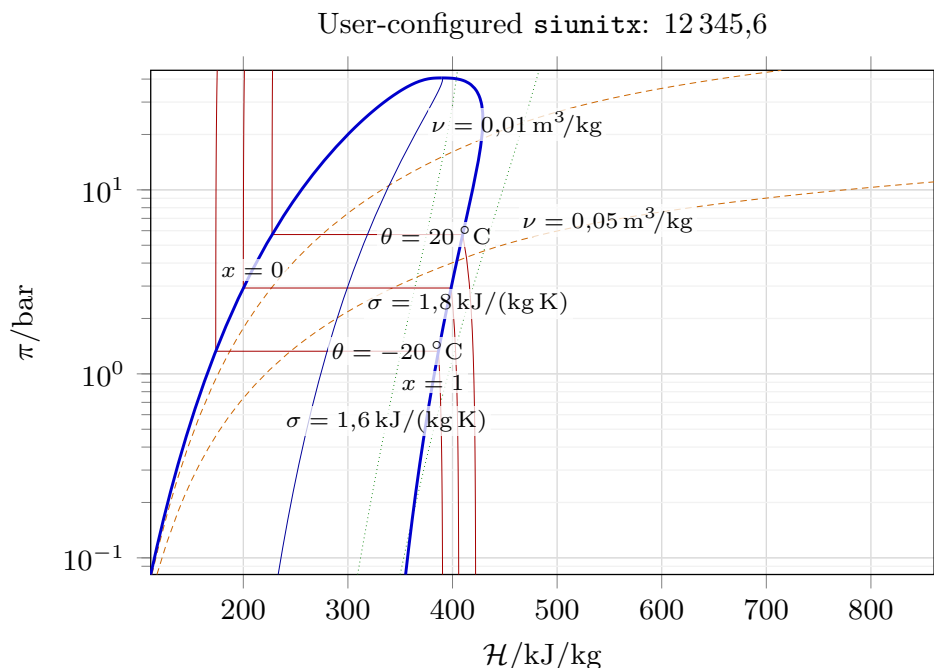
7.9 LaTeX number and unit formatting with `siunitx`

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`.

Configure `siunitx` before LuaCoolProp

```
\documentclass[border=3mm]{standalone}
\usepackage[
  output-decimal-marker={,},
  group-separator={\,},
  group-minimum-digits=4,
  per-mode=symbol
]{siunitx}
\usepackage{luacoolprop}
\begin{document}
  \LCPPHDiagram[
    quality values={0,0.5,1},
    isotherm=true,temperature values={-20,0,20},
    labels=true]
\end{document}
```

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.

7.10 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.

8 Basic users: quick start

8.1 A fully automatic diagram

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

Minimal PH diagram

```
\documentclass[border=3mm]{standalone}
\usepackage{luacoolprop}

\begin{document}
\LCPPHDiagram[fluid=R134a]
\end{document}
```

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
  }
]
```

8.2 Another fluid

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

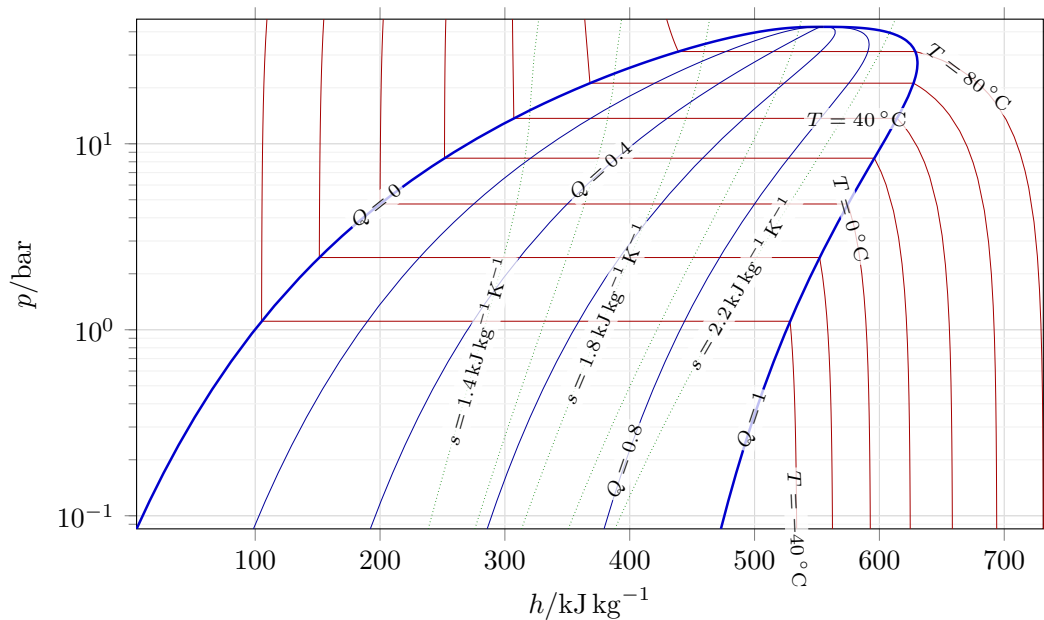
Propane quick diagram

```
\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
]

```



9 Diagram identifiers and stable API

Diagram identifiers are uppercase coordinate pairs. This release implements PH, PV, and TS. 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`, `isentrop`, `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.

10 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.

Manual construction in an axis

```
\begin{tikzpicture}
\begin{axis}[
  lcp fluid=R134a,
  xlabel={\$/\rm kJ\,kg^{-1}}$,
  ylabel={\$/\rm bar}$,
  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
]
\LCAddPHIsentropes[
    entropy values={1.0,1.2,1.4,1.6,1.8},
    labels=true,
    sloped labels=true
]
\LCAddPHIsochores[
    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}

```

10.1 Curve-family macros

Macro	Purpose
<code>\LCAddPHQuality</code>	add Q = constant curves
<code>\LCAddPHIsotherms</code>	add T = constant curves
<code>\LCAddPHIsentropes</code>	add s = constant curves
<code>\LCAddPHIsochores</code>	add v = constant curves
<code>\LCAddPHProcess</code>	add one thermodynamic process segment

10.2 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

```

\LCAddPHQuality[quality values={0,0.25,0.5,0.75,1}]
\LCAddPHIsotherms[temperature values={-20,0,20,40,60,80}]
\LCAddPHIsentropes[entropy values={1.0,1.2,1.4,1.6,1.8}]
\LCAddPHIsochores[specific volume values={0.001,0.002,0.005,0.01,0.02}]

```

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

11 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 standalone `pgfplots-autonode` 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:

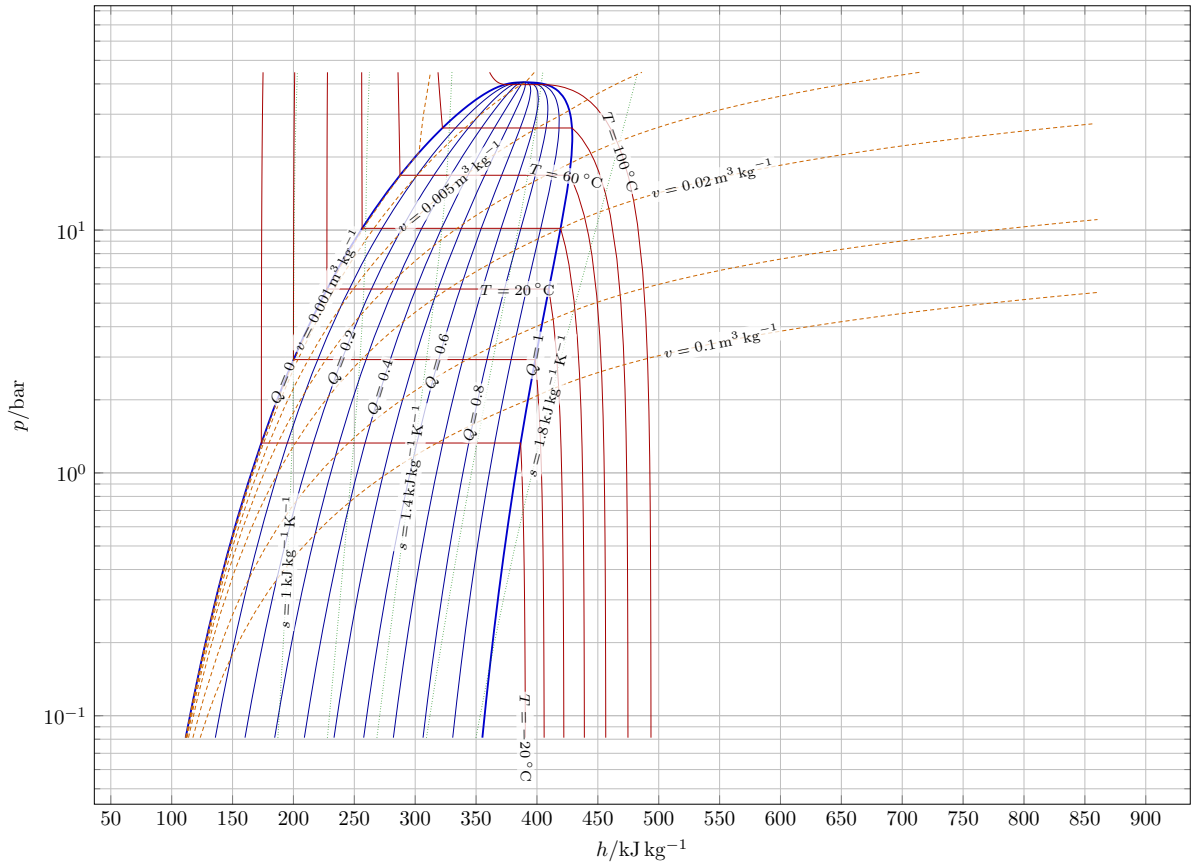
Generated structure

```
\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

```
\begin{tikzpicture}
\begin{axis}[
  lcp fluid=R134a,
  xlabel={\rm kJ\,kg^{-1}},
  ylabel={\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,
]
\LCAddPHQuality[
  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
]
\LCAddPHIsotherms[
  temperature values={-20,0,20,40,60,80,100},
  labels=true,
  sloped labels=true
]
\LCAddPHIsentropes[
  entropy values={1.0,1.2,1.4,1.6,1.8},
  labels=true,
  sloped labels=true
]
\LCAddPHIsochores[
  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}
```



11.1 LuaCoolProp keys forwarded to autonode

Key	Meaning
<code>autonode candidates</code>	number of candidate positions sampled on each plot path
<code>autonode candidate strategy</code>	distribution strategy for candidates
<code>autonode clearance</code>	additional clearance around measured labels
<code>autonode inner sep</code>	node padding used by the backend
<code>autonode normal shift</code>	shift normal to the sampled curve
<code>autonode preferred weight</code>	penalty for moving away from the preferred position
<code>autonode overlap weight</code>	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.

12 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}
```

13 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

```
\LCPAddPHProcess[  
  fluid=R134a,  
  type=isentropo,  
  from={pressure=1bar,quality=1},  
  to={pressure=10bar},  
  style={very thick,-Latex},  
  color=black,  
  label={ $1 \rightarrow 2$ },  
  mark endpoints=true  
]
```

Constant-quality segment

```
\LCPAddPHProcess[  
  fluid=R134a,  
  type=quality,  
  quality=0.5,  
  from={temperature=50C},  
  to={temperature=-20C},  
  style={very thick,-Latex},  
  color=purple!80!black,  
  label={ $Q=0.5$ }  
]
```

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 III

Complete TeX reference

14 How to read this reference

This part is the normative TeX reference for `luacoolprop`. Every public command has a blue entry; every public key has a green 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 left is exactly the code which produced the result on the right. Longer examples put the result below the source. This is the same literate, executable-documentation principle used by the PGF/TikZ and PGFPlots manuals.

Signatures use $\langle argument \rangle$ for a required semantic value, $\{\langle argument \rangle\}$ for a braced argument, and $[\langle options \rangle]$ 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.

15 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

```
\documentclass{article}
\usepackage{luacoolprop}
\begin{document}
  \LCPPHDiagram
\end{document}
```

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.

16 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>}`

Sets the CoolProp shared-library path for subsequent calls. Usually the environment variable `LUACOOOLPROP_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

<pre>\LCPDeclareFluid{R134a} Fluid: \LCPFluidName\par \$T_{\mathrm c}=\LCPFluidTcritC\,\mathrm C\$\par \$p_{\mathrm c}=\LCPFluidPcritBar\,\mathrm{bar}\$</pre>	<pre>Fluid: R134a T_c = 101.061966585 °C p_c = 40.5927637379 bar</pre>
--	--

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.

16.1 Global keys: /luacoolprop

fluid=*<CoolProp identifier>* Default: R134a

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

Example: fluid=Propane.

library=*<file path>* Default: empty

Overrides discovery of the external shared library. Prefer LUACOOOLPROP_LIB in reproducible workflows.

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

reference state=*<DEF, IIR, ASHRAE, or NBP>* 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.

enthalpy axis unit=*<jkg or kjpg>* Default: kjpg

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

Example: enthalpy axis unit=jkg.

pressure axis unit=*<pa, kpa, mpa, or bar>* Default: bar

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

Example: pressure axis unit=mpa.

`specific volume axis unit=⟨m3kg or lkg⟩` Default: m3kg

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

Example: `specific volume axis unit=lkg`.

`entropy axis unit=⟨jkgk or kjkgk⟩` Default: kjkgk

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

Example: `entropy axis unit=jkgk`.

`temperature axis unit=⟨kelvin or celsius⟩` Default: kelvin

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

Example: `temperature axis unit=celsius`.

`enthalpy scale=⟨number⟩` 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`.

`pressure scale=⟨number⟩` 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`.

`specific volume scale=⟨number⟩` 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`.

`entropy scale=⟨number⟩` 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`.

`temperature scale=⟨number⟩` 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.

17 The generic diagram API

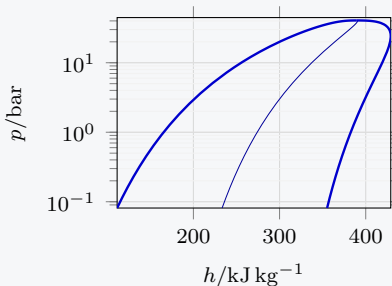
Use this API in libraries or in documents which may later switch diagram coordinates. The diagram identifier is case-sensitive and uppercase.

`\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`, `isentropes`, and `isochore`; PV registers `quality`, `isotherm`, and `isentropes`; TS registers `quality` and `isenthalp`; HS registers `quality`, `isochore`, `isotherm`, and `isobar`; PT registers `phase_envelope`, `isentropes`, `isenthalp`, and `isochore`.

Add only PH isotherms

```
\LCPAddDiagramFamily{PH}{isotherm}[  
  temperature values={-20,0,20,40}]
```

\LCPAddDiagramProcess{<type>}[<options>]

Adds one thermodynamic process path to an existing axis.

Add an isobaric PH process

```
\LCPAddDiagramProcess{PH}[  
  type=isobar,pressure=8bar,  
  from={quality=0},to={quality=1}]
```

17.1 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, `isentropo=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.

17.2 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.

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>.

reference state= $\langle$ *DEF, IIR, ASHRAE, or NBP* $\rangle$

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`.

17.3 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.

domain policy= $\langle$ *ignore, warning, or error* $\rangle$

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

```
\LCPSetup{temperature axis unit=celsius}
\LCPAddTSQuality[quality values={0,1}]
\LCPAddTSProcess[type=isobar,pressure=2bar,
  from={quality=0},to={quality=1}]
```

18 PH convenience commands

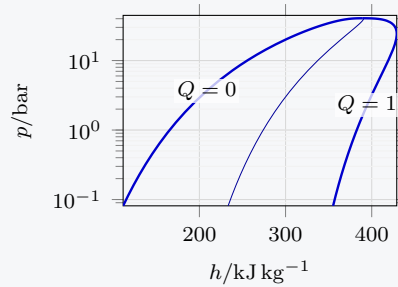
The following commands are stable conveniences for the current PH diagram. Their options use `/luacoolprop/diagram/PH`; process options use `/luacoolprop/process/PH`.

\LCPPHDiagram $\langle$ *options* $\rangle$

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

```
\LCPAddPHPlots[quality=true,isotherm=true,
  temperature values={-20,0,20,40}]
```

\LCPAddPHQuality[*<options>*]

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

Five quality curves

```
\LCPAddPHQuality[quality values={0,0.25,0.5,0.75,1}]
```

\LCPAddPHIsotherms[*<options>*]

Adds only constant-temperature curves.

Explicit Celsius temperatures

```
\LCPAddPHIsotherms[temperature values={-20,0,20,40}]
```

\LCPAddPHIsentropes[*<options>*]

Adds only constant-specific-entropy curves.

Explicit entropy values

```
\LCPAddPHIsentropes[entropy values={1.6,1.8,2.0}]
```

`\LCPAddPHIsochores[<options>]`

Adds only constant-specific-volume curves.

Logarithmic specific-volume grid

```
\LCPAddPHIsochores[specific volume min=0.01,  
specific volume max=0.2,specific volume count=5]
```

`\LCPAddPHProcess[<options>]`

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

Named evaporation process

```
\LCPAddPHProcess[type=isobar,pressure=8bar,  
from={quality=0},to={quality=1},  
name=evaporation,label={evaporator}]
```

18.1 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]`.

19 PH diagram keys

Unless stated otherwise, all keys in this section belong to `/luacoolprop/diagram/PH`. In a command option list, write only the leaf name, for example `fluid=Propane`.

19.1 Fluid, families, ranges, and axis

fluid=*<CoolProp identifier>* Default: global fluid

Selects the fluid for this diagram only.

Example: `fluid=Propane`.

library=*<file path>* Default: global library

Overrides the shared-library path for this diagram.

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

quality=*<boolean>* Default: true

Enables or disables the quality family.

Example: `quality=false`.

isotherm=*<boolean>* Default: false

Enables or disables the isotherm family.

Example: `isotherm=true`.

isentropo=*<boolean>* Default: false

Enables or disables the isentropo family.

Example: `isentropo=true`.

isochore=*<boolean>* Default: false

Enables or disables the isochore family.

Example: `isochore=true`.

pressure min=*<number or auto>* Default: auto

Sets the lower sampling pressure in pascals.

Example: `pressure min=50000`.

pressure max=*<number or auto>* Default: auto

Sets the upper sampling pressure in pascals.

Example: `pressure max=5E6`.

`pressure max factor=` *$\langle$ positive number $\rangle$* 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.`

`enthalpy scale=` *$\langle$ number $\rangle$* 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.`

`pressure scale=` *$\langle$ number $\rangle$* 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.`

`enthalpy axis unit=` *$\langle$ jkg or kjpg $\rangle$* Default: kjpg

Atomically selects the enthalpy multiplier and horizontal-axis unit.

Example: `enthalpy axis unit=jkg.`

`pressure axis unit=` *$\langle$ pa, kpa, mpa, or bar $\rangle$* Default: bar

Atomically selects the pressure multiplier and vertical-axis unit.

Example: `pressure axis unit=mpa.`

`enthalpy symbol=` *$\langle$ TeX math material $\rangle$* Default: h

Sets the enthalpy symbol in the horizontal axis created by `\LCPPHDiagram`. Omit surrounding math delimiters.

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

`pressure symbol=` *$\langle$ TeX math material $\rangle$* Default: p

Sets the pressure symbol in the vertical axis created by `\LCPPHDiagram`. Omit surrounding math delimiters.

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

`axis options=` *$\langle$ PGFPlots options $\rangle$* Default: empty

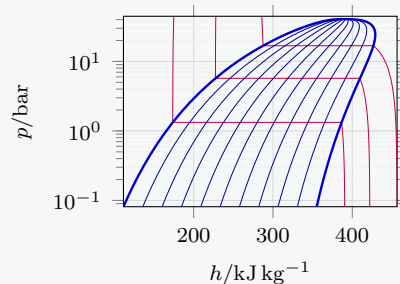
Appends options to the axis created by `\LCPPHDiagram`. Width, height, labels, limits, legend placement, and fonts may all be overridden.

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=4.1
    cm,
    font=\scriptsize}
]
```



19.2 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.

quality min=*<number>*

Default: 0

Lower quality in a generated grid.

Example: quality min=0.1.

quality max=*<number>*

Default: 1

Upper quality in a generated grid.

Example: quality max=0.9.

quality step=*<positive number>*

Default: 0.1

Increment used by a linear grid.

Example: quality step=0.2.

quality count=*<integer or auto>*

Default: `auto`

Requests an evenly distributed number of values.

Example: `quality count=6.`

quality values=*<comma list>*

Default: `empty`

Supplies the exact qualities and overrides generated-grid controls.

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

quality mode=*<choice>*

Default: `linear`

Selects how grid values are obtained.

linear Uniform spacing between the bounds.

log Logarithmic spacing; meaningful only for positive bounds.

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.`

quality preset=*<name>*

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`.

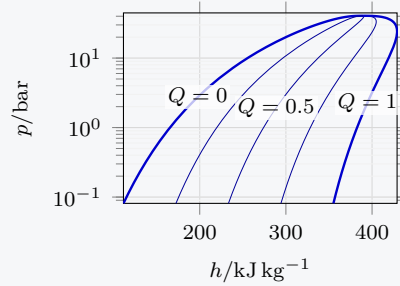
Accepted alias: `q count` forwards to `quality count`.

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=4.1
    cm,
    font=\scriptsize}
]
```



19.3 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.

19.3.1 Temperature and isotherms

`temperature unit=⟨unit⟩`

Default: celsius

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

Example: `temperature unit=kelvin.`

`temperature min=⟨number or auto⟩`

Default: auto

Lower temperature for the generated grid.

Example: `temperature min=-40.`

`temperature max=⟨number or auto⟩`

Default: auto

Upper temperature for the generated grid.

Example: `temperature max=80.`

`temperature step=⟨positive number⟩`

Default: 10

Linear-grid increment in the selected unit.

Example: `temperature step=20.`

`temperature count=⟨integer or auto⟩`

Default: auto

Requests a fixed number of temperatures.

Example: `temperature count=7.`

`temperature values=⟨comma list⟩`

Default: empty

Uses exactly the listed temperatures.

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

temperature mode=*<choice>*

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.

temperature preset=*<name>*

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.

19.3.2 Specific entropy and isentropes

entropy unit=*<unit>*

Default: kJkgk

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

Example: entropy unit=si.

entropy min=*<number or auto>*

Default: auto

Lower entropy for the generated grid.

Example: entropy min=1.4.

entropy max=*<number or auto>*

Default: auto

Upper entropy for the generated grid.

Example: entropy max=2.2.

entropy step=*<positive number>*

Default: 0.1

Linear-grid entropy increment.

Example: entropy step=0.2.

entropy count=*<integer or auto>*

Default: auto

Requests a fixed number of entropy values.

Example: entropy count=6.

entropy values=*<comma list>*

Default: empty

Uses exactly the listed entropy values.

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

entropy mode=*<choice>*

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.

entropy preset=*<name>*

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.

19.3.3 Specific volume and isochores

specific volume unit=*<unit>*

Default: m3kg

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

Example: specific volume unit=1kg.

specific volume min=*<number or auto>*

Default: auto

Lower specific volume for the generated grid.

Example: specific volume min=0.01.

`specific volume max=⟨number or auto⟩` Default: auto

Upper specific volume for the generated grid.

Example: `specific volume max=0.2.`

`specific volume step=⟨number or auto⟩` Default: auto

Increment for a linear grid.

Example: `specific volume step=0.02.`

`specific volume count=⟨integer⟩` Default: 8

Requests a fixed number of volume values.

Example: `specific volume count=6.`

`specific volume values=⟨comma list⟩` Default: empty

Uses exactly the listed volumes.

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

`specific volume mode=⟨choice⟩` 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.`

`specific volume preset=⟨name⟩` 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`.

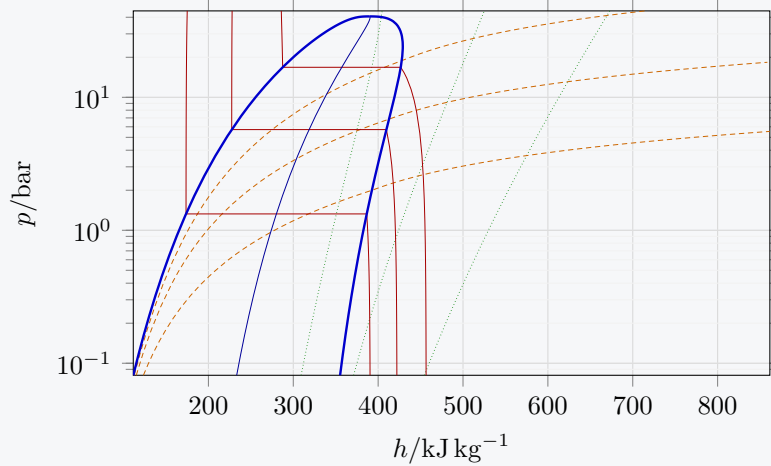
Combining all thermodynamic families

```
\LCPPHDiagram[
  quality values={0,0.5,1},
  isotherm=true,temperature values={-20,20,60},
```

```

isentropo=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}
]

```



19.4 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.

`initial intervals=`*⟨positive integer⟩*

Default: 18

Initial subdivision count for quality curves.

Example: `initial intervals=24`.

`max depth=`*⟨nonnegative integer⟩*

Default: 8

Maximum recursive subdivision depth for quality curves.

Example: `max depth=10`.

tolerance=*⟨positive number⟩*

Default: 0.25

Geometric refinement tolerance for quality curves.

Example: tolerance=0.15.

log weight=*⟨nonnegative number⟩*

Default: 30

Weights pressure changes in logarithmic space during refinement.

Example: log weight=40.

coord digits=*⟨positive integer⟩*

Default: 6

Decimal significant precision emitted to PGFPlots coordinates.

Example: coord digits=8.

isotherm initial intervals=*⟨positive integer⟩*

Default: 14

Initial subdivisions for each isotherm.

Example: isotherm initial intervals=18.

isotherm max depth=*⟨nonnegative integer⟩*

Default: 7

Maximum adaptive depth for isotherms.

Example: isotherm max depth=9.

isotherm tolerance=*⟨positive number⟩*

Default: 0.35

Adaptive tolerance for isotherms.

Example: isotherm tolerance=0.2.

isentropes initial intervals=*⟨positive integer⟩*

Default: 14

Initial subdivisions for each isentrope.

Example: isentrope initial intervals=18.

isentropes max depth=*⟨nonnegative integer⟩*

Default: 7

Maximum adaptive depth for isentropes.

Example: isentrope max depth=9.

isentropes tolerance=*⟨positive number⟩*

Default: 0.30

Adaptive tolerance for isentropes.

Example: isentrope tolerance=0.2.

isochore initial intervals= $\langle$ *positive integer* $\rangle$

Default: 16

Initial subdivisions for each isochore.

Example: isochore initial intervals=20.

isochore max depth= $\langle$ *nonnegative integer* $\rangle$

Default: 8

Maximum adaptive depth for isochores.

Example: isochore max depth=10.

isochore tolerance= $\langle$ *positive number* $\rangle$

Default: 0.28

Adaptive tolerance for isochores.

Example: isochore tolerance=0.18.

saturation pressure epsilon= $\langle$ *positive number* $\rangle$

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.

19.5 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.

curve style= $\langle$ *TikZ style* $\rangle$

Default: empty

Adds common drawing options to all emitted thermodynamic curves.

Example: curve style={opacity=.8}.

quality color= $\langle$ *colour* $\rangle$

Default: blue!60!black

Sets interior quality-curve colour.

Example: quality color=cyan!60!black.

quality boundary color= $\langle$ *colour* $\rangle$

Default: blue!80!black

Sets the saturated-liquid and saturated-vapour boundary colour.

Example: quality boundary color=navy.

isotherm color= $\langle$ *colour* $\rangle$

Default: red!65!black

Sets isotherm colour.

Example: isotherm color=magenta!70!black.

isentropes color= $\langle colour \rangle$

Default: green!50!black

Sets isentropes colour.

Example: isentropes color=green!50!black.

isochore color= $\langle colour \rangle$

Default: orange!80!black

Sets isochore colour.

Example: isochore color=brown.

interior style= $\langle TikZ style \rangle$

Default: line width=0.25pt

Styles interior quality curves.

Example: interior style={line width=.4pt,densely dashed}.

boundary style= $\langle TikZ style \rangle$

Default: line width=0.9pt

Styles the quality-envelope boundaries.

Example: boundary style={line width=1.2pt}.

isotherm style= $\langle TikZ style \rangle$

Default: line width=0.25pt

Styles all isotherms.

Example: isotherm style={line width=.35pt,dashed}.

isentropes style= $\langle TikZ style \rangle$

Default: dotted thin line

Styles all isentropes.

Example: isentropes style={line width=.35pt,densely dotted}.

isochore style= $\langle TikZ style \rangle$

Default: short dashed thin line

Styles all isochores.

Example: isochore style={line width=.35pt,dash dot}.

legend= $\langle boolean \rangle$

Default: false

Adds emitted curves to the PGFPlots legend when true.

Example: legend=true.

forget plot= $\langle boolean \rangle$

Default: true

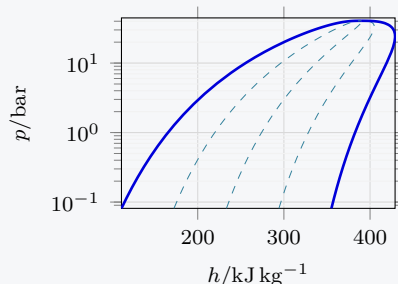
Applies PGFPlots forget plot; set false when constructing a custom legend.

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=4.1
    cm,
    font=\scriptsize}
]
```



19.6 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 PH 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`.

`quality symbol=`*TeX math material*

Default: Q

Sets the symbol preceding every generated quality value.

Example: `quality symbol=x`.

`temperature symbol=`*TeX math material*

Default: T

Sets the symbol preceding every generated isotherm value.

Example: `temperature symbol={\theta}`.

`entropy symbol=`*TeX math material*

Default: s

Sets the symbol preceding every generated isentrope value.

Example: `entropy symbol={\sigma}`.

`specific volume symbol=`*TeX math material*

Default: v

Sets the symbol preceding every generated isochore value.

Example: `specific volume symbol={\nu}`.

`labels=⟨boolean⟩` Default: false

Enables labels for eligible curve families.

Example: `labels=true`.

`label placement=⟨autonode⟩` Default: autonode

Selects autonomous placement. `autonode` is the supported placement engine.

Example: `label placement=autonode`.

`quality labels=⟨boolean or auto⟩` Default: auto

Overrides global label visibility for quality curves.

Example: `quality labels=true`.

`isotherm labels=⟨boolean or auto⟩` Default: auto

Overrides global label visibility for isotherms.

Example: `isotherm labels=true`.

`isentropes labels=⟨boolean or auto⟩` Default: auto

Overrides global label visibility for isentropes.

Example: `isentropes labels=false`.

`isochore labels=⟨boolean or auto⟩` Default: auto

Overrides global label visibility for isochores.

Example: `isochore labels=true`.

`label pos=⟨fraction⟩` Default: 0.35

Sets the common preferred curve position.

Example: `label pos=0.5`.

`quality label pos=⟨fraction⟩` Default: 0.35

Sets the quality-family preferred position.

Example: `quality label pos=0.3`.

`isotherm label pos=⟨fraction⟩` Default: 0.20

Sets the isotherm-family preferred position.

Example: `isotherm label pos=0.25`.

`isentropes label pos=⟨fraction⟩`

Default: 0.45

Sets the isentrope-family preferred position.

Example: `isentropes label pos=0.5`.

`isochore label pos=⟨fraction⟩`

Default: 0.58

Sets the isochore-family preferred position.

Example: `isochore label pos=0.6`.

`label every=⟨positive integer⟩`

Default: 2

Labels one curve out of every n by default.

Example: `label every=3`.

`quality label every=⟨positive integer⟩`

Default: 2

Sets the quality-family label stride.

Example: `quality label every=1`.

`isotherm label every=⟨positive integer⟩`

Default: 2

Sets the isotherm-family label stride.

Example: `isotherm label every=3`.

`isentropes label every=⟨positive integer⟩`

Default: 2

Sets the isentrope-family label stride.

Example: `isentropes label every=2`.

`isochore label every=⟨positive integer⟩`

Default: 2

Sets the isochore-family label stride.

Example: `isochore label every=2`.

`label sloped=⟨boolean⟩`

Default: false

Sets the common label-rotation policy.

Example: `label sloped=true`.

`quality label sloped=⟨boolean or auto⟩`

Default: auto

Overrides rotation for quality labels.

Example: `quality label sloped=false`.

`isotherm label sloped=<boolean or auto>`

Default: auto

Overrides rotation for isotherm labels.

Example: `isotherm label sloped=true.`

`isentropes label sloped=<boolean or auto>`

Default: auto

Overrides rotation for isentropes labels.

Example: `isentropes label sloped=true.`

`isochore label sloped=<boolean or auto>`

Default: auto

Overrides rotation for isochore labels.

Example: `isochore label sloped=true.`

`label allow upside down=<boolean>`

Default: false

Sets the common PGFPlots orientation policy.

Example: `label allow upside down=true.`

`quality label allow upside down=<boolean or auto>`

Default: auto

Overrides upside-down orientation for quality labels.

Example: `quality label allow upside down=false.`

`isotherm label allow upside down=<boolean or auto>`

Default: auto

Overrides upside-down orientation for isotherm labels.

Example: `isotherm label allow upside down=false.`

`isentropes label allow upside down=<boolean or auto>`

Default: auto

Overrides upside-down orientation for isentropes labels.

Example: `isentropes label allow upside down=true.`

`isochore label allow upside down=<boolean or auto>`

Default: auto

Overrides upside-down orientation for isochore labels.

Example: `isochore label allow upside down=false.`

`label style=<TikZ style name>`

Default: `luacoolprop label node`

Sets the common label-node style name.

Example: `label style=luacoolprop label node.`

quality label style= $\langle$ TikZ style name $\rangle$ Default: luacoolprop quality label node

Sets the quality-label style name.

Example: quality label style=luacoolprop quality label node.

isotherm label style= $\langle$ TikZ style name $\rangle$ Default: luacoolprop isotherm label node

Sets the isotherm-label style name.

Example: isotherm label style=luacoolprop isotherm label node.

isentropelabel style= $\langle$ TikZ style name $\rangle$ Default: luacoolprop isentropelabel node

Sets the isentropelabel style name.

Example: isentropelabel style=luacoolprop isentropelabel node.

isochore label style= $\langle$ TikZ style name $\rangle$ 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.

label node style= $\langle$ TikZ node options $\rangle$ Default: package style

Redefines luacoolprop label node.

Example: label node style={fill=yellow!15,inner sep=2pt}.

quality label node style= $\langle$ TikZ node options $\rangle$ Default: inherits common style

Redefines the quality label node.

Example: quality label node style={fill=blue!8,text=blue!70!black}.

isotherm label node style= $\langle$ TikZ node options $\rangle$ Default: inherits common style

Redefines the isotherm label node.

Example: isotherm label node style={fill=red!8,text=red!70!black}.

isentropelabel node style= $\langle$ TikZ node options $\rangle$ Default: inherits common style

Redefines the isentropelabel node.

Example: isentropelabel node style={fill=green!8,text=green!40!black}.

`isochore label node style= $\langle$ TikZ node options $\rangle$` 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`.

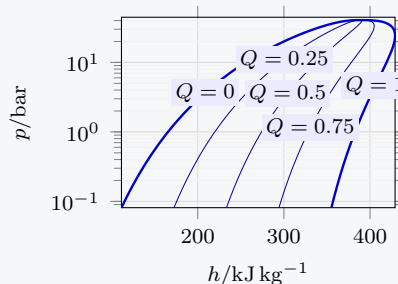
Accepted alias: `autonode labels forwards to label placement=autonode`.

Accepted alias: `pgfplots autonode labels forwards to label placement=autonode`.

Family label controls

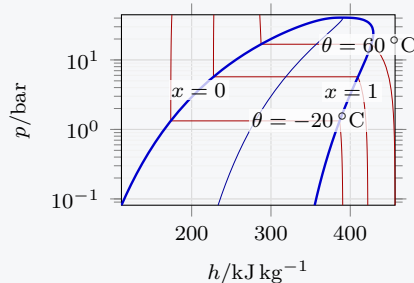
```
\LCPPHDiagram[
  quality values={0,0.25,0.5,0.75,1},
  labels=true,quality label every=1,
  quality label pos=.45,
  quality label node style={fill=blue!8,

  inner sep=1.5pt,font=\scriptsize},
  axis options={width=5.2cm,height=4.1
  cm,
  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=4.1
  cm,
  font=\scriptsize}
]
```



19.7 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.

`label pos for= $\langle$ selector and fraction $\rangle$` Default: none

Overrides one label's preferred position.

Example: `label pos for={lcp-ph-q-0p5}{0.62}`.

`label style for=<selector and style>`

Default: none

Overrides one label's TikZ style name or options.

Example: `label style for={lcp-ph-q-0p5}{fill=yellow}.`

`label text for=<selector and TeX text>`

Default: none

Replaces one automatically generated label.

Example: `label text for={lcp-ph-q-0p5}{ $q=50\%$ }.`

`label show for=<selector and boolean>`

Default: none

Forces one label to be shown or hidden.

Example: `label show for={lcp-ph-q-0p5}{false}.`

`label fixed for=<selector and boolean>`

Default: none

Marks one label position as fixed rather than preferred.

Example: `label fixed for={lcp-ph-q-0p5}{true}.`

`label sloped for=<selector and boolean>`

Default: none

Overrides rotation for one label.

Example: `label sloped for={lcp-ph-q-0p5}{true}.`

`label allow upside down for=<selector and boolean>`

Default: none

Overrides the orientation policy for one label.

Example: `label allow upside down for={lcp-ph-q-0p5}{false}.`

`curve style for=<selector and TikZ style>`

Default: none

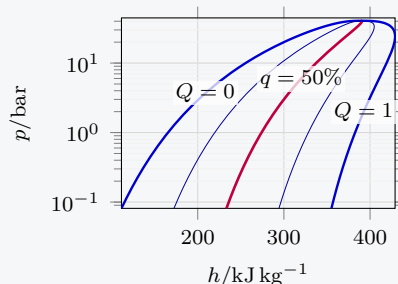
Overrides the drawing style for one curve.

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=50\%
\$},
  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=4.1
  cm,
  font=\scriptsize}
]
```



19.8 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.

autonode candidates=*⟨positive integer⟩*

Default: 51

Sets the number of sampled positions for each PH label.

Example: autonode candidates=31.

autonode candidate strategy=*⟨strategy⟩*

Default: around-preferred

Uses uniform, around-preferred, or adaptive candidate ordering.

Example: autonode candidate strategy=adaptive.

autonode clearance=*⟨TeX dimension⟩*

Default: 1pt

Adds collision clearance around PH labels.

Example: autonode clearance=2pt.

autonode inner sep=*⟨TeX dimension⟩*

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.

autonode normal shift=*⟨TeX dimension⟩*

Default: 0pt

Offsets PH labels along the local curve normal.

Example: autonode normal shift=4pt.

`autonode preferred weight=<number>`

Default: 8

Weights distance from the preferred position.

Example: `autonode preferred weight=12.`

`autonode overlap weight=<number>`

Default: 1000

Weights label overlap relative to preferred-position distance.

Example: `autonode overlap weight=2000.`

`label min pos=<fraction>`

Default: 0

Sets the earliest candidate position along each PH curve.

Example: `label min pos=0.12.`

`label max pos=<fraction>`

Default: 1

Sets the latest candidate position along each PH curve.

Example: `label max pos=0.88.`

19.9 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.

`label collision policy=<value>`

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`.

`label hide overlap threshold=<number>`

Default: 0.06

Accepted without solver effect; use the autonode failure policy.

Example: `label hide overlap threshold=0.08.`

`label hide edge threshold=<number>`

Default: 0.025

Accepted without solver effect; use auto node border margin.

Example: `label hide edge threshold=0.03.`

`label candidate count=<integer>`

Default: 21

Accepted without solver effect; use autonode candidates.

Example: `label candidate count=31.`

`label separation=<number>` Default: 0.018

Accepted without solver effect; use `autonode clearance`.

Example: `label separation=0.02`.

`label overlap penalty=<number>` Default: 1000

Accepted without solver effect; use `autonode overlap weight`.

Example: `label overlap penalty=1500`.

`label distance penalty=<number>` Default: 1

Accepted without solver effect; use `autonode preferred weight`.

Example: `label distance penalty=2`.

`label edge penalty=<number>` Default: 5

Accepted without solver effect; use `auto node border margin`.

Example: `label edge penalty=8`.

`label box base width=<number>` Default: 0.045

Accepted without solver effect; `autonode` measures the real TeX node.

Example: `label box base width=0.05`.

`label box char width=<number>` Default: 0.0060

Accepted without solver effect; `autonode` measures the real TeX node.

Example: `label box char width=0.007`.

`label box height=<number>` Default: 0.060

Accepted without solver effect; `autonode` measures the real TeX node.

Example: `label box height=0.07`.

`label box sloped multiplier=<number>` Default: 1.35

Accepted without solver effect; use `auto node bbox mode=oriented`.

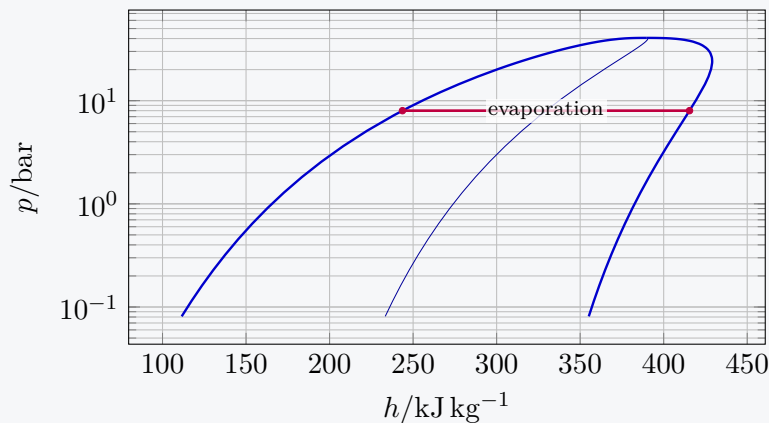
Example: `label box sloped multiplier=1.4`.

20 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={\mathstrut h/\mathrm{kJ}\,\mathrm{kg}^{-1}},ylabel={\mathstrut p/\mathrm{bar}},
grid=both]
\LCPPAddPHQuality[quality values={0,0.5,1}]
\LCPPAddPHProcess[type=isobar,pressure=8bar,
from={quality=0},to={quality=1},
color=purple,label={evaporation},mark endpoints=true]
\end{axis}
\end{tikzpicture}
```



20.1 Thermodynamic definition

fluid= $\langle$ *CoolProp identifier* $\rangle$

Default: global fluid

Selects the process fluid.

Example: fluid=R134a.

library= $\langle$ *file path* $\rangle$

Default: global library

Overrides the external library path for this process.

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

type= $\langle$ *process kind* $\rangle$

Default: isobar

Selects the property held constant along the path.

isobar Constant pressure.

isotherm Constant temperature.

isenthalp Constant specific enthalpy.

isentrop Constant specific entropy.

isochore Constant specific volume.

quality Constant vapour quality.

Example: type=isentropo.

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.

from=*⟨state key list⟩*

Default: empty

Defines the initial state on the process constraint.

Example: from={quality=0}.

to=*⟨state key list⟩*

Default: empty

Defines the final state on the process constraint.

Example: to={quality=1}.

value=*⟨quantity⟩*

Default: empty

Supplies the conserved property when its named key is not used. Unit parsing follows the selected process type.

Example: value=8bar.

pressure=*⟨pressure⟩*

Default: empty

Sets an isobar. A bare number is in pascals; accepted suffixes are Pa, kPa, MPa, bar, and mbar.

Example: pressure=8bar.

temperature=*⟨temperature⟩*

Default: empty

Sets an isotherm. A bare number is in kelvin; suffix C, degC, or celsius selects degrees Celsius.

Example: temperature=20C.

enthalpy=*⟨specific enthalpy⟩*

Default: empty

Sets an isenthalpic path. A bare number is in J/kg; kJkg selects kJ/kg.

Example: enthalpy=250kJkg.

entropy=*⟨specific entropy⟩*

Default: empty

Sets an isentropo. A bare number is in J/(kg K); kJkgK selects kJ/(kg K).

Example: entropy=1.8kJkgK.

quality=*<number>*

Default: empty

Sets constant vapour quality for a quality path.

Example: quality=0.5.

specific volume=*<specific volume>*

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 m ³ kg ⁻¹ , and L/kg or dm ³ /kg is also accepted.
rho, density, rhomass	rho=25kg/m3	Mass density; bare values use 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`.

20.2 Identity, appearance, and labels

name=*<identifier>* Default: empty

Assigns the stable TikZ `name` path selector used by labels, intersections, and higher-level documents.

Example: `name=evaporation`.

color=*<colour>* Default: black

Sets the process-curve colour.

Example: `color=purple`.

style=*<TikZ style>* 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.

label=*<TeX text>* Default: empty

Adds a label managed by `autonode`.

Example: `label={\$1\to 2\$}`.

label pos=*<fraction>* Default: 0.5

Sets the preferred label position along the process.

Example: `label pos=0.6`.

label style=*<TikZ style name>* Default: `luacoolprop process label node`

Selects the process-label node style.

Example: `label style=luacoolprop process label node`.

label node style=*<TikZ node options>* Default: `package style`

Redefines `luacoolprop process label node`.

Example: `label node style={fill=purple!8,text=purple}`.

`label sloped=<boolean>`

Default: true

Rotates the process label with the local tangent.

Example: `label sloped=false`.

`label allow upside down=<boolean>`

Default: false

Allows the label to retain an upside-down tangent orientation.

Example: `label allow upside down=true`.

`mark endpoints=<boolean>`

Default: false

Draws markers at the start and end states.

Example: `mark endpoints=true`.

`marker style=<TikZ plot style>`

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`.

20.3 Named intersections and numeric endpoint access

`export coordinates=<control-sequence prefix>`

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`.

`log coordinates=<boolean>`

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 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	J kg^{-1}	Mass-specific enthalpy.
EntropySI	$\text{J kg}^{-1} \text{K}^{-1}$	Mass-specific entropy.
SpecificVolumeSI	$\text{m}^3 \text{kg}^{-1}$	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]
\LCAddPHProcess[
  fluid=R717,
  type=isentropo,
  from={pressure=3bar,quality=1},to={pressure=10bar},
  name=cycle-ab,export coordinates=CycleAB,
  log coordinates=true]
\LCAddPHProcess[
  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.

20.4 Process scaling and sampling

enthalpy scale=*<number>* Default: global scale

Multiplies process enthalpies before plotting.

Example: enthalpy scale=0.001.

pressure scale=*<number>* Default: global scale

Multiplies process pressures before plotting.

Example: pressure scale=0.00001.

initial intervals=*<positive integer>* Default: 12

Sets initial process-path subdivisions.

Example: initial intervals=18.

max depth=*<nonnegative integer>* Default: 7

Sets maximum adaptive process depth.

Example: max depth=9.

tolerance=*<positive number>* Default: 0.25

Sets geometric process refinement tolerance.

Example: tolerance=0.15.

log weight=*<nonnegative number>* Default: 30

Weights logarithmic pressure differences during refinement.

Example: log weight=40.

coord digits=*<positive integer>* 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.

21 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 $\text{m}^3 \text{kg}^{-1}$ 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.

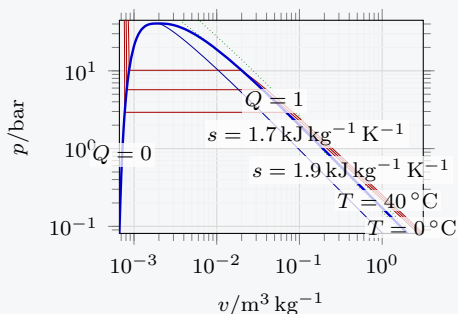
21.1 PV convenience commands

`\LCPVDiagram[<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

```
\LCPVDiagram[
  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={width=5.6cm,
    height=4.4cm,font=\scriptsize}
]
```



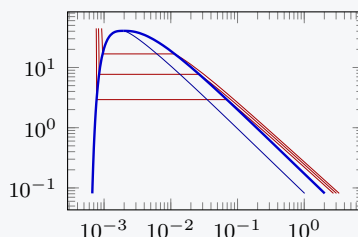
It is exactly equivalent to `\LCPDiagram{PV}[<options>]`.

`\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}
```

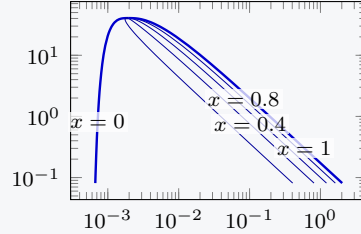


`\LCPAddPVQuality[\langle options \rangle]`

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}
```

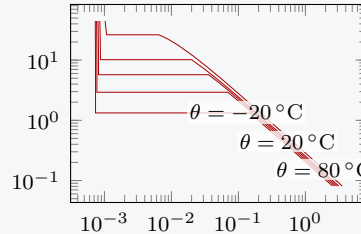


`\LCPAddPVIsotherms[\langle options \rangle]`

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}
```



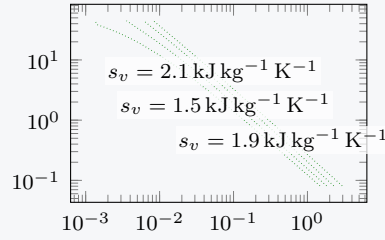
`\LCPAddPVIsentropes[\langle options \rangle]`

Adds only constant-specific-entropy curves. Entropy inputs follow `entropy unit`; the default is $\text{kJ kg}^{-1} \text{K}^{-1}$.

PV isentropes

```
\begin{tikzpicture}
\begin{loglogaxis}[width=5.5cm,auto
  node placement,
  height=4.2cm,font=\scriptsize]
\LCPPAddPVIsentropes[
  entropy values={1.5,1.7,1.9,2.1},

  entropy symbol={s_v},
  labels=true]
\end{loglogaxis}
\end{tikzpicture}
```

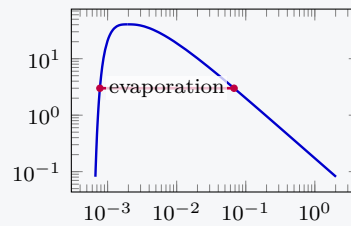


\LCPPAddPVProcess[*<options>*]

Adds one process in PV coordinates. State specifications, conserved-property inference, named paths, endpoint exports, styles, and diagnostics are identical to \LCPPAddPHPProcess; 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]
\LCPPAddPVQuality[
  quality values={0,1}]
\LCPPAddPVProcess[
  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:

PV through the generic registry

```
\LCPDiagram{PV}[isotherm=true]
\LCPPAddDiagramPlots{PV}[isentropes=true]
\LCPPAddDiagramFamily{PV}{quality}[quality step=0.1]
\LCPPAddDiagramProcess{PV}[
  type=isobar,pressure=3bar,
  from={quality=0},to={quality=1}]
```

21.2 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; the PH section is merely where some leaf-key entries appear earlier in this manual.

Group	Keys accepted without change in PV
Source and selection	<code>fluid</code> , <code>library</code> , <code>quality</code> , <code>isotherm</code> , <code>isentrop</code> , and their documented aliases.
Pressure	<code>pressure min</code> , <code>pressure max</code> , <code>pressure max factor</code> , <code>pressure scale</code> , and aliases.
Quality grid	Every <code>quality ...</code> grid key and every <code>q ...</code> alias: <code>mode</code> , <code>values</code> , <code>minimum</code> , <code>maximum</code> , <code>step</code> , <code>count</code> , and <code>preset</code> .
Temperature grid	Every <code>temperature ...</code> and <code>t ...</code> grid key, including <code>unit</code> , <code>mode</code> , <code>values</code> , <code>limits</code> , <code>step</code> , <code>count</code> , and <code>preset</code> .
Entropy grid	Every <code>entropy ...</code> and <code>s ...</code> grid key, including <code>unit</code> , <code>mode</code> , <code>values</code> , <code>limits</code> , <code>step</code> , <code>count</code> , and <code>preset</code> .
Symbols	<code>pressure symbol</code> , <code>specific volume symbol</code> , <code>quality symbol</code> , <code>temperature symbol</code> , and <code>entropy symbol</code> .
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 isentrop label keys; all autonode forwarding and per-curve override keys.
Sampling	<code>initial intervals</code> , <code>max depth</code> , <code>tolerance</code> , <code>log weight</code> , family-specific refinements, <code>saturation pressure epsilon</code> , and <code>coord digits</code> .

PV has no isochore background family: an explicit `isochore` switch is an error. To draw an isochoric transformation, use `\LCPAddPVProcess` with `type=isochore`.

`specific volume scale=`*<positive number>* 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`.

`specific volume axis unit=`*<m3kg or lkg>* Default: m3kg

Atomically selects the specific-volume conversion and horizontal label.

Example: `specific volume axis unit=lkg`.

`pressure axis unit=`*<pa, kpa, mpa, or bar>* Default: bar

Atomically selects the pressure conversion and vertical label.

Example: `pressure axis unit=kpa`.

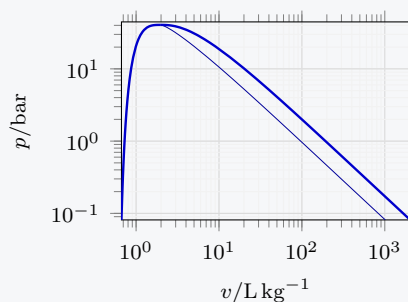
`log x weight=`*<nonnegative number>* Default: 30

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

```
\LCPVDiagram[
  specific volume axis unit=lkg,
  quality values={0,0.5,1},
  axis options={width=5.4cm,
    height=4.2cm,font=\scriptsize}
]
```



21.3 PV curve names, intersections, and overrides

Stable PV path names are separate from PH names:

Family	Pattern	Example
quality	<code>lcp-pv-q-⟨value⟩</code>	<code>lcp-pv-q-0p5</code>
isotherm	<code>lcp-pv-T-⟨value⟩-C</code>	<code>lcp-pv-T-20-C</code>
isentrop	<code>lcp-pv-s-⟨value⟩-kJkgK</code>	<code>lcp-pv-s-1p8-kJkgK</code>
process	<code>lcp-pv-process-⟨type⟩-...</code>	customisable with <code>name</code>

Consequently the common per-curve keys target PV curves by their PV name:

A PV-specific curve override

```
\LCPVDiagram[
  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=50\%},
  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.

21.4 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.

specific volume scale=*⟨positive number⟩* 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.

log x weight=*⟨nonnegative number⟩* 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 =  $v_{\text{from}}$  times specific volume scale` and `\StateABFromY =  $p_{\text{from}}$  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

```
\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.
```

21.5 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}
\LCPVDiagram[isotherm=true,isentrope=true]
\bye
```

ConTeXt MkIV

```
\usemodule[t][luacoolprop]
\pgfplotsset{compat=1.18}
\starttext
  \LCPVDiagram[isotherm=true,isentrope=true]
\stoptext
```

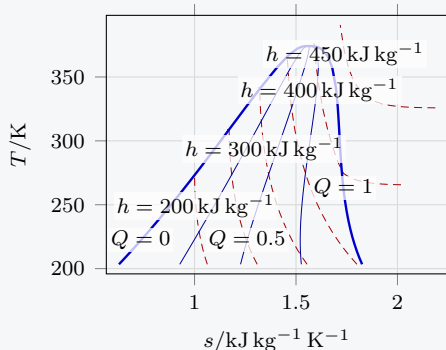
22 TS temperature–entropy diagrams

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 $\text{kJ kg}^{-1} \text{K}^{-1}$ 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=5.0
    cm,
    font=\scriptsize}
]
```



22.1 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{axis}[xlabel={s$},ylabel={T$}]
  \LCPAddTSPlots[fluid=R134a,isenthalp=true]
\end{axis}
```

`\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

```
\LCPAddTSQuality[fluid=R134a,  
  quality preset=boundaries]
```

`\LCPAddTSIsenthalps[<options>]`

Adds only constant mass-specific-enthalpy curves. Input values follow `enthalpy unit`; the default is `kJkg`.

Three explicit isenthalps

```
\LCPAddTSIsenthalps[fluid=R134a,  
  enthalpy values={250,300,350}]
```

`\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

```
\LCPAddTSProcess[fluid=R134a,type=isentropo,  
  from={p=2bar,Q=1},to={p=12bar}]
```

`\LCPAddTSIsoQuality[<options>]`

Alternate spelling for `\LCPAddTSQuality`; the latter is canonical.

Alternate spelling

```
\LCPAddTSIsoQuality[quality values={0,0.5,1}]
```

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.

22.2 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.

22.3 TS family, scale, and grid keys

isenthalp=*<boolean>*

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`.

entropy scale=*<number>*

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`.

temperature scale=*<number>*

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`.

`entropy axis unit=⟨jkgk or kjkgk⟩`

Default: `kjkgk`

Atomically selects the entropy conversion and horizontal label.

Example: `entropy axis unit=jkgk`.

`temperature axis unit=⟨kelvin or celsius⟩`

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`.

`enthalpy unit=⟨unit⟩`

Default: `kjkg`

Controls explicit enthalpy-grid input and generated labels. Use `kjkg` for kJ kg^{-1} or `si` for J kg^{-1} ; CoolProp always receives SI values.

Example: `enthalpy unit=si`.

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

`enthalpy min=⟨number or auto⟩`

Default: `auto`

Lower value used by an automatically generated isenthalp grid.

Example: `enthalpy min=200`.

`enthalpy max=⟨number or auto⟩`

Default: `auto`

Upper value used by an automatically generated isenthalp grid.

Example: `enthalpy max=450`.

`enthalpy step=⟨positive number or auto⟩`

Default: `auto`

Linear spacing. With `auto`, `enthalpy count` controls the number of curves.

Example: `enthalpy step=50`.

`enthalpy mode=⟨choice⟩`

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`.

`enthalpy count=⟨positive integer⟩`

Default: `8`

Number of linearly spaced automatic values when no step is supplied.

Example: `enthalpy count=6`.

`enthalpy values=<comma list>`

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`.

22.4 TS isenthalp appearance and labels

`isenthalp labels=<boolean or auto>`

Default: auto

Overrides the common `labels` switch for isenthalps.

Example: `isenthalp labels=true.`

`isenthalp label pos=<fraction>`

Default: 0.65

Preferred arc-length position forwarded to `pgfplots-autonode`.

Example: `isenthalp label pos=0.55.`

`isenthalp label every=<positive integer>`

Default: 2

Labels every *n*th isenthalp and always considers the last curve.

Example: `isenthalp label every=1.`

`isenthalp label sloped=<boolean or auto>`

Default: auto

Requests a label aligned with its local isenthalp tangent.

Example: `isenthalp label sloped=true.`

`isenthalp label allow upside down=<boolean or auto>`

Default: auto

Allows the automatic node to retain an upside-down tangent direction.

Example: `isenthalp label allow upside down=false.`

`isenthalp label style=<TikZ style name>`
node

Default: `luacoolprop isenthalp label node`

Selects the TikZ node style used for generated enthalpy labels.

Example: `isenthalp label style=luacoolprop label node.`

isenthalp label node style=*<TikZ options>* Default: package default

Redefines the dedicated luacoolprop isenthalp label node style.

Example: isenthalp label node style={fill=yellow!15,inner sep=1pt}.

isenthalp color=*<colour>* Default: red!65!black

Sets the base colour of constant-enthalpy curves.

Example: isenthalp color=orange!80!black.

isenthalp style=*<TikZ plot options>* 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.

22.5 TS sampling and axis bounds

isenthalp initial intervals=*<positive integer>* Default: 16

Initial logarithmic-pressure intervals before adaptive refinement.

Example: isenthalp initial intervals=24.

isenthalp max depth=*<nonnegative integer>* Default: 8

Maximum recursive bisection depth for an isenthalp.

Example: isenthalp max depth=10.

isenthalp tolerance=*<positive number>* Default: 0.10

Maximum scaled midpoint error accepted by isenthalp sampling.

Example: isenthalp tolerance=0.06.

temperature weight=*<positive number>* 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.

entropy axis min=*<number or auto>* Default: auto

Overrides the automatically padded horizontal lower bound in plotted entropy units.

Example: entropy axis min=0.8.

entropy axis max=*⟨number or auto⟩*

Default: auto

Overrides the automatically padded horizontal upper bound.

Example: entropy axis max=2.2.

temperature axis min=*⟨number or auto⟩*

Default: auto

Overrides the vertical lower bound after temperature scale.

Example: temperature axis min=220.

temperature axis max=*⟨number or auto⟩*

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.

22.6 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.

entropy scale=*⟨number⟩*

Default: 0.001

Multiplies process endpoint and path entropy coordinates.

Example: entropy scale=0.001.

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

temperature scale=*⟨number⟩*

Default: 1

Multiplies process endpoint and path temperature coordinates.

Example: temperature scale=1.

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

temperature weight=*⟨positive number⟩*

Default: 0.01

Controls the temperature contribution to adaptive TS process sampling.

Example: temperature weight=0.02.

Chain TS processes through exported SI entropy

```
\LCPAddTSProcess[type=isobar,pressure=2bar,
  from={Q=0},to={Q=1},export coordinates=EvapTS]
\LCPAddTSProcess[type=isentropo,
  from={p=2bar,Q=1},
  to={p=12bar,s=\EvapTSToEntropySI}]
```

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.

22.7 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.

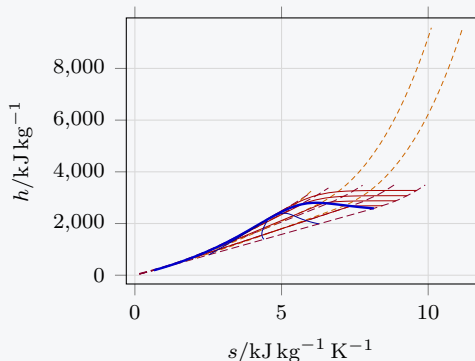
23 HS enthalpy–entropy diagrams

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 $\text{kJ kg}^{-1} \text{K}^{-1}$ and 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=5.1
    cm,
    font=\scriptsize}
]
```



23.1 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{axis}[xlabel={s},ylabel={h}]
  \LCPAddHSPlots[fluid=Water,isobar=true]
\end{axis}
```

\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

```
\LCPAddHSQuality[fluid=Water,  
quality values={0,1}]
```

\LCPAddHSIsochores[*<options>*]

Adds only constant-mass-specific-volume curves. Values and labels obey **specific volume unit**; the default is $\text{m}^3 \text{kg}^{-1}$.

Three Water isochores

```
\LCPAddHSIsochores[fluid=Water,  
specific volume values={0.01,0.1,1}]
```

\LCPAddHSIsotherms[*<options>*]

Adds only constant-temperature curves. Input and labels obey **temperature unit**, which defaults to degrees Celsius.

Selected Water isotherms

```
\LCPAddHSIsotherms[fluid=Water,  
temperature values={100,200,300}]
```

\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

```
\LCPAddHSIsobars[fluid=Water,  
isobar values={0.1,1,10,100}]
```

\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

```
\LCPAddHSProcess[fluid=Water,type=isentropo,  
  from={p=1bar,Q=1},to={p=10bar}]
```

`\LCPAddHSIsoQuality`[*<options>*]

Alternate spelling for `\LCPAddHSQuality`; the latter is canonical.

Alternate spelling

```
\LCPAddHSIsoQuality[quality values={0,0.5,1}]
```

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>]
```

23.2 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.

23.3 HS switches, scale, and error metric

`isobar=⟨boolean⟩`

Default: `false`

Enables the constant-pressure background family.

Example: `isobar=true`.

Accepted alias: `isobars` forwards to `isobar`.

`entropy scale=⟨number⟩`

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`.

`entropy axis unit=⟨jkgk or kjkgk⟩`

Default: `kjkgk`

Atomically selects the horizontal entropy conversion and label.

Example: `entropy axis unit=jkgk`.

`enthalpy axis unit=⟨jkg or kjkg⟩`

Default: `kjkg`

Atomically selects the vertical enthalpy conversion and label.

Example: `enthalpy axis unit=jkg`.

`enthalpy weight=⟨positive number⟩`

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.

23.4 Isobar grid and units

isobar unit=*<unit>*

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**.

isobar min=*<positive number or auto>*

Default: auto

Lower automatically generated pressure in **isobar unit**. Automatic means the diagram's lower pressure bound.

Example: **isobar min=0.1**.

isobar max=*<positive number or auto>*

Default: auto

Upper automatically generated pressure in **isobar unit**.

Example: **isobar max=250**.

isobar step=*<positive number or auto>*

Default: auto

Spacing used by a linear grid. With **auto**, **isobar count** determines the grid.

Example: **isobar step=10**.

isobar mode=*<choice>*

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**.

isobar count=*<positive integer>*

Default: 7

Number of automatically spaced isobars when no explicit step is given.

Example: **isobar count=9**.

isobar values=*<comma list>*

Default: empty

Explicit constant pressures in **isobar unit**.

Example: **isobar values={0.1,1,10,100,250}**.

isobar temperature min=*<number or auto>*

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**.

`isobar temperature max=⟨number or auto⟩`

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

```
\LCPAddHSIsobars[fluid=Water,  
  temperature unit=kelvin,  
  isobar values={1,10},  
  isobar temperature min=300,  
  isobar temperature max=700]
```

23.5 Isobar appearance and automatic labels

`isobar labels=⟨boolean or auto⟩`

Default: auto

Overrides the common labels switch for the isobar family.

Example: `isobar labels=true.`

`isobar label pos=⟨fraction⟩`

Default: 0.68

Preferred normalised arc-length position supplied to pgfplots-autonode.

Example: `isobar label pos=0.55.`

`isobar label every=⟨positive integer⟩`

Default: 2

Offers every n th isobar, and always the final isobar, for labelling.

Example: `isobar label every=1.`

`isobar label sloped=⟨boolean or auto⟩`

Default: auto

Aligns each generated pressure label with the local curve tangent.

Example: `isobar label sloped=true.`

`isobar label allow upside down=⟨boolean or auto⟩`

Default: auto

Permits the local tangent to leave an isobar label upside down.

Example: `isobar label allow upside down=false.`

`isobar label style=⟨TikZ style name⟩`

Default: `luacoolprop isobar label node`

Selects the node style attached to generated pressure labels.

Example: `isobar label style=luacoolprop label node.`

isobar label node style= $\langle$ *TikZ options* $\rangle$ 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}`.

isobar color= $\langle$ *colour* $\rangle$ Default: purple!70!black

Sets the base colour of every constant-pressure curve.

Example: `isobar color=violet!80!black`.

isobar style= $\langle$ *TikZ plot options* $\rangle$ 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

```
\LCPAddHSIsobars[pressure symbol=P,  
isobar values={1,10},labels=true]
```

Per-curve overrides use the stable isobar name; for example:

Emphasise the ten-bar path

```
\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}{ $p_{\rm boiler}$ }]
```

23.6 Isobar sampling and HS axis bounds

isobar initial intervals= $\langle$ *positive integer* $\rangle$ Default: 16

Initial entropy intervals before adaptive isobar refinement.

Example: `isobar initial intervals=24`.

isobar max depth= $\langle$ *nonnegative integer* $\rangle$ Default: 8

Maximum recursive midpoint-bisection depth for an isobar.

Example: `isobar max depth=10`.

isobar tolerance= $\langle$ *positive number* $\rangle$ Default: 0.10

Maximum scaled midpoint error accepted while sampling an isobar.

Example: `isobar tolerance=0.06`.

entropy axis min=*<number or auto>* Default: auto

Overrides the automatically padded horizontal minimum after **entropy scale**.

Example: entropy axis min=0.

entropy axis max=*<number or auto>* Default: auto

Overrides the horizontal maximum in plotted entropy units.

Example: entropy axis max=10.

enthalpy axis min=*<number or auto>* Default: auto

Overrides the automatically padded vertical minimum after **enthalpy scale**.

Example: enthalpy axis min=0.

enthalpy axis max=*<number or auto>* 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.

23.7 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.

entropy scale=*<number>* Default: 0.001

Multiplies entropy coordinates along an HS process path.

Example: entropy scale=0.001.

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

enthalpy weight=*<positive number>* Default: 0.01

Controls the vertical enthalpy contribution to adaptive HS process refinement.

Example: enthalpy weight=0.02.

Chain HS processes through exported entropy

```
\LCPAddHSProcess[fluid=Water,type=isobar,
  from={p=1bar,Q=0},to={p=1bar,Q=1},
  export coordinates=EvapHS,log coordinates=true]
```

```

\LCPSAddHSPProcess[fluid=Water,type=isentropo,
  from={p=1bar,Q=1},
  to={p=10bar,s=\EvapHSToEntropySI}]
% Plot coordinates: \EvapHSFromX, \EvapHSFromY,
%                  \EvapHSToX,   \EvapHSToY.
% Raw SI results:   \EvapHSToEntropySI, \EvapHSToEnthalpySI.

```

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.

23.8 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.

24 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.

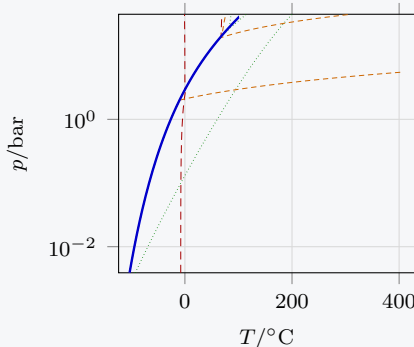
For a pure substance, saturated liquid and saturated vapour at equilibrium have the same T and p . Consequently, PT contains one coexistence locus, not separate bubble and dew curves and not a quality grid. The starting record uses CoolProp’s exact (T_t, p_t) constants; the final record uses the exact (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}
]
```



24.1 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

```
\LCPAddPTPhaseEnvelope[fluid=Propane,
  phase envelope color=blue,phase envelope style={very thick}]
```

Accepted alias: `\LCPAddPTSaturation`.

Target: `\LCPAddPTPhaseEnvelope`.

Example: `\LCPAddPTSaturation[fluid=R134a]`.

`\LCPAddPTIsentropes` [*<options>*]

Adds only constant-mass-specific-entropy curves.

Selected PT isentropes

```
\LCPAddPTIsentropes[entropy values={1.4,1.7,2.0}]
```

`\LCPAddPTIsenthalps` [*<options>*]

Adds only constant-mass-specific-enthalpy curves.

Selected PT isenthalps

```
\LCPAddPTIsenthalps[enthalpy values={200,300,400}]
```

`\LCPAddPTIsochores` [*<options>*]

Adds only constant-mass-specific-volume curves.

Selected PT isochores

```
\LCPAddPTIsochores[specific volume values={0.001,0.01,0.1}]
```

\LCPAddPTProcess[*<options>*]

Projects any of the six shared conserved-property process kinds onto PT.

An isentropic compression in PT coordinates

```
\LCPAddPTProcess[type=isentropo,  
  from={p=2bar,quality=1},to={p=10bar},  
  name=pt-compression,export coordinates=PTCompression]
```

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}.`

24.2 Families, ranges, and coordinate units

PT options use `/luacoolprop/diagram/PT`; the lowercase `/luacoolprop/pt` diagram path is an alternate public spelling.

fluid=*<CoolProp pure-fluid identifier>* Default: global fluid

Selects the pure fluid used by every PT family.

Example: fluid=Water.

library=*<file path>* Default: global library

Selects an external CoolProp shared library for this diagram.

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

reference state=*<DEF, IIR, ASHRAE, or NBP>* Default: DEF

Selects the locked enthalpy/entropy origin used by isenthalps and isentropes.

Example: reference state=DEF.

phase envelope=*<boolean>* 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.

isentrope=*<boolean>* Default: false

Enables constant-mass-specific-entropy curves.

Example: isentrope=true.

Accepted alias: isentropes forwards to isentrope.

isenthalp=*<boolean>* Default: false

Enables constant-mass-specific-enthalpy curves.

Example: isenthalp=true.

Accepted alias: isenthalps forwards to isenthalp.

Accepted alias: enthalpy curves forwards to isenthalp.

isochore=*<boolean>* Default: false

Enables constant-mass-specific-volume curves.

Example: isochore=true.

Accepted alias: isochores forwards to isochore.

pressure min=*<positive number or auto>* 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.

pressure max=*<positive number or auto>* Default: auto

Sets the axis maximum in Pa; automatic PT bounds use $1.10p_c$.

Example: pressure max=5E6.

pressure max factor=*<positive number>* Default: 1.10

Sets the automatic pressure maximum as a multiple of p_c .

Example: pressure max factor=1.25.

temperature axis min=*<number or auto>* Default: auto

Overrides the sampled horizontal minimum in the selected temperature axis unit.

Example: temperature axis min=-100.

`temperature axis max=⟨number or auto⟩` Default: auto

Overrides the sampled horizontal maximum in the selected temperature axis unit.

Example: `temperature axis max=150.`

`temperature axis unit=⟨kelvin or celsius⟩` Default: kelvin

Changes both horizontal coordinates and the axis label. Celsius applies $T_{\text{plot}} = T_{\text{SI}} - 273.15$, not a multiplicative approximation.

Example: `temperature axis unit=celsius.`

`pressure axis unit=⟨pa, kpa, mpa, or bar⟩` Default: bar

Changes both vertical coordinates and the generated pressure-axis label.

Example: `pressure axis unit=mpa.`

`temperature scale=⟨number⟩` 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`.

`pressure scale=⟨number⟩` 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`.

`temperature symbol=⟨TeX math material⟩` Default: T

Changes the horizontal-axis symbol without changing the CoolProp property.

Example: `temperature symbol={\theta }.`

`pressure symbol=⟨TeX math material⟩` Default: p

Changes the vertical-axis symbol.

Example: `pressure symbol={P}.`

`enthalpy symbol=⟨TeX math material⟩` Default: h

Changes generated isenthalp labels.

Example: `enthalpy symbol={i}.`

entropy symbol= $\langle TeX\ math\ material \rangle$

Default: s

Changes generated isentrope labels.

Example: entropy symbol={\sigma }.

specific volume symbol= $\langle TeX\ math\ material \rangle$

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 19: 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.

24.3 Enthalpy grid and isenthalp presentation

enthalpy unit= $\langle kjpg, jkg, or st \rangle$

Default: kjpg

Sets the unit used by the grid and generated labels.

Example: enthalpy unit=jkg.

Accepted alias: h unit forwards to enthalpy unit.

enthalpy min= $\langle number\ or\ auto \rangle$

Default: auto

Sets the lower automatic-grid bound.

Example: enthalpy min=200.

enthalpy max= $\langle number\ or\ auto \rangle$

Default: auto

Sets the upper automatic-grid bound.

Example: enthalpy max=450.

enthalpy step= $\langle positive\ number\ or\ auto \rangle$

Default: auto

Sets the linear grid spacing.

Example: enthalpy step=50.

enthalpy mode= $\langle linear\ or\ list \rangle$

Default: linear

Selects generated values; an explicit value list implies list mode.

Example: enthalpy mode=linear.

enthalpy count=*⟨positive integer⟩*

Default: 8

Sets the requested automatic count when no step is supplied.

Example: enthalpy count=6.

enthalpy values=*⟨numeric list⟩*

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.

isenthalp labels=*⟨boolean or auto⟩*

Default: auto

Overrides the shared labels switch for isenthalps.

Example: isenthalp labels=true.

isenthalp label pos=*⟨fraction⟩*

Default: 0.62

Sets the preferred autonode arc-length position.

Example: isenthalp label pos=0.7.

isenthalp label every=*⟨positive integer⟩*

Default: 2

Labels one isenthalp out of every n .

Example: isenthalp label every=1.

isenthalp label sloped=*⟨boolean or auto⟩*

Default: auto

Overrides the common rotation policy.

Example: isenthalp label sloped=true.

isenthalp label allow upside down=*⟨boolean or auto⟩*

Default: auto

Overrides the common tangent-orientation policy.

Example: isenthalp label allow upside down=false.

isenthalp label style=*⟨TikZ style⟩*

Default: luacoolprop isenthalp label node

Selects the generated label-node style.

Example: isenthalp label style=luacoolprop isenthalp label node.

`isenthalp label node style=<TikZ node options>` Default: package style

Redefines the PT isenthalp label style.

Example: `isenthalp label node style={fill=red!8,text=red!70!black}`.

`isenthalp color=<colour>` Default: red!65!black

Sets the isenthalp colour.

Example: `isenthalp color=purple`.

`isenthalp style=<TikZ plot options>` Default: thin, densely dashed

Sets the isenthalp drawing style.

Example: `isenthalp style={line width=.4pt,dashed}`.

24.4 Phase-envelope presentation and sampling

`phase envelope labels=<boolean or auto>` Default: auto

Overrides labels for the coexistence curve.

Example: `phase envelope labels=true`.

`phase envelope label text=<TeX text>` Default: liquid-vapour equilibrium

Sets the coexistence-curve label.

Example: `phase envelope label text={saturation}`.

`phase envelope label pos=<fraction>` Default: 0.45

Sets the preferred autonode position.

Example: `phase envelope label pos=0.6`.

`phase envelope label sloped=<boolean or auto>` Default: auto

Overrides label rotation.

Example: `phase envelope label sloped=true`.

`phase envelope label allow upside down=<boolean or auto>` Default: auto

Overrides tangent orientation.

Example: `phase envelope label allow upside down=false`.

`phase envelope label style= $\langle$ TikZ style $\rangle$` Default: `luacoolprop phase envelope label node`

Selects the label style.

Example: `phase envelope label style=luacoolprop phase envelope label node.`

`phase envelope label node style= $\langle$ TikZ node options $\rangle$` Default: `package style`

Redefines the label-node style.

Example: `phase envelope label node style={fill=blue!8,text=blue!70!black}.`

`phase envelope color= $\langle$ colour $\rangle$` Default: `blue!80!black`

Sets the coexistence-curve colour.

Example: `phase envelope color=black.`

`phase envelope style= $\langle$ TikZ plot options $\rangle$` Default: `line width=0.9pt`

Sets its drawing style.

Example: `phase envelope style={very thick}.`

`phase envelope initial intervals= $\langle$ positive integer $\rangle$` Default: `20`

Sets the initial logarithmic-pressure subdivision.

Example: `phase envelope initial intervals=28.`

`phase envelope max depth= $\langle$ nonnegative integer $\rangle$` Default: `9`

Limits recursive adaptive refinement.

Example: `phase envelope max depth=10.`

`phase envelope tolerance= $\langle$ positive number $\rangle$` 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.

24.5 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:

initial intervals=*⟨positive integer⟩*

Default: 18

Sets the common initial logarithmic-pressure subdivision.

Example: initial intervals=24.

max depth=*⟨nonnegative integer⟩*

Default: 8

Limits common adaptive recursion.

Example: max depth=9.

tolerance=*⟨positive number⟩*

Default: 0.10

Sets the common PT midpoint tolerance.

Example: tolerance=0.05.

log weight=*⟨nonnegative number⟩*

Default: 30

Weights logarithmic pressure in the midpoint metric.

Example: log weight=40.

domain policy=*⟨ignore, warning, or error⟩*

Default: ignore

Controls diagnostics for invalid CoolProp samples without reconnecting separate valid domains.

Example: domain policy=warning.

isenthalp initial intervals=*⟨positive integer⟩*

Default: 18

Overrides initial intervals for isenthalps.

Example: isenthalp initial intervals=24.

isenthalp max depth=*⟨nonnegative integer⟩*

Default: 8

Overrides recursion depth for isenthalps.

Example: isenthalp max depth=9.

isenthalp tolerance=*⟨positive number⟩*

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.

24.6 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.

temperature axis unit=*<kelvin or celsius>* Default: global unit

Couples process X coordinates to the PT axis; Celsius subtracts exactly 273.15 from absolute temperature.

Example: `temperature axis unit=celsius.`

pressure axis unit=*<pa, kpa, mpa, or bar>* Default: global unit

Couples process Y coordinates to the PT pressure axis.

Example: `pressure axis unit=bar.`

temperature scale=*<number>* 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.

24.7 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}
\LCPPPTDiagram[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
  \LCPPPTDiagram[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.

25 PGFPlots fluid style

`luacoolprop fluid=<CoolProp identifier>`

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{axis}[luacoolprop fluid=Propane,ymode=log]
  \LCPAddPHQuality
\end{axis}
```

Part IV

Developer reference

26 Architecture

The package is split into three layers:

File	Responsibility
<code>luacoolprop.lua</code>	FFI declarations, CoolProp wrappers, state resolution, adaptive sampling, curve-record construction, process validation, and PGFPlots code emission
<code>luacoolprop.tex</code>	generic TeX interface, <code>pgfkeys</code> trees, PGFPlots integration, wrappers around Lua calls
<code>luacoolprop.sty</code>	LaTeX wrapper loading PGFPlots and the generic TeX layer
<code>t-luacoolprop.tex</code>	ConTeXt wrapper loading the ConTeXt PGFPlots frontend and the generic TeX layer
<code>p-luacoolprop.tex</code>	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.

27 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.

28 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 .

29 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.

30 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`.

31 Lua API: scope, contracts, and safe use

The public Lua API intentionally separates three concerns. 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 only the first two layers.

The complete field-by-field reference is also maintained as `docs/luacoolprop-lua-api.md`. Public functions carry LDoc comments next to their implementations in `luacoolprop.lua`; the Markdown reference adds longer examples, extension rules, and a contributor checklist. The file `examples/lua-api-smoke.lua` executes the principal contracts through `texlua` as part of `scripts/build-examples.sh`.

31.1 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 _, loaded_from = lcp.load_library()
io.write("LuaCoolProp ", lcp._VERSION, "\n")
io.write("CoolProp ", lcp.version(), "\n")
io.write("CoolProp revision ", lcp.gitrevision(), "\n")
io.write("Loaded from ", loaded_from, "\n")
```

`M._VERSION` describes this package. In contrast, `M.version()` and `M.gitrevision()` query the loaded external CoolProp library. Reproducible reports should record all three values plus the result of `M.loaded_library()`.

The library search order is deterministic: an explicit path, then `LUACOOLOPPROP_LIB`, the secondary `COOLOPPROP_LIB`, platform names in the current directory, the active TeX resolver, and finally the operating system's dynamic loader. The first successful handle is cached. A later call with another path does not replace it; select the library before starting the engine when exact dependency control matters.

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.

31.2 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	<code>p</code>	Pa
temperature	<code>T</code>	K
mass enthalpy	<code>h</code>	J/kg
mass entropy	<code>s</code>	J/(kg K)
mass density	<code>rho</code>	kg/m ³
specific volume	<code>v</code>	m ³ /kg
vapor quality	<code>q</code>	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 m³/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.

31.3 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 CoolProp fluids reported as pure. 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 fluid, and process metadata records it. 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.

31.4 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. `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.

31.5 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.

31.6 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
<code>fluid</code>	requested fluid specification	—
<code>pcrit</code>	critical pressure	Pa
<code>ptriple</code>	triple-point pressure	Pa
<code>Tcrit</code>	critical temperature	K
<code>Ttriple</code>	triple-point temperature	K
<code>Tmin, Tmax</code>	model temperature bounds	K
<code>rhocrit</code>	critical mass density	kg/m ³

31.7 Structured diagram interface

The preferred accessors are `M.diagram.get_type("PH")`, `M.diagram.get_type("PV")`, and `M.diagram.get_type("TS")`, and `M.diagram.get_type("HS")`. They decouple application code from the lowercase storage fields `M.diagram.ph`, `M.diagram.pv`, `M.diagram.ts`, and `M.diagram.hs`. 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.

31.8 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 m³/kg.

31.9 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
<code>fluid</code>	"R134a"	CoolProp fluid name
<code>library</code>	search	explicit shared-library candidate
<code>reference_state</code>	"DEF"	locked DEF, IIR, ASHRAE, or NBP convention
<code>pressure_min</code>	fluid based	lower pressure in Pa
<code>pressure_max</code>	fluid based	upper pressure in Pa
<code>enthalpy_scale</code>	1e-3	multiplier from SI enthalpy to plot x
<code>pressure_scale</code>	1e-5	multiplier from SI pressure to plot y
<code>coord_digits</code>	6	serialization precision
<code>initial_intervals</code>	family based	initial log-pressure intervals
<code>max_depth</code>	family based	maximum adaptive recursion depth
<code>tolerance</code>	family based	midpoint error threshold
<code>domain_policy</code>	ignore	omitted-domain diagnostic
<code>log_weight</code>	30	pressure contribution to error
<code>quality</code>	true	include quality curves
<code>isotherm</code>	false	include isotherms
<code>isentropes</code>	false	include isentropes
<code>isochore</code>	false	include isochores
<code>labels</code>	false	request labels
<code>label_placement</code>	autonode	label backend selection

PH, PV, TS, HS, and PT are peer implementations. Their declared background families are:

- PH: quality, isotherm, isentropes, and isochore;
- PV: quality, isotherm, and isentropes;
- 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`, `isentropie_max_depth`, and `isochore_initial_intervals`. Selection fields include `quality_values`, `temperature_values`, `entropy_values`, and `specific_volume_values`. A Lua value list is currently 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,
  isentropie = 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.

31.10 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.

31.11 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.

31.12 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`.

32 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.

33 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.

Command Index

\LCPAddDiagramFamily, 72
\LCPAddDiagramPlots, 72
\LCPAddDiagramProcess, 73
\LCPAddHSIsobars, 121
\LCPAddHSIsochores, 121
\LCPAddHSIsoQuality, 122
\LCPAddHSIsotherms, 121
\LCPAddHSPlots, 120
\LCPAddHSProcess, 121
\LCPAddHSQuality, 120
\LCPAddPHIsentropes, 75
\LCPAddPhIsentropes (alias), 76
\LCPAddPHIsochores, 75
\LCPAddPhIsochores (alias), 76
\LCPAddPHIsoQuality (alias), 76
\LCPAddPhIsoQuality (alias), 76
\LCPAddPHIsotherms, 75
\LCPAddPhIsotherms (alias), 76
\LCPAddPHPlots, 75
\LCPAddPhPlots (alias), 76
\LCPAddPHProcess, 76
\LCPAddPhProcess (alias), 76
\LCPAddPHQuality, 75
\LCPAddPTIsenthalps, 130
\LCPAddPTIsentropes, 130
\LCPAddPTIsochores, 130
\LCPAddPTPhaseEnvelope, 130
\LCPAddPTPlots, 129
\LCPAddPTProcess, 131
\LCPAddPTSaturation (alias), 130
\LCPAddPVIsotherms, 107
\LCPAddPVPlots, 106
\LCPAddPVProcess, 108
\LCPAddPVQuality, 106
\LCPAddTSIsenthalps, 113
\LCPAddTSIsoQuality, 113
\LCPAddTSPlots, 112
\LCPAddTSProcess, 113
\LCPAddTSQuality, 112
\LCPDeclareFluid, 69
\LCPDiagram, 72
\LCPFluidName, 70
\LCPFluidPcritBar, 70
\LCPFluidPcritPlot, 70
\LCPFluidPcritSI, 70
\LCPFluidPtripleBar, 70
\LCPFluidPtripleSI, 70
\LCPFluidReferenceState, 70
\LCPFluidRhocritSI, 70
\LCPFluidTcritC, 70
\LCPFluidTcritK, 70
\LCPFluidTtripleC, 70
\LCPFluidTtripleK, 70
\LCPHSDiagram, 120
\LCPPHDiagram, 74
\LCPPhDiagram (alias), 76
\LCPPhIsoQualityDiagram (alias), 76
\LCPPTDiagram, 129
\LCPPVDiagram, 106
\LCPSetFluid, 68
\LCPSetLibrary, 68
\LCPSetReferenceState, 69
\LCPSetup, 68
\LCPTSDiagram, 112
\LCPUpdateFluidConstants, 69

Key and Value Index

automatic labels (alias), 94
autonode candidate strategy, 96
autonode candidate
 strategy=around-preferred, 96
autonode candidate strategy=uniform,
 96
autonode candidates, 96
autonode clearance, 96
autonode inner sep, 96
autonode labels (alias), 94
autonode normal shift, 96
autonode overlap weight, 97
autonode preferred weight, 96
axis options, 78

boundary color (alias), 88
boundary style, 88

color, 102
coord digits, 86, 105
curve style, 87
curve style for, 95

discontinuity policy (alias), 74
domain policy, 74, 138

enthalpy, 100
enthalpy axis max, 127
enthalpy axis min, 127
enthalpy axis unit, 70, 78, 123
enthalpy count, 115, 134
enthalpy curves (alias), 114, 132
enthalpy max, 115, 134
enthalpy min, 115, 134
enthalpy mode, 115, 134
enthalpy scale, 71, 78, 105
enthalpy step, 115, 134
enthalpy symbol, 78, 133
enthalpy unit, 115, 134
enthalpy values, 115, 135
enthalpy weight, 123, 127
entropy, 100
entropy axis max, 117, 127
entropy axis min, 117, 126
entropy axis unit, 71, 114, 123
entropy count, 82
entropy max, 82
entropy min, 82
entropy mode, 83
entropy mode=auto, 83
entropy mode=explicit, 83
entropy mode=linear, 83
entropy mode=list, 83
entropy mode=log, 83
entropy mode=logarithmic, 83
entropy mode=predefined, 83
entropy mode=preset, 83
entropy preset, 83
entropy preset=default, 83
entropy preset=dense, 83
entropy preset=refrigeration, 83
entropy preset=sparse, 83
entropy scale, 71, 114, 118, 123, 127
entropy step, 82
entropy symbol, 89, 133
entropy unit, 82
entropy unit=j/kg/k, 82
entropy unit=jkgk, 82
entropy unit=kjkgk, 82
entropy unit=si, 82
entropy values, 83
equilibrium curve (alias), 132
export coordinates, 103

fluid, 70, 77, 99, 131
forget plot, 88
from, 100

h (alias), 101
h count (alias), 116, 135
h max (alias), 116, 135
h min (alias), 116, 135
h mode (alias), 116, 135
h scale (alias), 72, 79, 105
h step (alias), 116, 135
h unit (alias), 115, 134
h values (alias), 116, 135

id (alias), 103
initial intervals, 85, 105, 137
interior style, 88
isenthalp, 114, 132
isenthalp color, 117, 136
isenthalp initial intervals, 117, 138
isenthalp label allow upside down,
 116, 135
isenthalp label every, 116, 135
isenthalp label node style, 116, 136
isenthalp label pos, 116, 135
isenthalp label sloped, 116, 135
isenthalp label style, 116, 135
isenthalp labels, 116, 135

- isenthalp max depth, 117, 138
- isenthalp style, 117, 136
- isenthalp tolerance, 117, 138
- isenthalps (alias), 114, 132
- isentropes, 77, 132
- isentropes color, 87
- isentropes initial intervals, 86
- isentropes label allow upside down, 92
- isentropes label every, 91
- isentropes label node style, 93
- isentropes label pos, 90
- isentropes label sloped, 92
- isentropes label style, 93
- isentropes labels, 90
- isentropes max depth, 86
- isentropes style, 88
- isentropes tolerance, 86
- isentropes (alias), 79, 132
- iso quality (alias), 78
- isobar, 123
- isobar color, 126
- isobar count, 124
- isobar initial intervals, 126
- isobar label allow upside down, 125
- isobar label every, 125
- isobar label node style, 125
- isobar label pos, 125
- isobar label sloped, 125
- isobar label style, 125
- isobar labels, 125
- isobar max, 124
- isobar max depth, 126
- isobar min, 124
- isobar mode, 124
- isobar step, 124
- isobar style, 126
- isobar temperature max, 124
- isobar temperature min, 124
- isobar tolerance, 126
- isobar unit, 124
- isobar values, 124
- isobars (alias), 123
- isochore, 77, 132
- isochore color, 88
- isochore initial intervals, 86
- isochore label allow upside down, 92
- isochore label every, 91
- isochore label node style, 93
- isochore label pos, 91
- isochore label sloped, 92
- isochore label style, 93
- isochore labels, 90
- isochore max depth, 87
- isochore style, 88
- isochore tolerance, 87
- isochores (alias), 79, 132
- isoquality (alias), 79
- isotherm, 77
- isotherm color, 87
- isotherm initial intervals, 86
- isotherm label allow upside down, 92
- isotherm label every, 91
- isotherm label node style, 93
- isotherm label pos, 90
- isotherm label sloped, 91
- isotherm label style, 93
- isotherm labels, 90
- isotherm max depth, 86
- isotherm style, 88
- isotherm tolerance, 86
- isotherms (alias), 79
- kind (alias), 101
- label, 102
- label allow upside down, 92, 103
- label allow upside down for, 95
- label box base width, 98
- label box char width, 98
- label box height, 98
- label box sloped multiplier, 98
- label candidate count, 97
- label collision (alias), 97
- label collision policy, 97
- label distance penalty, 98
- label edge penalty, 98
- label every, 91
- label fixed for, 95
- label hide edge threshold, 97
- label hide overlap threshold, 97
- label max pos, 97
- label min pos, 97
- label node style, 93, 102
- label overlap penalty, 98
- label placement, 90
- label pos, 90, 102
- label pos for, 94
- label separation, 97
- label show for, 95
- label sloped, 91, 102
- label sloped for, 95
- label style, 92, 102
- label style for, 94
- label text (alias), 103
- label text for, 95

- labels, 89
- lcp fluid (alias), 140
- legend, 88
- library, 70, 77, 99, 131
- log coordinates, 103
- log weight, 86, 105, 138
- log x weight, 109, 111
- luacoolprop fluid, 140

- mark endpoints, 103
- marker style, 103
- markers (alias), 103
- max depth, 85, 105, 138

- name, 102

- p (alias), 101
- p max (alias), 79
- p max factor (alias), 79
- p min (alias), 79
- p scale (alias), 72, 79, 105, 133
- pgfplots autonode labels (alias), 94
- phase envelope, 131
- phase envelope color, 137
- phase envelope initial intervals, 137
- phase envelope label allow upside down, 136
- phase envelope label node style, 137
- phase envelope label pos, 136
- phase envelope label sloped, 136
- phase envelope label style, 136
- phase envelope label text, 136
- phase envelope labels, 136
- phase envelope max depth, 137
- phase envelope style, 137
- phase envelope tolerance, 137
- pressure, 100
- pressure axis unit, 70, 78, 109, 133, 139
- pressure max, 77, 132
- pressure max factor, 77, 132
- pressure min, 77, 132
- pressure scale, 71, 78, 105, 133
- pressure symbol, 78, 133
- process (alias), 101
- process color (alias), 103
- process style (alias), 103

- Q (alias), 101
- q (alias), 101
- q count (alias), 80
- q max (alias), 80
- q min (alias), 80
- q mode (alias), 80
- q preset (alias), 80
- q step (alias), 80
- q values (alias), 80
- quality, 77, 100
- quality boundary color, 87
- quality color, 87
- quality count, 79
- quality curves (alias), 79
- quality label allow upside down, 92
- quality label every, 91
- quality label node style, 93
- quality label pos, 90
- quality label sloped, 91
- quality label style, 92
- quality labels, 90
- quality max, 79
- quality min, 79
- quality mode, 80
- quality mode=auto, 80
- quality mode=explicit, 80
- quality mode=linear, 80
- quality mode=list, 80
- quality mode=log, 80
- quality mode=predefined, 80
- quality mode=preset, 80
- quality preset, 80
- quality preset=boundaries, 80
- quality preset=boundary, 80
- quality preset=default, 80
- quality preset=dense, 80
- quality preset=sparse, 80
- quality step, 79
- quality symbol, 89
- quality values, 80

- reference state, 70, 74, 131

- s (alias), 101
- s count (alias), 83
- s max (alias), 83
- s min (alias), 83
- s mode (alias), 83
- s preset (alias), 83
- s scale (alias), 72, 114, 118, 123, 127
- s step (alias), 83
- s values (alias), 83
- saturation (alias), 132
- saturation pressure epsilon, 87
- sloped label (alias), 103
- sloped labels (alias), 94
- smart labels (alias), 94
- specific volume, 101
- specific volume axis unit, 70, 109

specific volume count, 84
 specific volume max, 83
 specific volume min, 83
 specific volume mode, 84
 specific volume mode=auto, 84
 specific volume mode=explicit, 84
 specific volume mode=linear, 84
 specific volume mode=list, 84
 specific volume mode=log, 84
 specific volume mode=logarithmic, 84
 specific volume mode=predefined, 84
 specific volume mode=preset, 84
 specific volume preset, 84
 specific volume preset=default, 84
 specific volume preset=dense, 84
 specific volume preset=refrigeration, 84
 specific volume preset=sparse, 84
 specific volume scale, 71, 109, 110
 specific volume step, 84
 specific volume symbol, 89, 134
 specific volume unit, 83
 specific volume unit=dm³/kg, 83
 specific volume unit=dm³kg, 83
 specific volume unit=l/kg, 83
 specific volume unit=lkg, 83
 specific volume unit=m³kg, 83
 specific volume values, 84
 start (alias), 101
 stop (alias), 101
 style, 102

 T (alias), 101
 t (alias), 101
 t count (alias), 82
 t max (alias), 82
 t min (alias), 82
 t mode (alias), 82
 t preset (alias), 82
 t scale (alias), 72, 114, 118, 133, 139
 t step (alias), 82
 t values (alias), 82
 temperature, 100
 temperature axis max, 118, 132
 temperature axis min, 118, 132
 temperature axis unit, 71, 115, 133, 139
 temperature count, 81
 temperature max, 81
 temperature min, 81
 temperature mode, 82
 temperature mode=auto, 82
 temperature mode=explicit, 82
 temperature mode=linear, 82
 temperature mode=list, 82
 temperature mode=log, 82
 temperature mode=logarithmic, 82
 temperature mode=predefined, 82
 temperature mode=preset, 82
 temperature preset, 82
 temperature preset=default, 82
 temperature preset=dense, 82
 temperature preset=refrigeration, 82
 temperature preset=sparse, 82
 temperature scale, 71, 114, 118, 133, 139
 temperature step, 81
 temperature symbol, 89, 133
 temperature unit, 81
 temperature unit=c, 81
 temperature unit=celsius, 81
 temperature unit=degc, 81
 temperature unit=k, 81
 temperature unit=kelvin, 81
 temperature values, 81
 temperature weight, 117, 118
 to, 100
 tolerance, 85, 105, 138
 type, 99
 type=enthalpy, 100
 type=entropy, 100
 type=h, 100
 type=isenthalp, 99
 type=isenthalpic, 100
 type=isentrop, 99
 type=isentropic, 100
 type=isobar, 99
 type=isobaric, 100
 type=isochore, 99
 type=isochoric, 100
 type=isoquality, 100
 type=isotherm, 99
 type=isothermal, 100
 type=p, 100
 type=pressure, 100
 type=q, 100
 type=quality, 99
 type=s, 100
 type=t, 100
 type=temperature, 100
 type=throttle, 100
 type=v, 100
 type=volume, 100

 v (alias), 101
 v count (alias), 84

`v max` (alias), 84
`v min` (alias), 84
`v mode` (alias), 84
`v preset` (alias), 84
`v scale` (alias), 72, 109, 111

`v step` (alias), 84
`v unit` (alias), 84
`v values` (alias), 84
`value`, 100
`volume unit` (alias), 84