The numodel package*

Paul Zuurbier
mail@paulzuurbier.nl

May 23, 2026

Abstract

A LuaLaTeX package for writing and rendering numerical models (Euler-integrated dynamical systems) directly inside LaTeX documents, aimed at physics teaching material. It provides a text-model pipeline (`\mvar`, `\mrule`, `\computemodel`), Forrester stock-and-flow diagrams, and optional plots of the computed time series via the sibling package numodel-plot.

Contents

1 Introduction

numodel lets an author write a dynamical system (stocks, flows, helper variables, rules, and a stop condition) as a sequence of LaTeX macros and renders three complementary views of that model directly in the document:

- a *text model* — a typeset rule table with initial values (`\textmodel`);

*This document corresponds to numodel v0.5.0, dated May 23, 2026.

- a *graphic model* — a Forrester stock-and-flow diagram with auto-layout (`\graphicmodel`);
- a *diagram* — a numerical Euler simulation plus PGFPlot of any pair of variables (`\computemodel` followed by `\diagrammodel`; the plot is rendered through the sibling package numodel-plot).

All three views are produced from a single set of declarations so the textbook description, the conceptual stock-and-flow diagram, and the numerical result of the same model are guaranteed to stay in sync. Variables and rules live in namespaces (*prefixes*) so a document can contain multiple independent models; `\newmodelprefix{P}` starts a fresh one. The simulation engine runs in Lua (through luacode) for $\mathcal{O}(1)$ appends and cheap min/max tracking; the rendering layer is pure expl3.

2 First example: a free-falling ball

A ball dropped from $h_0 = 100\,\mathrm{m}$ under constant gravitational acceleration. The complete model:

```
\usepackage[syntax=EN]{numodel}

\newmodelprefix{ball}
\mvar{T}{t}{0}{\s}{2}{system}
\mvar{Dt}{dt}{0.1}{\s}{2}{system}
\mvar{V}{v}{0}{\m\per\s}{2}{stock}
\mvar{Y}{y}{100}{\m}{3}{stock}
\mvar{G}{g}{-9.81}{\m\per\s\squared}{3}{aux}

\mrule{V}{\ballV + \ballG * \ballDt}
\mrule{Y}{\ballY + \ballV * \ballDt}
\mrule{T}{\ballT + \ballDt}
\mstop{\ballY <= 0}
```

After the declarations above, three render commands produce the three views shown below, each from the *same* model.

2.1 `\textmodel` — rule table

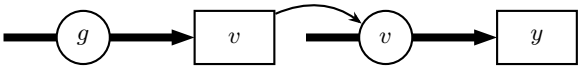
The verbatim source rendered by `\textmodel`:

	model	**initial values**
1	$v \;=\; v + g \cdot dt$	$t = 0\,\mathrm{s}$
2	$y \;=\; y + v \cdot dt$	$dt = 0.10\,\mathrm{s}$
3	$t \;=\; t + dt$	$v = 0\,\mathrm{m\,s^{-1}}$
4	IF $y \leqslant 0$ THEN STOP ENDIF	$y = 100\,\mathrm{m}$
		$g = -9.81\,\mathrm{m\,s^{-2}}$

Each `\mvar` with a non-empty start value contributes a row in the *initial values* column; each `\mrule` contributes a row in the *model* column. Symbols come from the second `\mvar` argument (the display text), values are formatted through siunitx. `<=` is rendered as $\leqslant$.

2.2 `\graphicmodel` — Forrester diagram

`\graphicmodel` draws the same model as a stock-and-flow diagram. Stocks (type `stock`) are rectangles, constants (`constant`) are circles, helpers (`aux`) are unboxed identifiers; flows are inferred from rule structure (`\<stock> + ...` or `... + \<stock>` on the right-hand side):



Layout is automatic from the rule graph. Manual placement is available through `gridx`/`gridy` keys on the `\mvar` call (see Section 4). Wide diagrams can be capped to a maximum number of grid columns with `\numodelsetup{gridmaxx=N}`: when a row reaches `N`, the auto layout shifts the affected items up one row and continues filling.

2.3 `\computemodel` + `\diagrammodel` — numerical plot

`\computemodel` iterates the rules forward in time using Euler integration with step size `\ballDt`, stopping when `\mstop`'s condition becomes true (or when the `maxiter` safety limit is reached, see Section 3). `\diagrammodel{xvar}{yvar}{label}` then plots one variable against another:

```
\computemodel
\diagrammodel{T}{Y}{ball-fall}
```

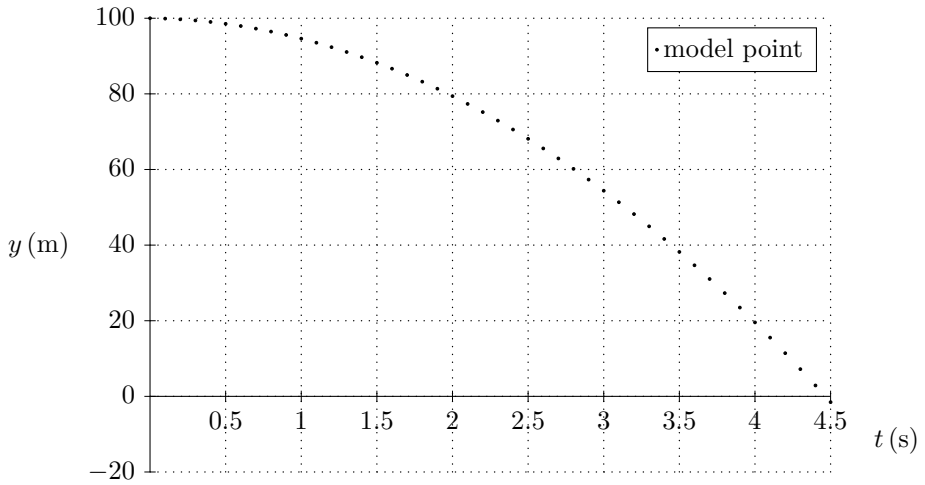


Figure 1: $y(t)$-diagram

Axis ranges, tick lattice, and labels are computed automatically from the simulated min/max of each variable; the plot inherits the numodel-plot style (see that package's documentation for configuration).

3 Configuration

`\numodelsetup` Runtime configuration:

```
\numodelsetup{syntax=NL, maxiter=50000}
```

The same keys can also be passed as package options: `\usepackage[syntax=NL]{numodel}`. Recognised keys:

`syntax` Language tag for the rule-table rendering. Built-in values:

> **`EN`** (default) XMILE-style ALL-CAPS keywords: `IF`/`THEN`/`ELSE`, `AND`, `OR`, `ABS`, `SIGN`, …
>
> **`NL`** Dutch CoachTaal keywords: `Als`/`Dan`/`Anders`, `EN`, `OF`, `Abs`, `Teken`, …
>
> The legacy names `english`, `coachtaal` and `dutch` are accepted as aliases for `EN` and `NL`. Each language tag `X` corresponds to a file `numodel-X.def` located via `kpse` when the package processes the key. The package ships with `numodel-EN.def` and `numodel-NL.def`; drop your own `numodel-FR.def` (or any other tag) in `TEXMFHOME/tex/latex/numodel/` and select it with `\usepackage[syntax=FR]{numodel}` – no package rebuild needed. The setting affects display only; the expression syntax in `\mrule` bodies is always `\fp_eval`-compatible.

`maxiter` Safety limit on the number of `\computemodel` iterations (default 20 000). When reached, the simulation aborts with a warning naming the unmet stop condition.

`graphscalex` Horizontal grid spacing in centimetres for `\graphicmodel`'s Forrester layout (default 2). Larger values spread the diagram out horizontally.

`graphscaley` Vertical grid spacing in centimetres for `\graphicmodel`'s Forrester layout (default 2). Larger values spread the diagram out vertically.

stockwidth Half-width of stock rectangles in `\graphicmodel` (default 0.375).

gridmaxx Maximum number of grid columns the auto layout may fill on any one row before wrapping (integer, default `0` = no limit). When the limit is reached, items already placed on the affected row (and everything above it for the stocks row, everything but stocks for the aux row, only the constants for the constants row) shift up by one row to free space, and placement continues from column 0. Manually positioned variables (`\mvar[gridx=...,gridy=...]`) are kept where they are. When wrapping is active the default centring of the aux row and the right-aligning of the stocks row are disabled, so the diagram fills left-to-right, bottom-to-top.

diagram-style Rendering style for the case where a helper or constant is the direct inflow/outflow of a stock. Three values:

- **tight** (default) the valve takes the helper's/constant's label; the helper/constant itself is not drawn as a separate node. Compact and most LaTeX-native.
- **forrester** Forrester/Sterman convention: the valve is drawn without a label and the helper/constant remains as a separate node connected to the valve by a causal arrow.
- **edu** Didactic dual form: the valve carries the label *and* the helper/constant is drawn as a separate node with a causal arrow to the valve. Visually busy but pedagogically explicit.

flowarrow-style Visual style of the flow pipe. `hollow` renders the classic Forrester double-line pipe with an open arrow head; `filled` renders a thick solid arrow. The default tracks `diagram-style`: `forrester` picks `hollow`, the other styles pick `filled`. An explicit value overrides this coupling.

valve-style Visual style of the valve node. `valve` draws the bow-tie/butterfly icon (Forrester); `circle` draws an empty circle on the flow pipe; `edu` draws a labelled circle (the flow variable's display text inside). The default tracks `diagram-style`: `forrester` picks `valve`, the other styles pick `edu`.

flowarrow-cloud-tip Whether the open end of an inflow or outflow pipe is anchored to a cloud node, signalling the model boundary. Default tracks `diagram-style`: `forrester` picks `true`, the other styles pick `false`. May be set globally via `\numodelsetup`, per-render via `\graphicmodel`, or per-stock via `\mvar[flowarrow-cloud-tip=...]`. The most specific source wins.

units Whether the *initial values* cells in `\textmodel` display the SI unit alongside the value (`\qty`) or only the numeric value (`\num`). Boolean, default `true`. May also be supplied to `\textmodel[units=false]` as a per-table override; the global setting is restored after rendering.

tblrenv Which tabularray environment wraps the `\textmodel` table. Three values: `longtblr` (default, page-breakable); `tblr` (inline, no page breaks); `talltblr` (inline, no page breaks, supports `\caption`/notes). Pick `tblr` or `talltblr` when `\textmodel` sits inside an enclosing environment that suppresses page breaks (`subfigure`, `minipage`, …); the default `longtblr` would otherwise emit spurious "(Continued)" / *Continued on next page* markers in that setting. May also be supplied per render as `\textmodel[tblrenv=...]`; the global setting is restored afterwards.

decimal-separator Decimal mark used by every number that `numodel` renders: the *initial values* column of `\textmodel`, and the tick labels of `\diagrammodel`. Two values:

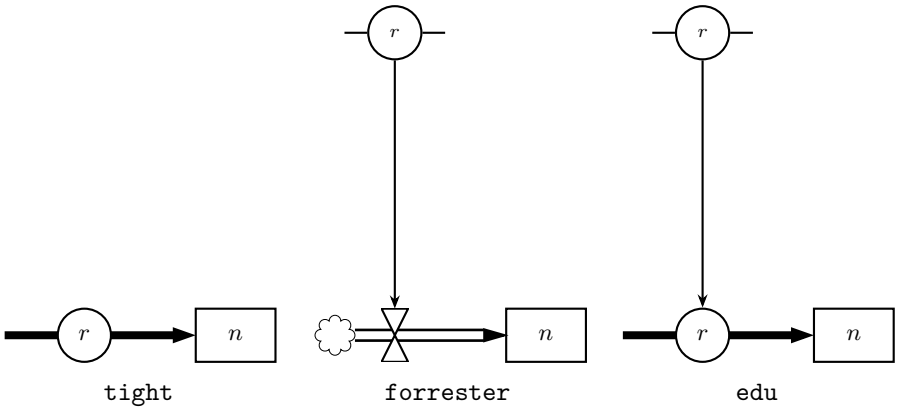
- **comma** use a comma (siunitx `output-decimal-marker={,}`, pgfplots `/pgf/number format/use comma`).
- **point** use a full stop (siunitx `output-decimal-marker={.}`, pgfplots `/pgf/number format/use period`).

The default tracks `syntax`: `NL` picks `comma`, `EN` picks `point`. Other language files can publish a default by defining `\__numodel_kw_<LANG>_dsep_default:` (expanding to `point` or `comma`); when the macro is absent the default is `point`. An explicit `decimal-separator` key locks that choice and overrides any future `syntax` change. The override is scoped: `numodel` applies it only inside its own renderers (siunitx's state is restored on group exit), so a document-wide `\sisetup` is not perturbed.

3.1 Diagram styles in practice

The same model rendered under each of the three `diagram-style` values. The model is the simplest case that distinguishes the styles: one stock N with a constant inflow R:

```
\newmodelprefix{flux}
\mvar{T}{t}{0}{\s}{2}{system}
\mvar{Dt}{dt}{1}{\s}{2}{system}
\mvar{N}{n}{0}{}{0}{stock}
\mvar{R}{r}{5}{\per\s}{2}{constant}
\mrule{N}{\fluxN + \fluxR * \fluxDt}
\mrule{T}{\fluxT + \fluxDt}
\mstop{\fluxT >= 5}
\graphicmodel[diagram-style=tight]
\graphicmodel[diagram-style=forrester]
\graphicmodel[diagram-style=edu]
```



- `tight` collapses the constant R into the valve label, producing the most compact diagram.
- `forrester` keeps the canonical System-Dynamics convention: unlabelled bow-tie valve, the constant remains a separate node, the link from R to the valve is a thin causal arrow.
- `edu` is a didactic dual: the valve carries the label *and* the constant remains as a separate node with a causal arrow. Less compact but pedagogically explicit – useful when first introducing the stock/flow vocabulary.

4 Public API

4.1 Variables and rules

`\mvar` Declares a model variable. Signature:

```
\mvar[<keys>]{<Name>}{<text>}{<start>}{<unit>}{<sig>}{<type>}
```

where ⟨*Name*⟩ is a short alphabetic identifier (the prefix-qualified accessor becomes `\<prefix><Name>`), ⟨*text*⟩ is the math-mode display symbol used in the rule table and diagram (e.g. `F_{res}`), ⟨*start*⟩ is the initial value (a number, or empty for helpers computed by a rule, or any `expl3` `fp`-evaluable expression involving previously defined model variables), ⟨*unit*⟩ is a bare `siunitx` unit macro sequence (e.g. `\m\per\s\squared`), ⟨*sig*⟩ is the number of significant figures used by `\<prefix><Name>num/qty`, and ⟨*type*⟩ is one of `stock`, `aux`, `constant`, or `system`. Each English type also accepts a Dutch alias (`voorraad`, `hulp`, `constante`, `systeem`) for backwards compatibility with existing teaching material. See Section 5.

Optional ⟨*keys*⟩:

`prefix` Override the current prefix for this single call.

`gridx, gridy` Manual placement in the `\graphicmodel` grid; integers, −1 leaves the slot to auto-layout (default).

`alias` Math-mode token list that replaces the entire *initial values* cell.

aliasleft, aliasright Replace just the left symbol or right value half of the cell.

`\mrule` Adds a rule of the form $\langle LHS \rangle \leftarrow \langle expr \rangle$. Signature:

```
\mrule*[<keys>]{<LHS>}{<expr>}
```

Both forms add the rule to *both* the rule table (`\textmodel`) and the simulation (`\computemodel`); execution is identical. The star only changes the typeset layout when $\langle expr \rangle$ is a ternary `cond ? a : b`. Without the star the ternary is rendered on a single table row,

```
IF cond THEN lhs = a ELSE lhs = b ENDIF
```

which is compact but wide. With the star (`\mrule*`) the same ternary is broken across rows,

```
IF cond THEN
    lhs = a
ELSE
    lhs = b
ENDIF
```

keeping the table column narrow so a `\graphicmodel` can sit alongside it, and making the source itself easier to read. For non-ternary expressions the star has no effect. $\langle expr \rangle$ may use the full `\fp_eval` expression grammar. See Table 2 (Section 4.2) for a complete overview of supported operators and functions with their XMILE and CoachTaal equivalents.

`\mruletext` Inserts a free-text row in the rule table without registering a rule with the simulator. Signature: `\mruletext[<keys>]{<text>}`. Useful for inserting comments or section dividers in long rule tables.

`\mstop` Sets the simulation stop condition. Signature: `\mstop[<keys>]{<expr>}`. The simulation halts at the first step where $\langle expr \rangle$ evaluates true. Exactly one `\mstop` per model prefix is required before `\computemodel`. Without one, `\computemodel` issues a warning.

4.2 Expression reference

Table 2 lists all expression constructs for `\mrule` bodies. The **XMILE** column shows XMILE v1.0 syntax, **l3fp** shows the `\fp_eval`-compatible form used inside `\mrule`, and **CoachTaal** shows the Coach 7 equivalent (function names from the standard CoachTaal math reference; note that argument separators in CoachTaal are semicolons). Orange entries are not yet fully supported by `numodel` (the expression computes correctly, but the rendered keyword in the rule table is not yet translated). *Italic* cells contain a derived equivalent rather than a native keyword.

Table 2: Expression reference: XMILE, `\fp_eval`, and CoachTaal

XMILE	**l3fp (`\fpeval`)**	**CoachTaal**
Arithmetic operators		
+	+	+
-	-	-
*	*	*
/	/	/
^	^	^
MOD(x,y)	*x - trunc(x/y)*y*	*x - Entier(x/y)*y*
Comparison operators		
<	<	<
<=	<=	<=
>	>	>
>=	>=	>=
=	=	=
<>	!=	<>
Boolean operators		
AND	&&	EN

Continued on next page

Table 2: Expression reference: XMILE, `\fp_eval`, and CoachTaal (Continued)

XMILE	l3fp (\fpeval)	CoachTaal
OR	||	OF
NOT	!	NIET
Control flow		
IF c THEN a ELSE b	c ? a : b	Als c Dan a Anders b EindAls
General math functions		
ABS(x)	abs(x)	Abs(x)
SIGN(x)	sign(x)	Teken(x)
SQRT(x)	sqrt(x)	Sqrt(x)
INT(x)	trunc(x)	*Entier(Abs(x))*Teken(x)*
INT(x+0.5)	round(x)	Round(x)
INT(x)	floor(x)	Entier(x)
-INT(-x)	ceil(x)	*-Entier(-x)*
MIN(x,y)	min(x,y)	Min(x;y)
MAX(x,y)	max(x,y)	Max(x;y)
—	fact(x)	Fac(x)
INT(LOG10(ABS(x)))	logb(x)	*Entier(Log(Abs(x)))*
Exponential and logarithmic		
EXP(x)	exp(x)	Exp(x)
LN(x)	ln(x)	Ln(x)
LOG10(x)	*ln(x)/ln(10)*	Log(x)
Trigonometry (radians)		
SIN(x)	sin(x)	Sin(x)
COS(x)	cos(x)	Cos(x)
TAN(x)	tan(x)	Tan(x)
ARCSIN(x)	asin(x)	Arcsin(x)
ARCCOS(x)	acos(x)	Arccos(x)
ARCTAN(x)	atan(x)	Arctan(x)
ARCTAN2(y,x)	atan(y,x)	*Arctan(y/x)*
1/TAN(x)	cot(x)	*1/Tan(x)*
1/SIN(x)	csc(x)	*1/Sin(x)*
1/COS(x)	sec(x)	*1/Cos(x)*
ARCSIN(1/x)	acsc(x)	*Arcsin(1/x)*
ARCCOS(1/x)	asec(x)	*Arccos(1/x)*
ARCTAN(1/x)	acot(x)	*Arctan(1/x)*
ARCTAN2(x,y)	acot(y,x)	*Arctan(x/y)*
Trigonometry (degrees)		
*SIN(PI/180*x)*	sind(x)	*Sin(Pi/180*x)*
*COS(PI/180*x)*	cosd(x)	*Cos(Pi/180*x)*
*TAN(PI/180*x)*	tand(x)	*Tan(Pi/180*x)*
*180/PI*ARCSIN(x)*	asind(x)	*180/Pi*Arcsin(x)*
*180/PI*ARCCOS(x)*	acosd(x)	*180/Pi*Arccos(x)*
*180/PI*ARCTAN(x)*	atand(x)	*180/Pi*Arctan(x)*
*180/PI*ARCTAN2(y,x)*	atand(y,x)	*180/Pi*Arctan(y/x)*
*1/TAN(PI/180*x)*	cotd(x)	*1/Tan(Pi/180*x)*
*1/SIN(PI/180*x)*	cscd(x)	*1/Sin(Pi/180*x)*
*1/COS(PI/180*x)*	secd(x)	*1/Cos(Pi/180*x)*
*180/PI*ARCSIN(1/x)*	acscd(x)	*180/Pi*Arcsin(1/x)*
*180/PI*ARCCOS(1/x)*	asecd(x)	*180/Pi*Arccos(1/x)*

Continued on next page

Table 2: Expression reference: XMILE, `\fp_eval`, and CoachTaal (Continued)

XMILE	l3fp (`\fpeval`)	CoachTaal
*180/PI*ARCTAN(1/x)*	`acotd(x)`	*180/Pi*Arctan(1/x)*
*180/PI*ARCTAN2(x,y)*	`acotd(y,x)`	*180/Pi*Arctan(x/y)*
Constants		
`PI`	`pi`	`Pi`
`e`	*exp(1)*	*Exp(1)*
`INF`	`inf`	—
`NAN`	`nan`	—
PI/180	`deg`	*Pi/180*
1	`true`	`Aan`
0	`false`	`Uit`
Simulation-specific functions (XMILE only)		
`TIME`	*user variable (`\mvar`)*	*user variable*
`DT`	*user variable (`\mvar`)*	*user variable*
`STEP(h,t0)`	*T >= t0 ? h : 0*	*Als T >= t0 Dan h Anders 0 EindAls*
`RAMP(s,t0)`	*T > t0 ? s*(T-t0) : 0*	*Als T > t0 Dan s*(T-t0) Anders 0 EindAls*
`DELAY(x,dt)`	*not supported*	*not supported*
`SMOOTH(x,t)`	*not supported*	*not supported*
`INIT(x)`	*not supported*	*not supported*
`PREVIOUS(x)`	*not supported*	*not supported*
RANDOM(0,1)	`rand()`	`Rand`
`RANDOM(lo,hi)`	*lo + (hi-lo)*rand()*	*lo + (hi-lo)*Rand*
not supported	`randint(n)`	*not supported*
not supported	`randint(m,n)`	*not supported*

4.3 Render commands

`\textmodel` Renders the rule-and-startvalue table. Optional `[<keys>]` accepts `prefix=<name>` (render a non-current model), `units=true` or `units=false`, and `tblrenv=tblr`longtblr`talltblr`. Each per-call key overrides the global `\numodelsetup` setting for this single render only; the global state is restored afterwards.

`tblrenv` selects which `tabularray` environment wraps the table: `longtblr` (default) breaks across pages, `tblr` renders inline without page breaks, `talltblr` renders inline but with caption and note support. Use `tblr` or `talltblr` when `\textmodel` sits inside an enclosing environment that suppresses page breaks (`subfigure`, `minipage`, …); the default `longtblr` would otherwise emit spurious continuation markers there.

Row spacing. `numodel` loads `tabularray` and sets `\SetTblrInner{rowsep=0pt}` globally so the rule listing renders compactly. This applies to *every* `tabularray` table in the document; if you want the default spacing back in your own tables, issue `\SetTblrInner{rowsep=2pt}` (or whatever value you prefer) somewhere in your document.

`\graphicmodel` Renders the Forrester stock-and-flow diagram. Variables of type `stock` become rectangles, `constant` become circles, `aux` become identifier nodes; flow arrows connect stocks to constant or helper sources/sinks based on which variables appear in the right-hand side of stock-updating rules.

Optional `[<keys>]` accepts `prefix=<name>` (render a non-current model) and `diagram-style=tight`forresteredu which overrides the global `\numodelsetup` setting for this single render only. The global state is restored afterwards, so multiple `\graphicmodel` calls can each pick their own style without re-issuing `\numodelsetup`.

`\computemodel` Runs the Euler simulation in Lua. Records every variable's value at every step, plus running min and max. After it returns, the accessors `\<prefix><Name>min` / `\<prefix><Name>max` hold the extrema, and `\<prefix><Name>` holds the final-step value. The time-series can be retrieved with `\mcoords` / `\mstep`.

`\diagrammodel` Convenience wrapper that produces a complete `figure` with caption and label. Signature:

`\diagrammodel[<keys>]{<xvar>}{<yvar>⟨,…⟩}[<extra>]{<label>}`

Reads min/max and display text from `\<prefix><xvar>` etc., delegates to `\drawplot` from numodel-

plot, and emits `\caption{$y(x)$-diagram}\label{fig:<label>}`. The optional ⟨*extra*⟩ argument is appended to the `axis` body, useful for additional `\addplot` lines (annotations, theoretical curves). Must be called after `\computemodel`.

The ⟨*yvar*⟩ argument accepts a comma-separated list of variables: every entry whose `unitraw` matches the first one's is drawn into the same diagram as discrete model points (no connecting lines), with the y-axis range scaled to the joint min/max of the kept series. Entries with a non-matching unit are dropped and a warning is issued. The seven colours cycle through Okabe & Ito's colour-blind safe palette (yellow omitted), ordered so consecutive series differ in luminance – which keeps them distinguishable on a greyscale printout as well. The legend lists each kept variable's display text.

4.4 Series accessors

`\mcoords` Returns a comma-separated PGFPlots coordinate list of the simulated series for two variables. Fully expandable. Signature: `\mcoords{<xvar>}{<yvar>}` (current prefix); `\mcoordsp{<prefix>}{<xvar>}{<yvar>}` (explicit prefix). Both forms are usable inside `\addplot coordinates{ ... }`.

`\mstep` Returns the value of one variable at a chosen iteration. Fully expandable. Signature: `\mstep{<Name>}{<i>}`; `\mstepp{<prefix>}{<Name>}{<i>}`. The current prefix is prepended automatically by `\mstep`. Step indexing is 0-based: step 0 is the initial-values row, step $N-1$ is the final recorded step after `\computemodel`. Negative indices count from the end (Python-style), so `\mstep{Y}{-1}` is the last recorded y and `\mstep{Y}{-2}` is the penultimate one. Returns nothing (silently) if the index is out of range.

> Example – a red secant line through the last two simulated (t, y) points of the free-fall ball (the model from the first example), extrapolated across the whole t-domain and labelled in the legend. The slope is the rise-over-run of the last two steps, the intercept is the final y-value:

```
\diagrammodel{T}{Y}[%
  \addplot[red, very thick, domain=\ballTmin:\ballTmax]
    {\mstep{Y}{-1}
      + (\mstep{Y}{-1} - \mstep{Y}{-2})
      / (\mstep{T}{-1} - \mstep{T}{-2})
      * (x - \mstep{T}{-1})};
  \addlegendentry{secant endpoints}
]{ball-fall-secant}
```

Inside the optional `[<extra>]` argument `\mstep` expands as a literal number into PGFPlots' math parser, so each `\mstep` is evaluated only once (when the expression is constructed) rather than per sample. Applied to the `ball` model:

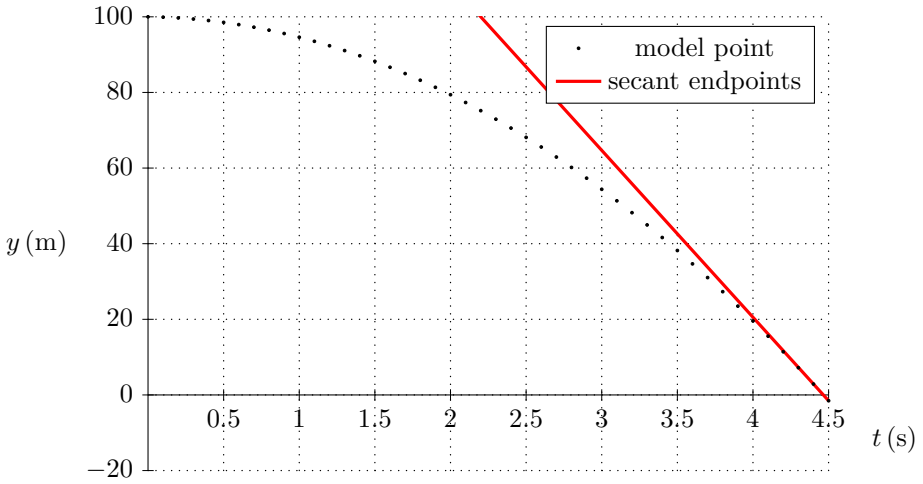


Figure 2: $y(t)$-diagram

4.5 Namespace management

`\newmodelprefix` Creates a new model namespace and switches to it. Signature: `\newmodelprefix{<name>}`. Subsequent `\mvar` / `\mrule` / `\mstop` calls bind to this prefix. All generated accessors are prefixed (so `\mvar{Y}{y}{...}{...}{...}{...}` under prefix `ball` produces `\ballY`, `\ballYtext`, etc.).

`\switchmodelprefix` Switches to a previously created prefix. Useful when a document defines several models early and renders them later out of order. Signature: `\switchmodelprefix{<name>}`.

`\NumodelForEachVar` Iterates a token list over every registered variable across every known prefix. Signature: `\NumodelForEachVar{<code with #1>}`; inside the body, `#1` is the full prefixed name (e.g. `ballY`). Used by external tools (e.g. a worksheet system) that need to expand every model accessor.

4.6 Shared valves

When the same auxiliary variable appears as the inflow of more than one stock, `\graphicmodel` draws a single valve next to the first such stock and threads a curved branch (`to[bend left=30]` – the same bend as the curved causal arrows) from that valve over the primary stock into each additional one. The auto-layout leaves the involved stocks side by side on the same row so the branch stays compact. A minimal example:

```
\mvar{R}{r}{5}{\m\cubed\per\s}{2}{aux}
\mvar{Va}{V_1}{0}{\m\cubed}{3}{stock}
\mvar{Vb}{V_2}{0}{\m\cubed}{3}{stock}
\mrule{Va}{\Va + \R * \Dt}
\mrule{Vb}{\Vb + \R * \Dt}
```

renders one `R` valve, a straight pipe `R` → `V_a`, and a curved branch `R` → `V_b` arcing over `V_a`. (Note: TeX control words do not accept digits, so the macro names `Va`/`Vb` are used internally while the display texts `V_1`/`V_2` appear in the rendered diagram and table.)

Shared *outflows* are handled symmetrically. When one variable drains more than one stock, the valve is placed to the right of the last-declared source stock, and every earlier source attaches with a curved branch arcing over the intervening stocks into the shared valve. This matters in particular for physical models where a single Stefan–Boltzmann constant or friction coefficient drains several reservoirs into the same sink.

Two flow-classification refinements support these layouts:

- The flow-variable heuristic prefers `aux` and `stock` variables over `constant`s when both appear in the same additive term, so an inflow valve carries the meaningful rate variable instead of an earlier-declared scaling constant.
- Top-level parenthesised inflow terms of the form `(A - B)*C` are distributed into `+A*C, -B*C` before classification, so `A` surfaces as the inflow valve and `B` (typically a radiative or dissipative loss involving a constant) surfaces as the outflow valve.

5 Variable types

The sixth `\mvar` argument (⟨*type*⟩) tags a variable with a role. The type drives both `\textmodel` layout (whether the row appears in *initial values*) and `\graphicmodel` node shape. Each type has a canonical English name and a Dutch alias; the two are interchangeable.

`stock` (alias `voorraad`) A stock that accumulates over time. Drawn as a rectangle in the Forrester diagram; its rule must be of the form stock ← stock + ... so that the integrator can detect inflows and outflows. Receives an *initial values* row.

`constant` (alias `constante`) A constant parameter (mass, gravitational acceleration, spring stiffness, ...). Drawn as a circle. Receives an *initial values* row.

`aux` (alias `hulp`) An auxiliary variable computed from other variables on each step. Drawn as a plain identifier node, no rectangle. No *initial values* row (start value is normally left empty).

`system` (alias `systeem`) System-level bookkeeping (time `T`, step size `Dt`, terminal time, ...). Drawn separately, no flow arrows. Receives an *initial values* row.

6 Generated accessors

Each `\mvar` call generates a family of accessor macros named `\<prefix><Name><suffix>`. The full set:

(no suffix) Current numeric value (post-rule-update if inside a step, post-final-step after `\computemodel`).

`text` The display symbol passed as the second argument of `\mvar`.

`unit` `\unit{...}` applied to the unit argument (siunitx-formatted).

`unitraw` The unit argument verbatim, without the `\unit{}` wrapper, suitable for use as a building block (e.g. `\xlabelunit` in numodel-plot).

`num` The current value formatted via siunitx's `\num{}` with the variable's significant figures. Used by `\textmodel` in the *initial values* column when `units=false`.

`qty` As `num`, but including the unit (`\qty{value}{unit}`). Used by `\textmodel` in the *initial values* column when `units=true` (the default).

`pre` As `qty`, but with engineering prefix mode (e.g. `1500 W` renders as `1{,}5\,kW`).

`sign` Significant-figure count (raw integer).

`type` Variable type (raw string).

`min`, `max` Extrema over the simulated series. Empty before `\computemodel`.

`gridx`, `gridy` Manual placement coordinates in the Forrester grid; -1 if left to auto-layout.

`alias`, `aliasleft`, `aliasright` Override tokens for the *startwaarden* cell layout.

7 Multiple models in one document

Each `\newmodelprefix` starts a fresh namespace. This lets a document contain several unrelated models without name clashes:

```
\newmodelprefix{ball}
\mvar{Y}{y}{100}{\m}{3}{stock}
% ... ball model ...

\newmodelprefix{spring}
\mvar{X}{x}{0.1}{\m}{3}{stock}
% ... spring model ...

% Render the ball model:
\switchmodelprefix{ball}\textmodel\computemodel\diagrammodel{T}{Y}{fall}

% Render the spring model:
\switchmodelprefix{spring}\textmodel\computemodel\diagrammodel{T}{X}{spring}
```

Each prefix carries an independent rule list, stop condition, and recorded series.

8 Requirements

numodel requires LuaLaTeX (the engine, for the Lua runtime) and TeX Live 2022 or later. Mandatory dependencies: expl3, xparse, l3keys2e, amsmath, amssymb, tikz, luacode, siunitx, float, and the sibling package numodel-plot (which itself pulls in pgfplots). The companion Lua module `numodel.lua` must be installed alongside the `.sty` in a directory searched by `kpse`.

9 Implementation

```
⟨*package⟩
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{numodel}[2026/05/23 v0.5.0
  Numerical physics models with Euler integration and Forrester diagrams]

\RequirePackage{expl3}
\RequirePackage{xparse}
\RequirePackage{l3keys2e}
\RequirePackage{amsmath}
\RequirePackage{amssymb}      % \leqslant / \geqslant in rendered <= / >=
\RequirePackage{tikz}
\usetikzlibrary{shapes.symbols}
\usetikzlibrary{arrows.meta}
% Custom Cloud arrow tip used as the open end of inflow / outflow
% pipes.  The silhouette consists of nine concatenated elliptical
% arcs (\pgfpatharc): each arc starts at the current path point,
% which pgf treats as lying on an ellipse at angle <start>, and
% sweeps through to angle <end>.  Varying the start and end angles
% per arc produces zigzagging half-circle puffs -- the
% characteristic cloud outline.  A single factor \puffscale sets the
% rx and ry of every ellipse and thus the size of all puffs at once.
% The chain has a horizontal span of (4+2*sqrt(2))*rx ~ 6.83*rx;
% with puffscale=0.22 that is ~ 1.5*\pgfarrowlength.  The path
% therefore starts at -0.5*\pgfarrowlength so the silhouette covers
% [-0.5L, L] in local arrow coordinates and the apex lands exactly
% at \pgfarrowlength -- where the built-in tips (Latex, Stealth,
% ...) also place their tip.  The line width is set explicitly to
% 0.3pt, independent of the arrow's line width, so a thick arrow
% does not also get a thick cloud outline.
\pgfdeclarearrow{
  name = Cloud,
  parameters = { \the\pgfarrowlength \the\pgfarrowwidth },
  setup code = {
    \pgfarrowssettipend{\pgfarrowlength}
    \pgfarrowssetbackend{-0.5\pgfarrowlength}
    \pgfarrowssetlineend{-0.5\pgfarrowlength}
    \pgfarrowssetvisualbackend{-0.5\pgfarrowlength}
    \pgfarrowssetvisualtipend{\pgfarrowlength}
  },
  defaults = { length = 1em, width = 1em },
  drawing code = {
    \def\puffscale{0.22}
    \edef\puffrx{\puffscale\pgfarrowlength}
    \edef\puffry{\puffscale\pgfarrowwidth}
    \pgfsetlinewidth{0.3pt}
    \pgfpathmoveto{\pgfqpoint{-0.5\pgfarrowlength}{0pt}}
    \pgfpatharc{180}{ 90}{\puffrx\space and \puffry}
    \pgfpatharc{225}{ 45}{\puffrx\space and \puffry}
    \pgfpatharc{180}{  0}{\puffrx\space and \puffry}
    \pgfpatharc{135}{-45}{\puffrx\space and \puffry}
    \pgfpatharc{ 90}{-90}{\puffrx\space and \puffry}
    \pgfpatharc{ 45}{-135}{\puffrx\space and \puffry}
    \pgfpatharc{  0}{-180}{\puffrx\space and \puffry}
    \pgfpatharc{-45}{-225}{\puffrx\space and \puffry}
    \pgfpatharc{-90}{-180}{\puffrx\space and \puffry}
    \pgfpathclose
    \pgfusepathqfillstroke
  },
}
\RequirePackage{luacode}
\RequirePackage{siunitx}
\RequirePackage{float}        % \diagrammodel uses [H] placement
\RequirePackage{tabularray}   % \textmodel uses longtblr (page-breakable,
                              % works inside floats/subfigure)
```

```
% Define a numodel theme for longtblr in \textmodel.  The default
% `normal' middlehead/lasthead prefixes the continuation marker with
% `Table N:' from \tablename/\thetable; we don't issue \caption, so we
% want only the conthead text (which is itself localised via
% \tblrcontheadname, updated by \__numodel_refresh_kw:).  Likewise for
% the foot.  Activated via `theme = {numodel}' in the longtblr options.
\DefTblrTemplate {firsthead}            {numodel} {}
\DefTblrTemplate {middlehead, lasthead} {numodel} { \UseTblrTemplate {conthead} {default} }
\DefTblrTemplate {firstfoot, middlefoot}{numodel} { \UseTblrTemplate {contfoot} {default} }
\DefTblrTemplate {lastfoot}             {numodel} {}
\NewTblrTheme {numodel}
  {
    \SetTblrTemplate {firsthead}            {numodel}
    \SetTblrTemplate {middlehead, lasthead} {numodel}
    \SetTblrTemplate {firstfoot, middlefoot}{numodel}
    \SetTblrTemplate {lastfoot}             {numodel}
  }
% Tighten the default row separation of all tabularray tables: the
% \textmodel rule listing is denser and more compact than the
% tabularray default.  Users who want the default spacing back in
% their own tables can simply issue \SetTblrInner{rowsep=<value>} in
% their document.
\SetTblrInner{rowsep=0pt}
\RequirePackage{numodel-plot}

% ====================================================================
% Non-expl3 helpers: `\nm@def@num@cs` / `\nm@def@qty@cs` / `\nm@def@pre@cs`
% define `\<fullname><suffix>` macros whose body contains literal
% siunitx options.  Defined OUTSIDE \ExplSyntaxOn so the option-string
% colons (`-3:n`, `-0:0`) and other punctuation keep normal catcodes.
% Storing those bodies inside an expl3-context would tokenise `:` as
% letter, which then breaks siunitx's option parser at expansion time.
% ====================================================================
\newcommand{\NumodelDefNumCs}[3]{% #1 = fullname, #2 = start expr, #3 = sigfigs
  \expandafter\gdef\csname #1num\endcsname{%
    \num[evaluate-expression=true,
         round-mode=figures, round-precision=#3,
         exponent-mode=threshold, exponent-thresholds=-3:#3]
        {#2}\relax
  }%
}
\newcommand{\NumodelDefQtyCs}[4]{% #1 = fullname, #2 = expr, #3 = sigfigs, #4 = unit
  \expandafter\gdef\csname #1qty\endcsname{%
    \qty[evaluate-expression=true,
         round-mode=figures, round-precision=#3,
         exponent-mode=threshold, exponent-thresholds=-3:#3]
        {#2}{#4}\relax
  }%
}
\newcommand{\NumodelDefPreCs}[4]{% same parameters as above
  \expandafter\gdef\csname #1pre\endcsname{%
    \qty[evaluate-expression=true,
         round-mode=figures, round-precision=#3,
         prefix-mode=combine-exponent,
         exponent-mode=engineering,
         exponent-thresholds=-0:0]
        {#2}{#4}\relax
  }%
}

\ExplSyntaxOn

% ====================================================================
% Lua module for data storage (O(1) append, O(N) total)
% ====================================================================
% All variable values are stored per step in Lua tables.  This
```

```
% replaces the O(N^2) \xdef accumulation for coordinates and makes
% min/max tracking efficient.  After \computemodel the results are
% pushed back to TeX macros.

\begin{luacode*}
local f = kpse.find_file("numodel.lua", "tex")
if f and f ~= "" then
  dofile(f)
else
  tex.error("numodel: cannot find companion Lua module numodel.lua."
         .. " Install it in a directory searched by kpse"
         .. " (e.g. TEXMFHOME/tex/lualatex/numodel/).")
end
\end{luacode*}

% ====================================================================
% Internal data structures
% ====================================================================

\seq_new:N  \g_mvar_names_seq      % all model variable names
\seq_new:N  \g_mvar_start_seq      % names with an initial value (for the table)
\seq_new:N  \g_mrule_seq           % model rules (display strings)
\seq_new:N  \g_mrule_type_seq      % type per display row: rule | cont
\seq_new:N  \g_mrule_calc_seq      % model rules for execution: {name}{expr}
\int_new:N  \g_mrule_counter_int   % rule counter (display numbering)
\tl_new:N   \g_numodel_stop_expr_tl  % stop-condition expression for execution
\int_new:N  \g_numodel_steps_int     % number of executed steps
\int_new:N  \g_numodel_maxiter_int   % safety limit (init via \numodelsetup)
\int_new:N  \g_numodel_gridmaxx_int  % \graphicmodel wrap threshold (0 = off)

% --- Graphic-model infrastructure (Phase 1) ---
\tl_new:N   \l__numodel_tmp_tl       % temporary helper
\tl_new:N   \l__numodel_scratch_tl   % scratch for type/text checks
\tl_new:N   \l__numodel_scratch_y_tl % scratch y-coord (svgy lookup)

% Diagram style for \graphicmodel (see \numodelsetup):
%   tight     -- valve takes the label of the direct inflow variable;
%                the aux/const is moved to the valve position and not
%                drawn as a separate node (compact, default).
%   forrester -- valve has no label; the aux/const stays at its own
%                gridy position; causal arrow from aux/const to the
%                valve.
%   edu       -- combination: valve takes the label *and* a separate
%                aux/const node remains with a causal arrow to it.
\tl_new:N \g__numodel_diagram_style_tl

% Sub-keys that fine-tune the appearance of the Forrester diagram
% (see \numodelsetup).  Empty value = "follow diagram-style default".
%   flowarrow-style    : hollow | filled
%                        forrester -> hollow, tight|edu -> filled
%   valve-style        : valve | circle | edu
%                        forrester -> valve, tight|edu -> edu
%   flowarrow-cloud-tip: true | false
%                        forrester -> true, tight|edu -> false
% Per-variable override for flowarrow-cloud-tip is stored in
% \<name>flowcloud (set via the [flowarrow-cloud-tip=...] key on
% \mvar; empty means "follow global key").
\tl_new:N \g__numodel_flowarrow_style_tl
\tl_new:N \g__numodel_valve_style_tl
\tl_new:N \g__numodel_flowcloud_tl

% Units in the \textmodel initial-values column (see \numodelsetup):
%   true  (default) -- initial value rendered via \<name>qty
%                      (number + unit)
%   false           -- initial value rendered via \<name>num
%                      (number only)
```

```
% Per-table override via \textmodel[units=true|false].
\bool_new:N \g__numodel_units_bool

% tabularray environment used by \textmodel (see \numodelsetup{tblrenv}):
%   longtblr -- default; page-breakable; suitable for body text.
%   tblr     -- basic; no page breaks; choose this when \textmodel sits
%               inside an outer environment that already suppresses page
%               breaks (subfigure, minipage, ...).
%   talltblr -- like tblr (no page breaks) but supports captions/notes.
% Per-table override via \textmodel[tblrenv=...].
\tl_new:N \g__numodel_tblrenv_tl

% Decimal separator for \textmodel initial values and \diagrammodel
% tick values (see \numodelsetup{decimal-separator}):
%   point   -- '.' (siunitx output-decimal-marker={.})
%   comma   -- ',' (siunitx output-decimal-marker={,})
% The default follows syntax: english -> point, coachtaal -> comma.
% An explicit decimal-separator key sets
% \g__numodel_dsep_explicit_bool so that a later syntax change no
% longer overrules the choice.
\tl_new:N   \g__numodel_dsep_tl
\bool_new:N \g__numodel_dsep_explicit_bool

% ======================================================================
% Syntax lookup
% ======================================================================
% Determines the language of the text-rendered model.  Affects display
% only (\textmodel, \mruletext, \mstop); \computemodel always uses
% \fpeval internally and is language-agnostic.  Each value of
% \g__numodel_syntax_tl is a tag that selects both the backing
% \__numodel_kw_<tag>_<key>: macros and the file numodel-<tag>.def
% that defines them; the package ships EN (English/XMILE) and NL
% (Dutch/CoachTaal).
%
% The CTAN default is EN.  The initial value is set below via
% \numodelsetup.

\tl_new:N \g__numodel_syntax_tl

% Lookup (fully expandable): \__numodel_kw:n {<key>} returns the
% translation in the current syntax language.  The key also controls
% the surrounding spaces (see keys 'then' vs 'then_nl' etc.).  The
% backing macros \__numodel_kw_<LANG>_<key>: are provided by the
% language file numodel-<LANG>.def loaded by \__numodel_load_syntax_def:n
% (see below).
\cs_new:Npn \__numodel_kw:n #1
  { \use:c { __numodel_kw_ \g__numodel_syntax_tl _ #1 : } }

% Pre-computed translations in tl variables so they are usable via
% \u{} in l3regex replacement text.  (l3regex treats {...} in the
% replacement as brace groups, so a bare \__numodel_kw:n{key} call
% fails there.)
\tl_new:N \g__numodel_kw_if_tl
\tl_new:N \g__numodel_kw_then_tl
\tl_new:N \g__numodel_kw_then_nl_tl
\tl_new:N \g__numodel_kw_else_tl
\tl_new:N \g__numodel_kw_else_nl_tl
\tl_new:N \g__numodel_kw_endif_tl
\tl_new:N \g__numodel_kw_endif_nl_tl
\tl_new:N \g__numodel_kw_and_tl
\tl_new:N \g__numodel_kw_or_tl
\tl_new:N \g__numodel_kw_not_tl
\tl_new:N \g__numodel_kw_sign_tl
\tl_new:N \g__numodel_kw_abs_tl
\tl_new:N \g__numodel_kw_sqrt_tl
\tl_new:N \g__numodel_kw_exp_tl
```

```
\tl_new:N \g__numodel_kw_ln_tl
\tl_new:N \g__numodel_kw_sin_tl
\tl_new:N \g__numodel_kw_cos_tl
\tl_new:N \g__numodel_kw_tan_tl
\tl_new:N \g__numodel_kw_asin_tl
\tl_new:N \g__numodel_kw_acos_tl
\tl_new:N \g__numodel_kw_stop_tl

% Recomputes all tl caches from \g__numodel_syntax_tl.  Must be
% called after any change to the syntax language.
\cs_new_protected:Npn \__numodel_refresh_kw:
  {
    \tl_gset:Ne \g__numodel_kw_if_tl       { \__numodel_kw:n {if}       }
    \tl_gset:Ne \g__numodel_kw_then_tl     { \__numodel_kw:n {then}     }
    \tl_gset:Ne \g__numodel_kw_then_nl_tl  { \__numodel_kw:n {then_nl}  }
    \tl_gset:Ne \g__numodel_kw_else_tl     { \__numodel_kw:n {else}     }
    \tl_gset:Ne \g__numodel_kw_else_nl_tl  { \__numodel_kw:n {else_nl}  }
    \tl_gset:Ne \g__numodel_kw_endif_tl    { \__numodel_kw:n {endif}    }
    \tl_gset:Ne \g__numodel_kw_endif_nl_tl { \__numodel_kw:n {endif_nl} }
    \tl_gset:Ne \g__numodel_kw_and_tl      { \__numodel_kw:n {and}      }
    \tl_gset:Ne \g__numodel_kw_or_tl       { \__numodel_kw:n {or}       }
    \tl_gset:Ne \g__numodel_kw_not_tl      { \__numodel_kw:n {not}      }
    \tl_gset:Ne \g__numodel_kw_sign_tl     { \__numodel_kw:n {sign}     }
    \tl_gset:Ne \g__numodel_kw_abs_tl      { \__numodel_kw:n {abs}      }
    \tl_gset:Ne \g__numodel_kw_sqrt_tl     { \__numodel_kw:n {sqrt}     }
    \tl_gset:Ne \g__numodel_kw_exp_tl      { \__numodel_kw:n {exp}      }
    \tl_gset:Ne \g__numodel_kw_ln_tl       { \__numodel_kw:n {ln}       }
    \tl_gset:Ne \g__numodel_kw_sin_tl      { \__numodel_kw:n {sin}      }
    \tl_gset:Ne \g__numodel_kw_cos_tl      { \__numodel_kw:n {cos}      }
    \tl_gset:Ne \g__numodel_kw_tan_tl      { \__numodel_kw:n {tan}      }
    \tl_gset:Ne \g__numodel_kw_asin_tl     { \__numodel_kw:n {asin}     }
    \tl_gset:Ne \g__numodel_kw_acos_tl     { \__numodel_kw:n {acos}     }
    \tl_gset:Ne \g__numodel_kw_stop_tl     { \__numodel_kw:n {stop}     }
    % tabularray continuation markers (used by longtblr in \textmodel)
    \tl_gset:Ne \tblrcontfootname { \__numodel_kw:n {contfoot} }
    \tl_gset:Ne \tblrcontheadname { \__numodel_kw:n {conthead} }
  }

% Helper: \__numodel_kwt:n {<key>} expands to \text{<kw-value>}.
% Meant for use inside \tl_set:Ne / \seq_gput_right:Ne -- the kw is
% inserted during e-expansion while \text remains protected.
\cs_new:Npn \__numodel_kwt:n #1 { \text { \__numodel_kw:n {#1} } }

% Language-file loader.  Each tag <LANG> is backed by a file
% numodel-<LANG>.def installed in a kpse-searched directory; the
% package ships numodel-EN.def and numodel-NL.def, users can drop
% additional files in TEXMFHOME/tex/latex/numodel/ without rebuilding
% the package.
%
% A small alias property maps the historical long names to the
% canonical two-letter tag used in the file name and the internal
% macro names.  Users who add their own file just pass the file's tag
% to syntax=<tag> directly; no alias is required.
\prop_new:N \g__numodel_syntax_aliases_prop
\prop_gput:Nnn \g__numodel_syntax_aliases_prop { english }   { EN }
\prop_gput:Nnn \g__numodel_syntax_aliases_prop { coachtaal } { NL }
\prop_gput:Nnn \g__numodel_syntax_aliases_prop { dutch }     { NL }

% Tags whose .def file has already been input during this run, so the
% lookup only triggers \InputIfFileExists the first time.
\seq_new:N \g__numodel_loaded_syntax_seq

\msg_new:nnn { numodel } { unknown-syntax }
  {
    Cannot~load~syntax~'#1':~file~'numodel-#1.def'~not~found~by~
    kpse.~The~package~ships~with~EN~(English)~and~NL~(Dutch~
```

```
    CoachTaal);~drop~your~own~numodel-<LANG>.def~in~
    TEXMFHOME/tex/latex/numodel/~to~add~more~languages.
  }

% \__numodel_load_syntax_def:n {<tag>}
% Inputs numodel-<tag>.def the first time it is requested.  Raises a
% LaTeX error if the file is not found by kpse.
\cs_new_protected:Npn \__numodel_load_syntax_def:n #1
  {
    \seq_if_in:NnF \g__numodel_loaded_syntax_seq {#1}
      {
        \InputIfFileExists { numodel-#1.def }
          { \seq_gput_right:Nn \g__numodel_loaded_syntax_seq {#1} }
          { \msg_error:nnn { numodel } { unknown-syntax } {#1} }
      }
  }

\tl_new:N \l__numodel_syntax_tag_tl

% \__numodel_set_syntax:n {<value>}
% Public-facing setter behind the syntax key.  Resolves aliases,
% loads the matching .def file (once), publishes the canonical tag
% in \g__numodel_syntax_tl, refreshes the keyword cache, and -- when
% the user has not pinned a decimal separator explicitly -- adopts the
% language's preferred decimal mark via the optional
% \__numodel_kw_<LANG>_dsep_default: hook.
\cs_new_protected:Npn \__numodel_set_syntax:n #1
  {
    \prop_get:NnNF \g__numodel_syntax_aliases_prop {#1}
      \l__numodel_syntax_tag_tl
      { \tl_set:Nn \l__numodel_syntax_tag_tl {#1} }
    \exp_args:NV \__numodel_load_syntax_def:n \l__numodel_syntax_tag_tl
    \tl_gset_eq:NN \g__numodel_syntax_tl \l__numodel_syntax_tag_tl
    \__numodel_refresh_kw:
    \bool_if:NF \g__numodel_dsep_explicit_bool
      {
        \cs_if_exist:cTF
          { __numodel_kw_ \g__numodel_syntax_tl _dsep_default: }
          { \tl_gset:Ne \g__numodel_dsep_tl { \__numodel_kw:n {dsep_default} } }
          { \tl_gset:Nn \g__numodel_dsep_tl { point } }
      }
  }

% Applies the decimal separator (only for the duration of the
% surrounding TeX group, so a document-wide \sisetup is left
% untouched).  Calls both siunitx (for \num/\qty in \textmodel
% initial values) and pgfplots' tick styles (for \diagrammodel tick
% labels) so the choice is consistent everywhere.
\cs_new_protected:Npn \__numodel_apply_dsep:
  {
    \str_if_eq:VnTF \g__numodel_dsep_tl { comma }
      {
        \sisetup{ output-decimal-marker = {,} }
        % Append behind the existing numodel/axis style so our
        % xticklabel-style replaces what numodel-plot hard-codes.
        % The \pgfplotsset mutation is \def-based and hence
        % group-local.
        \pgfplotsset
          {
            numodel/axis/.append~style=
              {
                xticklabel~style=
                  { /pgf/number~format/.cd, use~comma, /tikz/.cd } ,
                yticklabel~style=
                  { /pgf/number~format/.cd, use~comma, /tikz/.cd } ,
              }
```

```
          }
      }
      {
        \sisetup{ output-decimal-marker = {.} }
        \pgfplotsset
          {
            numodel/axis/.append~style=
              {
                xticklabel~style=
                  { /pgf/number~format/.cd, use~period, /tikz/.cd } ,
                yticklabel~style=
                  { /pgf/number~format/.cd, use~period, /tikz/.cd } ,
              }
          }
      }
  }

% ======================================================================
% Configuration command \numodelsetup
% ======================================================================
% Runtime API for settings.  Keys:
%   syntax     -- language tag (file numodel-<tag>.def must be on kpse
%                 path).  Built-in: EN, NL.  Legacy aliases: english,
%                 coachtaal, dutch.
%   maxiter    -- safety limit for \computemodel (default 20000)
%   graphscalex -- horizontal grid spacing \dgridx for Forrester
%                  diagrams (default 2)
%   graphscaley -- vertical grid spacing \dgridy for Forrester
%                  diagrams (default 2)
%   stockwidth  -- half-width of the stock rectangle (default 0.375)

% Helper for choice-with-empty-reset: validates #4 against #3 (a
% comma-separated whitelist) and writes it into #1 (a tl).  An empty
% value clears the tl (which means "follow the diagram-style
% default" for the flowarrow-style / valve-style /
% flowarrow-cloud-tip keys).  Any other value triggers a warning.
\msg_new:nnn { numodel } { bad-choice }
  { Key~'#1'~accepts~only~#2,~or~empty~to~reset.~Got:~'#3'. }

\cs_new_protected:Npn \__numodel_setup_choice:Nnnn #1 #2 #3 #4
  {
    \tl_if_blank:nTF {#4}
      { \tl_gclear:N #1 }
      {
        \clist_if_in:nnTF {#3} {#4}
          { \tl_gset:Nn #1 {#4} }
          { \msg_warning:nnnnn { numodel } { bad-choice } {#2} {#3} {#4} }
      }
  }

% Local-tl variant: same validation, local set/clear.  Used by the
% \graphicmodel one-render override and the per-\mvar override.
\cs_new_protected:Npn \__numodel_local_choice:Nnnn #1 #2 #3 #4
  {
    \tl_if_blank:nTF {#4}
      { \tl_clear:N #1 }
      {
        \clist_if_in:nnTF {#3} {#4}
          { \tl_set:Nn #1 {#4} }
          { \msg_warning:nnnnn { numodel } { bad-choice } {#2} {#3} {#4} }
      }
  }

\keys_define:nn { numodel / setup }
  {
    syntax              .code:n =
```

```
      { \__numodel_set_syntax:n {#1} },
    maxiter             .int_gset:N = \g_numodel_maxiter_int,
    graphscalex         .code:n = { \tl_gset:Nn \dgridx {#1} },
    graphscaley         .code:n = { \tl_gset:Nn \dgridy {#1} },
    stockwidth          .code:n = { \tl_gset:Nn \halfstockwidth {#1} },
    gridmaxx            .int_gset:N = \g_numodel_gridmaxx_int,
    diagram-style               .choice:,
    diagram-style / tight       .code:n =
      { \tl_gset:Nn \g__numodel_diagram_style_tl { tight } },
    diagram-style / forrester   .code:n =
      { \tl_gset:Nn \g__numodel_diagram_style_tl { forrester } },
    diagram-style / edu         .code:n =
      { \tl_gset:Nn \g__numodel_diagram_style_tl { edu } },
    flowarrow-style     .code:n =
      { \__numodel_setup_choice:Nnnn \g__numodel_flowarrow_style_tl
          { flowarrow-style } { hollow , filled } {#1} },
    valve-style         .code:n =
      { \__numodel_setup_choice:Nnnn \g__numodel_valve_style_tl
          { valve-style } { valve , circle , edu } {#1} },
    flowarrow-cloud-tip .code:n =
      { \__numodel_setup_choice:Nnnn \g__numodel_flowcloud_tl
          { flowarrow-cloud-tip } { true , false } {#1} },
    units                       .choice:,
    units / true                .code:n =
      { \bool_gset_true:N  \g__numodel_units_bool },
    units / false               .code:n =
      { \bool_gset_false:N \g__numodel_units_bool },
    tblrenv                     .choice:,
    tblrenv / tblr              .code:n =
      { \tl_gset:Nn \g__numodel_tblrenv_tl { tblr } },
    tblrenv / longtblr          .code:n =
      { \tl_gset:Nn \g__numodel_tblrenv_tl { longtblr } },
    tblrenv / talltblr          .code:n =
      { \tl_gset:Nn \g__numodel_tblrenv_tl { talltblr } },
    decimal-separator           .choice:,
    decimal-separator / comma   .code:n =
      {
        \tl_gset:Nn \g__numodel_dsep_tl { comma }
        \bool_gset_true:N \g__numodel_dsep_explicit_bool
      },
    decimal-separator / point   .code:n =
      {
        \tl_gset:Nn \g__numodel_dsep_tl { point }
        \bool_gset_true:N \g__numodel_dsep_explicit_bool
      },
  }

\NewDocumentCommand{\numodelsetup}{ m }
  { \keys_set:nn { numodel / setup } {#1} }

% Defaults (this call also initialises \g__numodel_syntax_tl,
% \g_numodel_maxiter_int, \dgridx, \dgridy, \halfstockwidth and
% \g__numodel_diagram_style_tl).
\numodelsetup
  {
    syntax        = EN,
    maxiter       = 20000,
    graphscalex   = 2,
    graphscaley   = 2,
    stockwidth    = 0.375,
    diagram-style = tight,
    units         = true,
    tblrenv       = longtblr,
  }

% Package-time options.  \usepackage[syntax=NL]{numodel} delegates
```

```
% to the same key infrastructure as \numodelsetup.
\keys_define:nn { numodel / pkg }
  {
    syntax              .meta:nn = { numodel / setup }{ syntax              = #1 },
    maxiter             .meta:nn = { numodel / setup }{ maxiter             = #1 },
    graphscalex         .meta:nn = { numodel / setup }{ graphscalex         = #1 },
    graphscaley         .meta:nn = { numodel / setup }{ graphscaley         = #1 },
    stockwidth          .meta:nn = { numodel / setup }{ stockwidth          = #1 },
    gridmaxx            .meta:nn = { numodel / setup }{ gridmaxx            = #1 },
    diagram-style       .meta:nn = { numodel / setup }{ diagram-style       = #1 },
    flowarrow-style     .meta:nn = { numodel / setup }{ flowarrow-style     = #1 },
    valve-style         .meta:nn = { numodel / setup }{ valve-style         = #1 },
    flowarrow-cloud-tip .meta:nn = { numodel / setup }{ flowarrow-cloud-tip = #1 },
    units               .meta:nn = { numodel / setup }{ units               = #1 },
    tblrenv             .meta:nn = { numodel / setup }{ tblrenv             = #1 },
    decimal-separator   .meta:nn = { numodel / setup }{ decimal-separator   = #1 },
  }
\ProcessKeyOptions [ numodel / pkg ]

% ======================================================================
% Prefix system
% ======================================================================
% Each model lives under a prefix (a short string).  The "live state"
% sits in the \g_mvar_*, \g_mrule_*, \g_numodel_* variables above;
% \newmodelprefix and \switchmodelprefix swap that state to/from
% per-prefix backup storage.
%
% - \g_numodel_current_prefix_tl: current prefix (empty at package load)
% - \g_numodel_prefixes_seq: list of all registered prefixes
% - \l__numodel_cmd_prefix_tl: prefix passed via [prefix=...] key
% - \l__numodel_eff_prefix_tl: effective prefix for the current command

\tl_new:N  \g_numodel_current_prefix_tl
\seq_new:N \g_numodel_prefixes_seq
\tl_new:N  \l__numodel_cmd_prefix_tl
\tl_new:N  \l__numodel_eff_prefix_tl
\tl_new:N  \l__numodel_fullname_tl

% Per-prefix storage: on swap the current state is copied to
% \g__numodel_<P>_<slot>_{seq,tl,int,prop}.  Load goes the other way.
\cs_new_protected:Npn \__numodel_save_state:n #1
  {
    \seq_gset_eq:cN { g__numodel_ #1 _vars_seq      } \g_mvar_names_seq
    \seq_gset_eq:cN { g__numodel_ #1 _starts_seq    } \g_mvar_start_seq
    \seq_gset_eq:cN { g__numodel_ #1 _rules_seq     } \g_mrule_seq
    \seq_gset_eq:cN { g__numodel_ #1 _ruletypes_seq } \g_mrule_type_seq
    \seq_gset_eq:cN { g__numodel_ #1 _rulecalc_seq  } \g_mrule_calc_seq
    \int_gset:cn    { g__numodel_ #1 _rulecounter_int }
      { \int_use:N \g_mrule_counter_int }
    \tl_gset_eq:cN  { g__numodel_ #1 _stopexpr_tl   } \g_numodel_stop_expr_tl
    \int_gset:cn    { g__numodel_ #1 _steps_int     }
      { \int_use:N \g_numodel_steps_int }
  }

\cs_new_protected:Npn \__numodel_load_state:n #1
  {
    \seq_gset_eq:Nc \g_mvar_names_seq  { g__numodel_ #1 _vars_seq      }
    \seq_gset_eq:Nc \g_mvar_start_seq  { g__numodel_ #1 _starts_seq    }
    \seq_gset_eq:Nc \g_mrule_seq       { g__numodel_ #1 _rules_seq     }
    \seq_gset_eq:Nc \g_mrule_type_seq  { g__numodel_ #1 _ruletypes_seq }
    \seq_gset_eq:Nc \g_mrule_calc_seq  { g__numodel_ #1 _rulecalc_seq  }
    \int_gset:Nn \g_mrule_counter_int
      { \int_use:c { g__numodel_ #1 _rulecounter_int } }
    \tl_gset_eq:Nc \g_numodel_stop_expr_tl  { g__numodel_ #1 _stopexpr_tl }
    \int_gset:Nn \g_numodel_steps_int
      { \int_use:c { g__numodel_ #1 _steps_int } }
```

```
}

\cs_new_protected:Npn \__numodel_clear_state:
  {
    \seq_gclear:N  \g_mvar_names_seq
    \seq_gclear:N  \g_mvar_start_seq
    \seq_gclear:N  \g_mrule_seq
    \seq_gclear:N  \g_mrule_type_seq
    \seq_gclear:N  \g_mrule_calc_seq
    \int_gzero:N   \g_mrule_counter_int
    \tl_gclear:N   \g_numodel_stop_expr_tl
    \int_gzero:N   \g_numodel_steps_int
  }

% Initialise per-prefix backup variables (once, at \newmodelprefix).
\cs_new_protected:Npn \__numodel_init_backup:n #1
  {
    \seq_new:c  { g__numodel_ #1 _vars_seq      }
    \seq_new:c  { g__numodel_ #1 _starts_seq    }
    \seq_new:c  { g__numodel_ #1 _rules_seq     }
    \seq_new:c  { g__numodel_ #1 _ruletypes_seq }
    \seq_new:c  { g__numodel_ #1 _rulecalc_seq  }
    \int_new:c  { g__numodel_ #1 _rulecounter_int }
    \tl_new:c   { g__numodel_ #1 _stopexpr_tl   }
    \int_new:c  { g__numodel_ #1 _steps_int     }
  }

% Public commands for prefix management
\NewDocumentCommand{\newmodelprefix}{ m }
  {
    \typeout{NEWPREFIX~start:~#1}
    \seq_if_in:NnTF \g_numodel_prefixes_seq {#1}
      { \msg_error:nnn { numodel } { prefix-exists } {#1} }
      {
        \typeout{NEWPREFIX~save~current:~\g_numodel_current_prefix_tl}
        % Save current state under the previous prefix (if any)
        \tl_if_empty:NF \g_numodel_current_prefix_tl
          { \exp_args:NV \__numodel_save_state:n \g_numodel_current_prefix_tl }
        \typeout{NEWPREFIX~register}
        % Register new prefix + init backup vars + Lua state
        \seq_gput_right:Nn \g_numodel_prefixes_seq {#1}
        \typeout{NEWPREFIX~init_backup}
        \__numodel_init_backup:n {#1}
        \typeout{NEWPREFIX~lua_init}
        \directlua{ numodel.init_prefix("#1") }
        \typeout{NEWPREFIX~clear_state}
        % Clear current state and set the prefix
        \__numodel_clear_state:
        \typeout{NEWPREFIX~set_current}
        \tl_gset:Nn \g_numodel_current_prefix_tl {#1}
        \typeout{NEWPREFIX~done:~#1}
      }
  }

\NewDocumentCommand{\switchmodelprefix}{ m }
  {
    \seq_if_in:NnTF \g_numodel_prefixes_seq {#1}
      {
        % Save current state (if any), load the new one
        \tl_if_empty:NF \g_numodel_current_prefix_tl
          { \exp_args:NV \__numodel_save_state:n \g_numodel_current_prefix_tl }
        \__numodel_load_state:n {#1}
        \tl_gset:Nn \g_numodel_current_prefix_tl {#1}
      }
      { \msg_error:nnn { numodel } { prefix-unknown } {#1} }
  }
```

```
% Iterate over *all* variables from *all* prefixes.
% #1 = code that uses ##1 = full variable name (prefix + short).
% Public registration API for external tools such as worksheet.tex.
\NewDocumentCommand{\NumodelForEachVar}{ +m }
  {
    \seq_map_inline:Nn \g_numodel_prefixes_seq
      {
        \seq_map_inline:cn { g__numodel_ ##1 _vars_seq } {#1}
      }
  }

% Resolve the effective prefix for a command call.  Takes
% [prefix=<p>] from the keyval argument; otherwise falls back to the
% current prefix.  Stores the result in \l__numodel_eff_prefix_tl.
\keys_define:nn { numodel / cmd }
  {
    prefix        .tl_set:N = \l__numodel_cmd_prefix_tl,
    % \graphicmodel-only: temporary override of the global
    % diagram-style.  Cleared after every \graphicmodel call.
    diagram-style              .choice:,
    diagram-style / tight      .code:n =
      { \tl_set:Nn \l__numodel_cmd_diagstyle_tl { tight } },
    diagram-style / forrester  .code:n =
      { \tl_set:Nn \l__numodel_cmd_diagstyle_tl { forrester } },
    diagram-style / edu        .code:n =
      { \tl_set:Nn \l__numodel_cmd_diagstyle_tl { edu } },
    % \graphicmodel-only: temporary overrides of the corresponding
    % global flowarrow/valve/cloud keys.  Empty after every call.
    flowarrow-style     .code:n =
      { \__numodel_local_choice:Nnnn \l__numodel_cmd_flowarrow_tl
          { flowarrow-style } { hollow , filled } {#1} },
    valve-style         .code:n =
      { \__numodel_local_choice:Nnnn \l__numodel_cmd_valve_tl
          { valve-style } { valve , circle , edu } {#1} },
    flowarrow-cloud-tip .code:n =
      { \__numodel_local_choice:Nnnn \l__numodel_cmd_flowcloud_tl
          { flowarrow-cloud-tip } { true , false } {#1} },
    % \textmodel-only: temporary override of the global units bool.
    % Cleared after every \textmodel call.
    units                      .choice:,
    units / true               .code:n =
      { \tl_set:Nn \l__numodel_cmd_units_tl { true  } },
    units / false              .code:n =
      { \tl_set:Nn \l__numodel_cmd_units_tl { false } },
    % \textmodel-only: temporary override of the global tblrenv tl.
    % Cleared after every \textmodel call.
    tblrenv                    .choice:,
    tblrenv / tblr             .code:n =
      { \tl_set:Nn \l__numodel_cmd_tblrenv_tl { tblr } },
    tblrenv / longtblr         .code:n =
      { \tl_set:Nn \l__numodel_cmd_tblrenv_tl { longtblr } },
    tblrenv / talltblr         .code:n =
      { \tl_set:Nn \l__numodel_cmd_tblrenv_tl { talltblr } },
  }
\tl_new:N \l__numodel_cmd_diagstyle_tl
\tl_new:N \l__numodel_cmd_flowarrow_tl
\tl_new:N \l__numodel_cmd_valve_tl
\tl_new:N \l__numodel_cmd_flowcloud_tl
\tl_new:N \l__numodel_cmd_units_tl
\tl_new:N \l__numodel_cmd_tblrenv_tl

\cs_new_protected:Npn \__numodel_resolve_prefix:n #1
  {
    \tl_clear:N \l__numodel_cmd_prefix_tl
    \keys_set:nn { numodel / cmd } {#1}
```

```
    \tl_if_empty:NTF \l__numodel_cmd_prefix_tl
      { \tl_set_eq:NN \l__numodel_eff_prefix_tl \g_numodel_current_prefix_tl }
      { \tl_set_eq:NN \l__numodel_eff_prefix_tl \l__numodel_cmd_prefix_tl }
  }

% Execute code under a temporarily different prefix (via state swap).
% #1 = target prefix (tl, in variable)
% #2 = code
\cs_new_protected:Npn \__numodel_with_prefix:Nn #1 #2
  {
    \tl_if_eq:NNTF #1 \g_numodel_current_prefix_tl
      { #2 }    % already the current prefix; no swap needed
      {
        \tl_set_eq:NN \l__numodel_saved_prefix_tl \g_numodel_current_prefix_tl
        \exp_args:NV \switchmodelprefix #1
        #2
        \exp_args:NV \switchmodelprefix \l__numodel_saved_prefix_tl
      }
  }
\tl_new:N \l__numodel_saved_prefix_tl

% Build the full name = <prefix><shortname>.  Stored in
% \l__numodel_fullname_tl.
\cs_new_protected:Npn \__numodel_set_fullname:n #1
  {
    \tl_set:Ne \l__numodel_fullname_tl { \l__numodel_eff_prefix_tl #1 }
  }

% Keys for the optional argument of \mvar (grid position +
% initial-value aliases + prefix).
\tl_new:N \l__numodel_alias_tl
\tl_new:N \l__numodel_aliasleft_tl
\tl_new:N \l__numodel_aliasright_tl

\keys_define:nn { numodel / mvar }
  {
    prefix      .tl_set:N = \l__numodel_cmd_prefix_tl ,
    gridx       .tl_set:N = \l__numodel_gridx_tl ,
    gridy       .tl_set:N = \l__numodel_gridy_tl ,
    alias       .tl_set:N = \l__numodel_alias_tl ,
    aliasleft   .tl_set:N = \l__numodel_aliasleft_tl ,
    aliasright  .tl_set:N = \l__numodel_aliasright_tl ,
    flowarrow-cloud-tip .code:n =
      { \__numodel_local_choice:Nnnn \l__numodel_mvar_flowcloud_tl
          { flowarrow-cloud-tip } { true , false } {#1} },
  }
\tl_new:N \l__numodel_mvar_flowcloud_tl

% ====================================================================
% \mvar[keys]{name}{display}{startvalue}{unit}{sig}{type}
% ====================================================================
% Declare a model variable.  Generates the following macros:
%
%   \<name>      -- numeric value (\edef + \fpeval), or warning if empty
%   \<name>text  -- display name for the model table (e.g. F_{res})
%   \<name>unit  -- SI unit (e.g. kN)
%   \<name>unitraw - raw SI unit (e.g. \kilo\N)
%   \<name>sign  -- number of significant figures
%   \<name>type  -- variable type: stock, constant, aux, system
%                   (Dutch synonyms voorraad/constante/hulp/systeem
%                   are normalised to the canonical English form)
%   \<name>coord -- coordinate list (empty; filled by \computemodel)
%   \<name>gridx -- x position in the graphic model (-1 = auto)
%   \<name>gridy -- y position in the graphic model (-1 = auto)
%   \<name>num   -- number with significance (via \num)
%   \<name>qty   -- number + unit (via \qty)
```

```
%   \<name>pre   -- engineering-prefix notation (via \qty)
%   \<name>alias       -- replaces the whole initial-value cell
%                         (empty = default)
%   \<name>aliasleft   -- replaces the left symbol in the initial value
%                         (empty = default)
%   \<name>aliasright  -- replaces the right number in the initial value
%                         (empty = default)
%
% Keys accepted in [#1]:
%   gridx, gridy          -- position in the graphic model
%   alias                 -- replace the entire initial-value cell
%                            (in math mode)
%   aliasleft             -- replace just the left symbol
%   aliasright            -- replace just the right value
%
% The base value is registered in \g_defqty_names_seq so that
% \includeimage expands it automatically.
%
% With an empty start value (#3 blank), the base value is not
% numerically defined; using it issues a warning.  Useful for
% auxiliary variables computed by \mrule.
%
% Example:
%   \mvar{modM}{m}{80}{\kg}{2}{constant}
%   \mvar{modFres}{F_{res}}{}{\N}{2}{aux}
%   \mvar[aliasright=\cdots]{modX}{x}{0}{\m}{3}{stock}
%   \mvar[alias={x \text{?}}]{modX}{x}{0}{\m}{3}{stock}

% Normalise the variable type (sixth \mvar argument) to its
% canonical English form.  Accepts the canonical names
% {stock, constant, aux, system} and the Dutch synonyms
% {voorraad, constante, hulp, systeem}.  An unknown value is left
% as-is and a warning is issued.  The result is returned through
% \l__numodel_type_tl.
\tl_new:N \l__numodel_type_tl
\cs_new_protected:Npn \__numodel_normalize_type:n #1
  {
    \str_case:nnF {#1}
      {
        { stock     } { \tl_set:Nn \l__numodel_type_tl { stock    } }
        { constant  } { \tl_set:Nn \l__numodel_type_tl { constant } }
        { aux       } { \tl_set:Nn \l__numodel_type_tl { aux      } }
        { system    } { \tl_set:Nn \l__numodel_type_tl { system   } }
        { voorraad  } { \tl_set:Nn \l__numodel_type_tl { stock    } }
        { constante } { \tl_set:Nn \l__numodel_type_tl { constant } }
        { hulp      } { \tl_set:Nn \l__numodel_type_tl { aux      } }
        { systeem   } { \tl_set:Nn \l__numodel_type_tl { system   } }
      }
      {
        \msg_warning:nne { numodel } { unknown-type } {#1}
        \tl_set:Nn \l__numodel_type_tl {#1}
      }
  }

\makeatletter
\NewDocumentCommand{\mvar}{ O{} m m m m m m }{%
  \typeout{MVAR~start:~#2}
  % Parse optional keys (prefix, gridx, gridy, alias, aliasleft, aliasright,
  % flowarrow-cloud-tip)
  \tl_set:Nn \l__numodel_gridx_tl { -1 }
  \tl_set:Nn \l__numodel_gridy_tl { -1 }
  \tl_clear:N \l__numodel_alias_tl
  \tl_clear:N \l__numodel_aliasleft_tl
  \tl_clear:N \l__numodel_aliasright_tl
  \tl_clear:N \l__numodel_cmd_prefix_tl
  \tl_clear:N \l__numodel_mvar_flowcloud_tl
```

```
  \typeout{MVAR~before-keys-set}
  \keys_set:nn { numodel / mvar } {#1}
  \typeout{MVAR~after-keys-set}
  \__numodel_resolve_eff_prefix:
  \typeout{MVAR~eff:~\l__numodel_eff_prefix_tl}
  \__numodel_with_prefix:Nn \l__numodel_eff_prefix_tl
    { \__numodel_mvar_body:nnnnnn {#2}{#3}{#4}{#5}{#6}{#7} }%
  \typeout{MVAR~done:~#2}
}
\makeatother

% Resolve the effective prefix from \l__numodel_cmd_prefix_tl.
\cs_new_protected:Npn \__numodel_resolve_eff_prefix:
  {
    \tl_if_empty:NTF \l__numodel_cmd_prefix_tl
      { \tl_set_eq:NN \l__numodel_eff_prefix_tl \g_numodel_current_prefix_tl }
      { \tl_set_eq:NN \l__numodel_eff_prefix_tl \l__numodel_cmd_prefix_tl }
  }

% The \mvar body runs in the context of the right prefix (live state
% has already been swapped).
\cs_new_protected:Npn \__numodel_mvar_body:nnnnnn #1 #2 #3 #4 #5 #6
  {
    % Guard: current prefix must not be empty (\newmodelprefix required)
    \tl_if_empty:NT \g_numodel_current_prefix_tl
      { \msg_error:nn { numodel } { no-prefix } }
    % Full name = current prefix + short name (#1)
    \tl_set:Ne \l__numodel_fullname_tl { \g_numodel_current_prefix_tl #1 }
    \typeout{MVAR~body~fullname:~\l__numodel_fullname_tl}
    % Register in the per-prefix seq.  The defqty registration is
    % optional: it only fires when the user has loaded the project-
    % specific 'defqty' system (used for worksheet expansion).
    % Standalone, the package works without that seq.
    \seq_gput_right:NV \g_mvar_names_seq   \l__numodel_fullname_tl
    \cs_if_exist:NT \g_defqty_names_seq
      { \seq_gput_right:NV \g_defqty_names_seq \l__numodel_fullname_tl }
    \directlua{ numodel.register(
      "\g_numodel_current_prefix_tl",
      "\l__numodel_fullname_tl") }
    \cs_if_exist:cT { \l__numodel_fullname_tl }
      { \msg_warning:nne { numodel } { redef } { \l__numodel_fullname_tl } }
    \tl_if_blank:nTF {#3}
      { \cs_gset:cpe { \l__numodel_fullname_tl } { \fp_eval:n { 0 } } }
      {
        \cs_gset:cpe { \l__numodel_fullname_tl } { \fp_eval:n {#3} }
        \seq_gput_right:NV \g_mvar_start_seq \l__numodel_fullname_tl
      }
    \cs_gset:cpn { \l__numodel_fullname_tl text }   {#2}
    \cs_gset:cpn { \l__numodel_fullname_tl unit }   { \unit{#4} }
    \cs_gset:cpn { \l__numodel_fullname_tl unitraw } {#4}
    \cs_gset:cpn { \l__numodel_fullname_tl sign }   {#5}
    \__numodel_normalize_type:n {#6}
    \cs_gset:cpe { \l__numodel_fullname_tl type }
      { \tl_use:N \l__numodel_type_tl }
    \cs_gset:cpn { \l__numodel_fullname_tl min }    { inf }
    \cs_gset:cpn { \l__numodel_fullname_tl max }    { -inf }
    \cs_gset:cpe { \l__numodel_fullname_tl gridx }  { \tl_use:N \l__numodel_gridx_tl }
    \cs_gset:cpe { \l__numodel_fullname_tl gridy }  { \tl_use:N \l__numodel_gridy_tl }
    % Preserve the original user input so that \__numodel_build_graphic:
    % can reset auto-placed positions to -1 on a second invocation.
    \cs_gset:cpe { \l__numodel_fullname_tl gridxinit } { \tl_use:N \l__numodel_gridx_tl }
    \cs_gset:cpe { \l__numodel_fullname_tl gridyinit } { \tl_use:N \l__numodel_gridy_tl }
    % Pillar A - Lua-side meta for compute_layout (additive in A1).
    % Detokenize text so that \luaescapestring works on bare chars.
    \tl_set:Ne \l__numodel_scratch_tl { \detokenize {#2} }
    \directlua{ numodel.set_meta(
```

```
      "\g_numodel_current_prefix_tl",
      "\l__numodel_fullname_tl",
      { type  = "\tl_use:N \l__numodel_type_tl",
        text  = "\luaescapestring{\l__numodel_scratch_tl}",
        gridx = \tl_use:N \l__numodel_gridx_tl,
        gridy = \tl_use:N \l__numodel_gridy_tl }) }
    \cs_gset:cpe { \l__numodel_fullname_tl alias }      { \exp_not:V \l__numodel_alias_tl }
    \cs_gset:cpe { \l__numodel_fullname_tl aliasleft }  { \exp_not:V \l__numodel_aliasleft_tl }
    \cs_gset:cpe { \l__numodel_fullname_tl aliasright } { \exp_not:V \l__numodel_aliasright_tl }
    \cs_gset:cpe { \l__numodel_fullname_tl flowcloud }  { \exp_not:V \l__numodel_mvar_flowcloud_tl }
    \tl_if_blank:nF {#3}
      {
        \exp_args:NV \NumodelDefNumCs \l__numodel_fullname_tl {\fp_eval:n {#3}} {#5}
        \exp_args:NV \NumodelDefQtyCs \l__numodel_fullname_tl {\fp_eval:n {#3}} {#5} {#4}
        \exp_args:NV \NumodelDefPreCs \l__numodel_fullname_tl {\fp_eval:n {#3}} {#5} {#4}
      }
  }

% ======================================================================
% Warning messages
% ======================================================================

\msg_new:nnn { numodel } { redef }
  { Model~variable~'#1'~is~being~redefined. }
\msg_new:nnn { numodel } { empty-use }
  { Model~variable~'#1'~has~no~start~value~and~is~used~before~assignment. }
\msg_new:nnn { numodel } { maxiter }
  { computemodel~stopped~after~\int_use:N \g_numodel_maxiter_int ~iterations~
    (safety~limit).~Check~stop~condition. }
\msg_new:nnn { numodel } { no-stop }
  { computemodel:~no~stop~condition~defined.~Use~\token_to_str:N \mstop\space
    before~\token_to_str:N \computemodel. }
\msg_new:nnn { numodel } { prefix-exists }
  { Model~prefix~'#1'~is~already~registered.~
    Use~\token_to_str:N \switchmodelprefix\space to~switch~to~an~existing~prefix. }
\msg_new:nnn { numodel } { prefix-unknown }
  { Model~prefix~'#1'~is~not~registered.~
    Use~\token_to_str:N \newmodelprefix\space to~create~it~first. }
\msg_new:nnn { numodel } { no-prefix }
  { No~current~model~prefix.~
    Call~\token_to_str:N \newmodelprefix{<name>}\space before~using~model~commands. }
\msg_new:nnn { numodel } { unknown-type }
  { Unknown~variable~type~'#1'.~
    Use~one~of~stock,~constant,~aux,~system~
    (or~the~Dutch~aliases~voorraad,~constante,~hulp,~systeem). }
\msg_new:nnn { numodel } { unit-mismatch }
  { Variable~'#1'~has~a~different~unit~from~the~first~variable~
    (expected~'#2').~Skipping~it~in~\token_to_str:N \diagrammodel\space
    so~the~remaining~series~share~a~common~y-axis. }

% ======================================================================
% Display helpers
% ======================================================================
% Translate computational expressions into syllabus-style display.
%
% Automatic translations:
%   \modXxx       -> display name (via \<name>text)
%   *             -> \cdot
%   >=  <=        -> \geqslant  \leqslant
%   sign(...)     -> SIGN(...)   / Teken(...) in coachtaal
%   abs(...)      -> ABS(...)    / Abs(...)   in coachtaal
%   sqrt(...)     -> SQRT(...)   / Sqrt(...)   in coachtaal
%   exp(...)      -> EXP(...)    / Exp(...)    in coachtaal
%   ln(...)       -> LN(...)     / Ln(...)     in coachtaal
%   sin(...)      -> SIN(...)    / Sin(...)    in coachtaal
%   cos(...)      -> COS(...)    / Cos(...)    in coachtaal
```

```
%   tan(...)      -> TAN(...)    / Tan(...)    in coachtaal
%   asin(...)     -> ARCSIN(...) / Arcsin(...) in coachtaal
%   acos(...)     -> ARCCOS(...) / Arccos(...) in coachtaal
%   &&            -> AND          / EN          in coachtaal
%   ||            -> OR           / OF          in coachtaal
%   cond ? a : b  -> IF cond THEN ... ELSE ... ENDIF (or coachtaal eq.)

\tl_new:N \l__numodel_display_tl
\tl_new:N \l__numodel_lhs_tl
\tl_new:N \l__numodel_rhs_tl
\tl_new:N \l__numodel_cond_tl
\tl_new:N \l__numodel_true_tl
\tl_new:N \l__numodel_false_tl

\cs_new_protected:Npn \__numodel_vars_to_display:N #1
  {
    \typeout{VTD~input:~\tl_to_str:N #1}
    \typeout{VTD~seq:~\seq_use:Nn \g_mvar_names_seq {|}}
    % Variable names -> display names
    \seq_map_inline:Nn \g_mvar_names_seq
      {
        \typeout{VTD~iter:~##1}
        \cs_if_exist:cT { ##1 text }
          {
            \tl_set:Ne \l_tmpa_tl { \use:c { ##1 text } }
            \typeout{VTD~replace~\c{##1}~with~\l_tmpa_tl}
            \regex_replace_all:nnN { \c{##1} } { \u{l_tmpa_tl} } #1
          }
      }
    % Arithmetic operators
    \regex_replace_all:nnN { \* } { \c{cdot} \x{20} } #1
    \regex_replace_all:nnN { >= } { \c{geqslant} \x{20} } #1
    \regex_replace_all:nnN { <= } { \c{leqslant} \x{20} } #1
    % Functions (syllabus notation).  Order matters: asin/acos must
    % match before sin/cos, otherwise the inner sin/cos gets replaced
    % first and the outer arc- prefix is left dangling.
    \regex_replace_all:nnN { sign \( }
      { \c{text}\cB\{ \u{g__numodel_kw_sign_tl}\cE\}( } #1
    \regex_replace_all:nnN { abs \( }
      { \c{text}\cB\{ \u{g__numodel_kw_abs_tl}\cE\}( } #1
    \regex_replace_all:nnN { sqrt \( }
      { \c{text}\cB\{ \u{g__numodel_kw_sqrt_tl}\cE\}( } #1
    \regex_replace_all:nnN { exp \( }
      { \c{text}\cB\{ \u{g__numodel_kw_exp_tl}\cE\}( } #1
    \regex_replace_all:nnN { ln \( }
      { \c{text}\cB\{ \u{g__numodel_kw_ln_tl}\cE\}( } #1
    \regex_replace_all:nnN { asin \( }
      { \c{text}\cB\{ \u{g__numodel_kw_asin_tl}\cE\}( } #1
    \regex_replace_all:nnN { acos \( }
      { \c{text}\cB\{ \u{g__numodel_kw_acos_tl}\cE\}( } #1
    \regex_replace_all:nnN { sin \( }
      { \c{text}\cB\{ \u{g__numodel_kw_sin_tl}\cE\}( } #1
    \regex_replace_all:nnN { cos \( }
      { \c{text}\cB\{ \u{g__numodel_kw_cos_tl}\cE\}( } #1
    \regex_replace_all:nnN { \b tan \( }
      { \c{text}\cB\{ \u{g__numodel_kw_tan_tl}\cE\}( } #1
    % Logical operators (syllabus notation)
    \regex_replace_all:nnN { \&\& }
      { \c{text}\cB\{\u{g__numodel_kw_and_tl}\cE\} } #1
    \regex_replace_all:nnN { \|\| }
      { \c{text}\cB\{\u{g__numodel_kw_or_tl}\cE\} } #1
  }

% Ternary detection: cond ? true_expr : false_expr
% Converted to: IF cond THEN lhs = true ELSE lhs = false ENDIF
% (or the coachtaal equivalent).  Supports only flat (non-nested)
```

```
% ternaries.
\bool_new:N \l__numodel_is_ternary_bool

\cs_new_protected:Npn \__numodel_parse_ternary:nN #1 #2
  {
    \bool_set_false:N \l__numodel_is_ternary_bool
    \regex_match:nnT { (.+) \? (.+) \: (.+) } {#1}
      {
        \bool_set_true:N \l__numodel_is_ternary_bool
        \tl_set:Nn \l__numodel_cond_tl {#1}
        \regex_replace_once:nnN { \s*(.+?) \s* \? .+ } { \1 } \l__numodel_cond_tl
        \tl_set:Nn \l__numodel_true_tl {#1}
        \regex_replace_once:nnN { .+? \? \s* (.+?) \s* \: .+ } { \1 } \l__numodel_true_tl
        \tl_set:Nn \l__numodel_false_tl {#1}
        \regex_replace_once:nnN { .+ \: \s* (.+?) \s* \Z } { \1 } \l__numodel_false_tl
        \__numodel_vars_to_display:N \l__numodel_cond_tl
        \__numodel_vars_to_display:N \l__numodel_true_tl
        \__numodel_vars_to_display:N \l__numodel_false_tl
      }
  }

% ======================================================================
% \mrule[keys]{varname}{calculation}
% ======================================================================
% Declare a model rule.
%
%   #1 (star)      -- starred variant: multiline IF/THEN/ELSE/ENDIF
%                     for ternary expressions
%   #2 (optional)  -- keys: alias, aliasleft, aliasright
%                       alias       -- replaces the whole display row
%                       aliasleft   -- replaces the left-hand side
%                                      (symbol)
%                       aliasright  -- replaces the right-hand side
%                                      (calculation)
%                     Use \cdots for omitted parts.
%   #3             -- name (string) of the left-hand variable
%   #4             -- calculation (with \modXxx macros and \fpeval
%                     syntax)
%
% With alias or aliasright the ternary logic is skipped (display
% shows the alias content verbatim, not IF/THEN/ELSE).  Execution
% always uses the calculation from #4.
%
% The calculation accepts all \fpeval operators:
%   +, -, *, /, ^, sign(), abs(), sin(), cos(), sqrt(), round()
%   Ternary: cond ? expr_true : expr_false
%   Logical: && (AND), || (OR), comparisons (<, >, <=, >=)
%
% The rule is stored for both display (\textmodel) and execution
% (\computemodel).
%
% Examples:
%   \mrule{modFres}{\modM * \modA}
%   \mrule{modFw}{sign(\modV) * \modK * \modV^2}
%   \mrule{modA}{(\modT < \modTq) || (\modT > \modTdq) ? \modA + \modDa : \modA - \modDa}
%   \mrule[aliasright=\cdots]{modT}{\modT + \modDt}
%   \mrule[aliasleft=a_x]{modAx}{\modFgx / \modM}
%   \mrule[alias={\text{(hidden)}}]{modAy}{\modFgy / \modM}

\NewDocumentCommand{\mrule}{ s O{} m m }{%
  \typeout{MRULE~start:~#3}
  \bool_set_false:N \l__numodel_is_ternary_bool
  % Parse keys (prefix, alias, aliasleft, aliasright; gridx/gridy ignored)
  \tl_clear:N \l__numodel_alias_tl
  \tl_clear:N \l__numodel_aliasleft_tl
  \tl_clear:N \l__numodel_aliasright_tl
```

```
  \tl_clear:N \l__numodel_cmd_prefix_tl
  \keys_set:nn { numodel / mvar } {#2}
  \__numodel_resolve_eff_prefix:
  \__numodel_with_prefix:Nn \l__numodel_eff_prefix_tl
    { \__numodel_mrule_body:nnn {#1}{#3}{#4} }%
  \typeout{MRULE~done:~#3}
}

\cs_new_protected:Npn \__numodel_mrule_body:nnn #1 #2 #3
  {
    \int_gincr:N \g_mrule_counter_int
    \tl_set:Ne \l__numodel_fullname_tl { \g_numodel_current_prefix_tl #2 }
    % Store execution data (target = full name)
    \seq_gput_right:Nx \g_mrule_calc_seq
      { { \l__numodel_fullname_tl } { \exp_not:n {#3} } }
    % Left-hand side: aliasleft or default display name
    \tl_if_blank:VTF \l__numodel_aliasleft_tl
      { \tl_set:Ne \l__numodel_lhs_tl { \use:c { \l__numodel_fullname_tl text } } }
      { \tl_set_eq:NN \l__numodel_lhs_tl \l__numodel_aliasleft_tl }
    % Generate display
    \tl_if_blank:VTF \l__numodel_alias_tl
      {
        \tl_if_blank:VTF \l__numodel_aliasright_tl
          {
            % Automatic display generation (ternary or normal)
            \__numodel_parse_ternary:nN {#3} \l__numodel_display_tl
            \bool_if:NTF \l__numodel_is_ternary_bool
              {
                \IfBooleanTF {#1}
                  {
                    % Starred ternary -> multiline IF/THEN/ELSE/ENDIF
                    \tl_set:Ne \l__numodel_display_tl
                      {
                        \__numodel_kwt:n {if}
                        \exp_not:V \l__numodel_cond_tl
                        \__numodel_kwt:n {then_nl}
                      }
                    \seq_gput_right:NV \g_mrule_seq \l__numodel_display_tl
                    \seq_gput_right:Nn \g_mrule_type_seq { rule }
                    \tl_set:Ne \l__numodel_display_tl
                      {
                        \exp_not:n { \quad }
                        \exp_not:V \l__numodel_lhs_tl \exp_not:n { \,=\, }
                        \exp_not:V \l__numodel_true_tl
                      }
                    \seq_gput_right:NV \g_mrule_seq \l__numodel_display_tl
                    \seq_gput_right:Nn \g_mrule_type_seq { cont }
                    \seq_gput_right:Ne \g_mrule_seq { \__numodel_kwt:n {else_nl} }
                    \seq_gput_right:Nn \g_mrule_type_seq { cont }
                    \tl_set:Ne \l__numodel_display_tl
                      {
                        \exp_not:n { \quad }
                        \exp_not:V \l__numodel_lhs_tl \exp_not:n { \,=\, }
                        \exp_not:V \l__numodel_false_tl
                      }
                    \seq_gput_right:NV \g_mrule_seq \l__numodel_display_tl
                    \seq_gput_right:Nn \g_mrule_type_seq { cont }
                    \seq_gput_right:Ne \g_mrule_seq { \__numodel_kwt:n {endif_nl} }
                    \seq_gput_right:Nn \g_mrule_type_seq { cont }
                  }
                  {
                    % Unstarred ternary -> single-line
                    \tl_set:Ne \l__numodel_display_tl
                      {
                        \__numodel_kwt:n {if}
                        \exp_not:V \l__numodel_cond_tl
```

```
                        \__numodel_kwt:n {then}
                        \exp_not:V \l__numodel_lhs_tl \exp_not:n { \,=\, }
                        \exp_not:V \l__numodel_true_tl
                        \__numodel_kwt:n {else}
                        \exp_not:V \l__numodel_lhs_tl \exp_not:n { \,=\, }
                        \exp_not:V \l__numodel_false_tl
                        \__numodel_kwt:n {endif}
                      }
                  }
              }
              {
                % Ordinary assignment: lhs = rhs
                \tl_set:Nn \l__numodel_rhs_tl {#3}
                \__numodel_vars_to_display:N \l__numodel_rhs_tl
                \typeout{MRULE~rhs~after:~\tl_to_str:N \l__numodel_rhs_tl}
                \tl_set:Ne \l__numodel_display_tl
                  {
                    \exp_not:V \l__numodel_lhs_tl \exp_not:n { \,=\, }
                    \exp_not:V \l__numodel_rhs_tl
                  }
                \typeout{MRULE~display:~\tl_to_str:N \l__numodel_display_tl}
              }
          }
          {
            % aliasright: lhs = aliasright (no ternary processing)
            \tl_set:Ne \l__numodel_display_tl
              {
                \exp_not:V \l__numodel_lhs_tl \exp_not:n { \,=\, }
                \exp_not:V \l__numodel_aliasright_tl
              }
          }
      }
      {
        % alias: replace the whole row
        \tl_set:Ne \l__numodel_display_tl { \exp_not:V \l__numodel_alias_tl }
      }
    % For starred ternary everything is already pushed above
    \bool_if:nF { \l__numodel_is_ternary_bool && #1 }
      {
        \seq_gput_right:NV \g_mrule_seq \l__numodel_display_tl
        \seq_gput_right:Nn \g_mrule_type_seq { rule }
      }
    % Pillar A - Lua-side rule registration.  Detokenize the raw
    % expression so Lua sees the macro names, not the evaluated
    % fp values.  Lua handles both the dependency extraction
    % (build_deps) and the flow detection (classify_flows) at
    % \graphicmodel time.
    \tl_set:Ne \l__numodel_scratch_tl { \detokenize {#3} }
    \directlua{ numodel.add_rule(
      "\g_numodel_current_prefix_tl",
      "\l__numodel_fullname_tl",
      "\luaescapestring{\l__numodel_scratch_tl}",
      "\bool_if:NTF \l__numodel_is_ternary_bool { ternary } { calc }") }
  }

% ====================================================================
% \mruletext{free text}
% ====================================================================
% Adds a free-text row in the model table.  Useful for structure
% that does not fit as \mrule, e.g.:
%   \mruletext{\text{IF } t < T/4 \text{ THEN}}
%   \mruletext{\quad a = a + da}
%   \mruletext{\text{ENDIF}}
%
% NOT executed by \computemodel (display only).

```

```
\NewDocumentCommand{\mruletext}{ O{} m }{%
  \tl_clear:N \l__numodel_cmd_prefix_tl
  \keys_set:nn { numodel / cmd } {#1}
  \__numodel_resolve_eff_prefix:
  \__numodel_with_prefix:Nn \l__numodel_eff_prefix_tl
    {
      \int_gincr:N \g_mrule_counter_int
      \seq_gput_right:Nn \g_mrule_seq {#2}
      \seq_gput_right:Nn \g_mrule_type_seq { rule }
    }
}

% ====================================================================
% \mstop[keys]{condition}
% \mstop*[keys]{condition}   (star reserved; currently identical)
% ====================================================================
% Declare the model's stop condition.
% Display: "IF condition THEN STOP ENDIF"
% Execution: the model stops when the condition is true (evaluates to 1).
%
% The condition uses \fpeval syntax with \mvar variables:
%   \mstop{\modT >= \modTmax}
%   \mstop{\modV <= 0}
%
% Supported keys:
%   alias={...}  -- replaces the whole display row (execution
%                   remains intact)
% aliasleft/aliasright are not (yet) supported for \mstop.

\tl_new:N \l__numodel_stop_tl
\NewDocumentCommand{\mstop}{ s O{} m }{%
  \typeout{MSTOP~start}
  % Parse keys (only alias is honoured; prefix via key resolver)
  \tl_clear:N \l__numodel_alias_tl
  \tl_clear:N \l__numodel_aliasleft_tl
  \tl_clear:N \l__numodel_aliasright_tl
  \tl_clear:N \l__numodel_cmd_prefix_tl
  \keys_set:nn { numodel / mvar } {#2}
  \__numodel_resolve_eff_prefix:
  \__numodel_with_prefix:Nn \l__numodel_eff_prefix_tl
    { \__numodel_mstop_body:n {#3} }%
  \typeout{MSTOP~done}
}

\cs_new_protected:Npn \__numodel_mstop_body:n #1
  {
    \int_gincr:N \g_mrule_counter_int
    % Store expression for execution (always the actual condition)
    \tl_gset:Nn \g_numodel_stop_expr_tl {#1}
    % Generate display
    \tl_if_blank:VTF \l__numodel_alias_tl
      {
        \tl_set:Nn \l__numodel_stop_tl {#1}
        \__numodel_vars_to_display:N \l__numodel_stop_tl
        \tl_set:Ne \l__numodel_display_tl
          {
            \__numodel_kwt:n {if}
            \exp_not:V \l__numodel_stop_tl
            \__numodel_kwt:n {then_nl}
            \__numodel_kwt:n {stop}
            \__numodel_kwt:n {endif_nl}
          }
      }
      {
        \tl_set:Ne \l__numodel_display_tl { \exp_not:V \l__numodel_alias_tl }
      }
```

```
    \seq_gput_right:NV \g_mrule_seq \l__numodel_display_tl
    \seq_gput_right:Nn \g_mrule_type_seq { rule }
  }

% ======================================================================
% \textmodel
% ======================================================================
% Builds a tabular with numbered model rules on the left and initial
% values on the right.  Can be placed anywhere, e.g. in a subfigure
% next to a graphic model.

\tl_new:N  \g__numodel_table_tl
\int_new:N \l__numodel_row_int
\int_new:N \l__numodel_dispnum_int
\int_new:N \l__numodel_startrow_int
\int_new:N \l__numodel_numrules_int
\int_new:N \l__numodel_numstarts_int

% Emit one initial-value cell with alias-key support.
%   - \<name>alias not empty       -> replaces whole cell (inside $...$)
%   - \<name>aliasleft not empty   -> replaces left symbol, else
%                                     \<name>text
%   - \<name>aliasright not empty  -> replaces right value, else
%                                     \<name>qty (units=true) or
%                                     \<name>num (units=false)
\cs_new_protected:Npn \__numodel_emit_startcell:n #1
  {
    \tl_if_blank:eTF { \use:c { #1 alias } }
      {
        \tl_gput_right:Nn \g__numodel_table_tl { $ }
        \tl_if_blank:eTF { \use:c { #1 aliasleft } }
          {
            \tl_gput_right:Ne \g__numodel_table_tl
              { \exp_not:e { \use:c { #1 text } } }
          }
          {
            \tl_gput_right:Ne \g__numodel_table_tl
              { \exp_not:e { \use:c { #1 aliasleft } } }
          }
        \tl_gput_right:Nn \g__numodel_table_tl { = }
        \tl_if_blank:eTF { \use:c { #1 aliasright } }
          {
            % Emit \<name>qty or \<name>num as a token name without
            % expanding it; siunitx macros must only expand at
            % \tl_use:N time, in the correct tabular context.
            \bool_if:NTF \g__numodel_units_bool
              {
                \tl_gput_right:Nx \g__numodel_table_tl
                  { \exp_not:c { #1 qty } }
              }
              {
                \tl_gput_right:Nx \g__numodel_table_tl
                  { \exp_not:c { #1 num } }
              }
          }
          {
            \tl_gput_right:Ne \g__numodel_table_tl
              { \exp_not:e { \use:c { #1 aliasright } } }
          }
        \tl_gput_right:Nn \g__numodel_table_tl { $ }
      }
      {
        \tl_gput_right:Nn \g__numodel_table_tl { $ }
        \tl_gput_right:Ne \g__numodel_table_tl
          { \exp_not:e { \use:c { #1 alias } } }
        \tl_gput_right:Nn \g__numodel_table_tl { $ }
```

```
    }
  }

\cs_new_protected:Npn \__numodel_build_table:
  {
    \typeout{BUILD_TABLE~rules:~\seq_use:Nn \g_mrule_seq {|}}
    \typeout{BUILD_TABLE~types:~\seq_use:Nn \g_mrule_type_seq {|}}
    \int_set:Nn \l__numodel_numrules_int  { \seq_count:N \g_mrule_seq }
    \int_set:Nn \l__numodel_numstarts_int { \seq_count:N \g_mvar_start_seq }
    \typeout{BUILD_TABLE~numrules:~\int_use:N \l__numodel_numrules_int}
    \typeout{BUILD_TABLE~numstarts:~\int_use:N \l__numodel_numstarts_int}
    \int_zero:N \l__numodel_row_int
    \int_zero:N \l__numodel_dispnum_int
    \int_zero:N \l__numodel_startrow_int
    \tl_gset:Nn \g__numodel_table_tl { \begin }
    \tl_gput_right:Ne \g__numodel_table_tl
      { { \tl_use:N \g__numodel_tblrenv_tl } }
    \tl_gput_right:Nn \g__numodel_table_tl
      { [theme={numodel}]{colspec={r|l|l}, rowhead=1} & \textbf }
    \tl_gput_right:Ne \g__numodel_table_tl
      { { \__numodel_kw:n {th_model} } }
    \tl_gput_right:Nn \g__numodel_table_tl
      { & \textbf }
    \tl_gput_right:Ne \g__numodel_table_tl
      { { \__numodel_kw:n {th_initvals} } }
    \tl_gput_right:Nn \g__numodel_table_tl
      { \\ \hline }
    \typeout{BUILD_TABLE~before-step~table:~\tl_to_str:N \g__numodel_table_tl}
    \int_step_inline:nn { \l__numodel_numrules_int }
      {
        \typeout{BUILD_TABLE~iter~row:~\int_use:N \l__numodel_row_int}
        \int_incr:N \l__numodel_row_int
        \int_incr:N \l__numodel_startrow_int
        \typeout{BUILD_TABLE~row:~\int_use:N \l__numodel_row_int~type:~\seq_item:Nn \g_mrule_type_seq
        % Check type: rule -> show number, cont -> blank
        \str_if_eq:eeTF
          { \seq_item:Nn \g_mrule_type_seq { \l__numodel_row_int } }
          { cont }
          {
            % Continuation rows: no number
            \tl_gput_right:Ne \g__numodel_table_tl
              {
                \exp_not:n { \rule{0pt}{2.6ex} }
                \exp_not:n { & $ }
                \exp_not:e { \seq_item:Nn \g_mrule_seq { \l__numodel_row_int } }
                \exp_not:n { $ & }
              }
          }
          {
            % Numbered row
            \int_incr:N \l__numodel_dispnum_int
            \typeout{BUILD_TABLE~emit~rule~num:~\int_use:N \l__numodel_dispnum_int~content:~\seq_item
            \tl_gput_right:Ne \g__numodel_table_tl
              {
                \exp_not:n { \rule{0pt}{2.6ex} }
                \int_use:N \l__numodel_dispnum_int
                \exp_not:n { & $ }
                \exp_not:e { \seq_item:Nn \g_mrule_seq { \l__numodel_row_int } }
                \exp_not:n { $ & }
              }
            \typeout{BUILD_TABLE~after-emit~table~tail:~\tl_tail:N \g__numodel_table_tl}
          }
        % Initial-value column (independent of rule/cont)
        \int_compare:nNnTF { \l__numodel_startrow_int } > { \l__numodel_numstarts_int }
          {
            \tl_gput_right:Nn \g__numodel_table_tl { \\ }
```

```
          }
          {
            \exp_args:Ne \__numodel_emit_startcell:n
              { \seq_item:Nn \g_mvar_start_seq { \l__numodel_startrow_int } }
            \tl_gput_right:Nn \g__numodel_table_tl { \\ }
          }
      }
    \int_while_do:nNnn { \l__numodel_startrow_int } < { \l__numodel_numstarts_int }
      {
        \int_incr:N \l__numodel_startrow_int
        \tl_gput_right:Nn \g__numodel_table_tl { & & }
        \exp_args:Ne \__numodel_emit_startcell:n
          { \seq_item:Nn \g_mvar_start_seq { \l__numodel_startrow_int } }
        \tl_gput_right:Nn \g__numodel_table_tl { \\ }
      }
    \tl_gput_right:Nn \g__numodel_table_tl { \end }
    \tl_gput_right:Ne \g__numodel_table_tl
      { { \tl_use:N \g__numodel_tblrenv_tl } }
  }

\NewDocumentCommand{\textmodel}{ O{} }{%
  \typeout{TEXTMODEL~start}
  \tl_clear:N \l__numodel_cmd_prefix_tl
  \tl_clear:N \l__numodel_cmd_units_tl
  \tl_clear:N \l__numodel_cmd_tblrenv_tl
  \keys_set:nn { numodel / cmd } {#1}
  \__numodel_resolve_eff_prefix:
  % Temporary units override: save the global value, apply the key
  % value before build, restore afterwards.  This way
  % \textmodel[units=false] flips one render without touching the
  % global \numodelsetup state.
  \bool_set_eq:NN \l__numodel_saved_units_bool \g__numodel_units_bool
  \tl_if_empty:NF \l__numodel_cmd_units_tl
    {
      \str_if_eq:VnTF \l__numodel_cmd_units_tl { true }
        { \bool_gset_true:N  \g__numodel_units_bool }
        { \bool_gset_false:N \g__numodel_units_bool }
    }
  % Same dance for tblrenv: save, apply per-call override, restore.
  \tl_set_eq:NN \l__numodel_saved_tblrenv_tl \g__numodel_tblrenv_tl
  \tl_if_empty:NF \l__numodel_cmd_tblrenv_tl
    { \tl_gset_eq:NN \g__numodel_tblrenv_tl \l__numodel_cmd_tblrenv_tl }
  \__numodel_with_prefix:Nn \l__numodel_eff_prefix_tl
    {
      \group_begin:
        \__numodel_apply_dsep:
        \__numodel_build_table:
        \typeout{TEXTMODEL~table:~\tl_to_str:N \g__numodel_table_tl}
        \tl_use:N \g__numodel_table_tl
      \group_end:
    }%
  \bool_gset_eq:NN \g__numodel_units_bool \l__numodel_saved_units_bool
  \tl_gset_eq:NN \g__numodel_tblrenv_tl \l__numodel_saved_tblrenv_tl
  \typeout{TEXTMODEL~done}
}
\bool_new:N \l__numodel_saved_units_bool
\tl_new:N \l__numodel_saved_tblrenv_tl

% ====================================================================
% \computemodel
% ====================================================================
% Run the model with the Euler method:
%   1. Record all variable values in Lua
%   2. Check stop condition (\mstop)
%   3. Execute every \mrule rule (in declaration order)
%   4. Repeat until stop or safety limit (default 20000 iterations)
```

```
%
% Requires: at least one \mstop and at least one \mrule.
% Performance: ~440 steps in ~4s, ~20000 steps in ~60s.

\cs_new_protected:Npn \__numodel_exec_rule:nn #1 #2
  {
    \cs_gset:cpe {#1} { \fp_eval:n {#2} }
  }

% Record all current variable values in Lua (O(1) per variable).
\cs_new_protected:Npn \__numodel_lua_record_all:
  {
    \seq_map_inline:Nn \g_mvar_names_seq
      {
        \directlua{ numodel.record(
          "\g_numodel_current_prefix_tl", "##1", \use:c{##1}) }
      }
    \directlua{ numodel.end_step("\g_numodel_current_prefix_tl") }
  }

% Set min/max TeX macros from Lua data after the simulation finishes.
\cs_new_protected:Npn \__numodel_set_minmax_from_lua:
  {
    \seq_map_inline:Nn \g_mvar_names_seq
      {
        \cs_gset:cpe { ##1 min }
          { \directlua{ numodel.get_min("\g_numodel_current_prefix_tl", "##1") } }
        \cs_gset:cpe { ##1 max }
          { \directlua{ numodel.get_max("\g_numodel_current_prefix_tl", "##1") } }
      }
  }


\NewDocumentCommand{\computemodel}{ O{} }{%
  \typeout{COMPUTE~start}
  \tl_clear:N \l__numodel_cmd_prefix_tl
  \keys_set:nn { numodel / cmd } {#1}
  \__numodel_resolve_eff_prefix:
  \__numodel_with_prefix:Nn \l__numodel_eff_prefix_tl
    { \__numodel_compute_body: }%
  \typeout{COMPUTE~done}
}

\cs_new_protected:Npn \__numodel_compute_body:
  {
    \tl_if_empty:NT \g_numodel_stop_expr_tl
      { \msg_error:nn { numodel } { no-stop } }
    \int_gzero:N \g_numodel_steps_int
    % Initialise Lua storage for this prefix
    \directlua{ numodel.init("\g_numodel_current_prefix_tl") }
    % Main loop
    \bool_gset_false:N \g_tmpa_bool
    \bool_do_until:Nn \g_tmpa_bool
      {
        \__numodel_lua_record_all:
        \int_gincr:N \g_numodel_steps_int
        \int_compare:nNnT
          { \fp_eval:n { \g_numodel_stop_expr_tl } } = { 1 }
          { \bool_gset_true:N \g_tmpa_bool }
        \bool_if:NF \g_tmpa_bool
          {
            \seq_map_inline:Nn \g_mrule_calc_seq
              { \__numodel_exec_rule:nn ##1 }
          }
        \int_compare:nNnT
          { \g_numodel_steps_int } > { \g_numodel_maxiter_int }
```

```
          {
            \bool_gset_true:N \g_tmpa_bool
            \msg_warning:nn { numodel } { maxiter }
          }
      }
    \__numodel_set_minmax_from_lua:
  }

% On-demand coordinates from Lua (after \computemodel).  Arguments
% are SHORT names; the current prefix is prepended automatically.
% See \mcoordsp for the form with an explicit prefix.
%
% Fully expandable (a plain \def around \directlua, no xparse
% wrapper).  Works directly inside \addplot coordinates
% {\mcoords{T}{V}} and pgfplots' math-expression parser, without
% pre-expansion via \edef.
\cs_new:Npn \mcoords #1#2
  {
    \directlua{ numodel.get_coords(
      "\g_numodel_current_prefix_tl",
      "\g_numodel_current_prefix_tl" .. "#1",
      "\g_numodel_current_prefix_tl" .. "#2") }
  }

% Variant with explicit prefix as first argument (instead of an
% optional argument, so the macro stays fully expandable).
\cs_new:Npn \mcoordsp #1#2#3
  {
    \directlua{ numodel.get_coords(
      "#1", "#1" .. "#2", "#1" .. "#3") }
  }

% Value of variable #1 at step #2 (0-based).  The variable name is
% SHORT (current prefix is prepended automatically).  See \mstepp for
% the form with an explicit prefix.
%
% Example -- tangent line at step 0:
%   \addplot[domain=0:\modTmax]
%     {\mstep{V}{0} + \mstep{A}{1} * (x - \mstep{T}{0})};
\cs_new:Npn \mstep #1#2
  {
    \directlua{ numodel.get_step(
      "\g_numodel_current_prefix_tl",
      "\g_numodel_current_prefix_tl" .. "#1", #2) }
  }

\cs_new:Npn \mstepp #1#2#3
  {
    \directlua{ numodel.get_step("#1", "#1" .. "#2", #3) }
  }

% ======================================================================
% \modelreset -- REMOVED
% ======================================================================
% \modelreset no longer exists.  Use \newmodelprefix{<prefix>} to set
% up a new model and \switchmodelprefix{<prefix>} to switch between
% existing ones.  Namespaces are additive -- variables from earlier
% models remain available.

% ======================================================================
% \graphicmodel -- Forrester diagram from \mvar/\mrule data
% ======================================================================
% Builds a tikzpicture with:
%   - stock, valve, aux and const nodes based on type and gridx/gridy
%   - flow arrows from valve to stock (via \flowarrow)
%   - causal arrows from the dependency graph
```

```
%
% Positioning: automatic (auto-layout) or manual via
%   \mvar[gridx=N, gridy=N]{...}.  Mixed mode is supported.
% Auto-layout places stocks+valves at gridy=0, aux at gridy=1, and
% constants at gridy=2.  Variables of type system are skipped.
% Auxiliary variables that act as inflow are placed as a valve
% (not as aux).

\tl_new:N   \g__numodel_graphic_tl
% Lua-populated cache (filled by numodel.tex_writeback at the start of
% each \graphicmodel call, consumed by the emit helpers below).
\prop_new:N \l__numodel_valve_for_prop      % aux/const -> stock (inflow)
% Secondary stocks for a shared inflow valve.  Value = comma-list of
% stock names; the renderer iterates it to draw a curved branch from
% the valve, sweeping over the primary stock into each extra target.
\prop_new:N \l__numodel_valve_extras_prop
\prop_new:N \l__numodel_outvalve_for_prop   % aux/const -> stock (outflow)
% Mirror of valve_extras_prop for shared outflow valves: a curved
% branch from each extra source stock arcs over the intermediate
% stocks into the shared valve's open end.
\prop_new:N \l__numodel_outvalve_extras_prop
\prop_new:N \l__numodel_between_valve_prop  % aux/const -> source_stock (between-flow)
\prop_new:N \l__numodel_between_target_prop % aux/const -> target_stock (between-flow)
\prop_new:N \l__numodel_stock_valve_prop    % stock -> source stock (stock-as-flow)
\prop_new:N \l__numodel_stock_phantom_valve_prop % stock -> source stock (phantom flow)
% For diagram-style=forrester|edu: separate valve gridx position next
% to the stock; the variable's own gridx/gridy keeps the natural
% position at gridy=1 (aux) or gridy=2 (constant).  vpos_y is always 0
% so it lives implicitly in the emit helpers.
\prop_new:N \l__numodel_vpos_x_prop         % varname -> valve gridx
\prop_new:N \l__numodel_vpos_y_prop         % varname -> valve gridy (used when gridmaxx wrap shifted
\tl_new:N   \l__numodel_flows_tl            % deferred flow arrows
\tl_new:N   \l__numodel_valves_tl           % deferred valve nodes (drawn on top)
\tl_new:N   \l__numodel_late_causals_tl     % causals pointing at valves

% --- Resolved render settings (set per-\graphicmodel call) ---
% These hold the effective values (after resolving diagram-style
% defaults and explicit global/local overrides) used by the emit
% helpers and by the public \flowarrow / \flowoutarrow macros.
\tl_new:N \l__numodel_flowarrow_eff_tl     % "hollow" or "filled"
\tl_new:N \l__numodel_valve_eff_tl         % "valve" | "circle" | "edu"
\tl_new:N \l__numodel_flowcloud_global_tl  % "true" or "false" (global default)

% Resolve flowarrow-style: explicit global key wins; otherwise
% diagram-style picks the default (forrester -> hollow,
% tight|edu -> filled).
\cs_new_protected:Npn \__numodel_resolve_flowarrow:
  {
    \tl_if_empty:NTF \g__numodel_flowarrow_style_tl
      {
        \str_if_eq:VnTF \g__numodel_diagram_style_tl { forrester }
          { \tl_set:Nn \l__numodel_flowarrow_eff_tl { hollow } }
          { \tl_set:Nn \l__numodel_flowarrow_eff_tl { filled } }
      }
      { \tl_set_eq:NN \l__numodel_flowarrow_eff_tl
                      \g__numodel_flowarrow_style_tl }
  }

% Resolve valve-style: explicit global key wins; otherwise
% diagram-style picks the default (forrester -> valve,
% tight|edu -> edu).
\cs_new_protected:Npn \__numodel_resolve_valve:
  {
    \tl_if_empty:NTF \g__numodel_valve_style_tl
      {
        \str_if_eq:VnTF \g__numodel_diagram_style_tl { forrester }
```

```
          { \tl_set:Nn \l__numodel_valve_eff_tl { valve } }
          { \tl_set:Nn \l__numodel_valve_eff_tl { edu   } }
      }
      { \tl_set_eq:NN \l__numodel_valve_eff_tl
                      \g__numodel_valve_style_tl }
  }

% Resolve the global flowarrow-cloud-tip default (no per-stock
% override yet; that's added on top in \__numodel_eff_flowcloud:n).
\cs_new_protected:Npn \__numodel_resolve_flowcloud:
  {
    \tl_if_empty:NTF \g__numodel_flowcloud_tl
      {
        \str_if_eq:VnTF \g__numodel_diagram_style_tl { forrester }
          { \tl_set:Nn \l__numodel_flowcloud_global_tl { true  } }
          { \tl_set:Nn \l__numodel_flowcloud_global_tl { false } }
      }
      { \tl_set_eq:NN \l__numodel_flowcloud_global_tl
                      \g__numodel_flowcloud_tl }
  }

% Per-stock flowcloud lookup.  #1 = stock varname.  Sets
% \l__numodel_flowcloud_eff_tl to "true" or "false".  Order: per-mvar
% override on the stock (via [flowarrow-cloud-tip=...]) wins over the
% global default.
\tl_new:N \l__numodel_flowcloud_eff_tl
\cs_new_protected:Npn \__numodel_eff_flowcloud:n #1
  {
    \tl_set:Ne \l__numodel_flowcloud_eff_tl { \use:c { #1 flowcloud } }
    \tl_if_empty:NT \l__numodel_flowcloud_eff_tl
      { \tl_set_eq:NN \l__numodel_flowcloud_eff_tl
                      \l__numodel_flowcloud_global_tl }
  }

% Sets the public \nmflowbody and \nmflowtip macros that the
% \flowarrow / \flowoutarrow / \flowbetweenarrow macros expand to.
% Reads the resolved flowarrow-style.
\cs_new_protected:Npn \__numodel_apply_flowarrow_style:
  {
    \str_if_eq:VnTF \l__numodel_flowarrow_eff_tl { hollow }
      {
        \cs_gset:Npn \nmflowbody { flowpipe-hollow }
        \cs_gset:Npn \nmflowtip  { flowpipe-hollow-tip }
        \cs_gset:Npn \nmflowtipcloudin  { flowpipe-hollow-cloudin }
        \cs_gset:Npn \nmflowtipcloudout { flowpipe-hollow-cloudout }
      }
      {
        \cs_gset:Npn \nmflowbody { flowpipe-filled }
        \cs_gset:Npn \nmflowtip  { flowpipe-filled-tip }
        \cs_gset:Npn \nmflowtipcloudin  { flowpipe-filled-cloudin }
        \cs_gset:Npn \nmflowtipcloudout { flowpipe-filled-cloudout }
      }
  }

% Sets the tikz node-style name for the valve and the boolean that
% controls whether the valve carries the variable's display label.
%   valve-style=valve  -> [valve-forrester], no label
%   valve-style=circle -> [valve-circle],   no label
%   valve-style=edu    -> [valve-edu],      with label
\tl_new:N   \l__numodel_valve_node_tl
\bool_new:N \l__numodel_valve_label_bool
\cs_new_protected:Npn \__numodel_apply_valve_style:
  {
    \str_case:VnF \l__numodel_valve_eff_tl
      {
        { valve  }
```

```
          {
            \tl_set:Nn \l__numodel_valve_node_tl { valve-forrester }
            \bool_set_false:N \l__numodel_valve_label_bool
          }
        { circle }
          {
            \tl_set:Nn \l__numodel_valve_node_tl { valve-circle }
            \bool_set_false:N \l__numodel_valve_label_bool
          }
        { edu }
          {
            \tl_set:Nn \l__numodel_valve_node_tl { valve-edu }
            \bool_set_true:N \l__numodel_valve_label_bool
          }
      }
      {
        \tl_set:Nn \l__numodel_valve_node_tl { valve-edu }
        \bool_set_true:N \l__numodel_valve_label_bool
      }
  }

% --- Diagram-style mode flag ---
% True at diagram-style=forrester|edu; consulted by \__numodel_place_node
% to decide between single-emit and natural+phantom-valve double-emit.
% Set in \__numodel_build_graphic from \g__numodel_diagram_style_tl.
\bool_new:N \l__numodel_keep_natural_bool


\cs_new_protected:Npn \__numodel_build_graphic:
  {
    \typeout{BUILD:~step~0~reset~auto-positions}
    % --- Reset gridx/gridy to the original user input ---
    % Auto-layout writes positions to var.gridx/gridy.  On a second
    % \graphicmodel call those would, without reset, be treated as
    % "manually placed".  The initial values are saved in
    % gridxinit/gridyinit by \mvar -- copy them back.
    \seq_map_inline:Nn \g_mvar_names_seq
      {
        \cs_gset:cpe { ##1 gridx } { \use:c { ##1 gridxinit } }
        \cs_gset:cpe { ##1 gridy } { \use:c { ##1 gridyinit } }
      }
    % Diagram-style mode: forrester|edu keep aux/const at their natural
    % gridy with a phantom valve next to the stock; tight collapses the
    % aux/const onto the valve position.  Consumed by \__numodel_place_node
    % to dispatch between single-emit and natural+phantom emit.
    \bool_set_false:N \l__numodel_keep_natural_bool
    \str_if_eq:VnT \g__numodel_diagram_style_tl { forrester }
      { \bool_set_true:N \l__numodel_keep_natural_bool }
    \str_if_eq:VnT \g__numodel_diagram_style_tl { edu }
      { \bool_set_true:N \l__numodel_keep_natural_bool }
    \typeout{BUILD:~step~1~lua~layout~writeback}
    % Pillar A - Lua is the single source of truth for flow detection
    % and auto-layout.  The writeback clears and fills \<var>gridx/gridy
    % and the flow-props that downstream emitters (place_node,
    % flow-builders, emit_natural_and_phantom, emit_stock_valve) read.
    \__numodel_lua_layout_writeback:
    \typeout{BUILD:~step~2~tikzpicture}
    % --- Build tikzpicture ---
    % Render order matters: each segment overlays the previous one,
    % so we draw flows first (one continuous arrow per flow), then
    % the valve nodes on top (with white fill so they cover the
    % section of the arrow that lies underneath), and finally the
    % causal arrows that point at those valves.
    \tl_gclear:N \g__numodel_graphic_tl
    \tl_clear:N  \l__numodel_flows_tl
    \tl_clear:N  \l__numodel_valves_tl
```

```
    \tl_clear:N  \l__numodel_late_causals_tl
    \tl_gput_right:Nn \g__numodel_graphic_tl
      { \begin{tikzpicture}[gridscale] }
    % Nodes (stocks, aux, const, clouds; flows -> flows_tl,
    % valves -> valves_tl)
    \seq_map_inline:Nn \g_mvar_names_seq
      { \typeout{NODE:~##1} \__numodel_place_node:n {##1} }
    \typeout{BUILD:~step~5~stock-valve~nodes}
    % Stock-as-flow valve nodes
    \prop_map_inline:Nn \l__numodel_stock_valve_prop
      {
        % ##1 = stock (e.g. modH), ##2 = source stock (e.g. modV)
        % Skip entries that are not real stock-valves (e.g. __svgx keys)
        \tl_if_in:nnF {##1} { __sv }
          { \__numodel_emit_stock_valve:nn {##1} {##2} }
      }
    % Stock-as-rate phantom-valve nodes (source stock has no matching
    % outflow term, so the inflow gets a cloud at the open end)
    \prop_map_inline:Nn \l__numodel_stock_phantom_valve_prop
      {
        \tl_if_in:nnF {##1} { __sv }
          { \__numodel_emit_stock_phantom_valve:nn {##1} {##2} }
      }
    \typeout{BUILD:~step~6~flow~arrows}
    % Flow arrows (one segment each; drawn underneath the valves)
    \tl_gput_right:NV \g__numodel_graphic_tl \l__numodel_flows_tl
    \typeout{BUILD:~step~6a~valve~nodes}
    % Valve nodes (white-filled, drawn ON TOP of the flow arrows)
    \tl_gput_right:NV \g__numodel_graphic_tl \l__numodel_valves_tl
    \typeout{BUILD:~step~6b+7~causal~arrows~(Lua)}
    % A2: causal arrows via Lua - emit_causals demultiplexes on
    % tgt_is_valve and pushes to \g__numodel_graphic_tl or
    % \l__numodel_late_causals_tl respectively.  Afterwards append
    % late-causals to graphic_tl once: that covers both the pushes
    % from the flow-builders (step 6/6a) and those from emit_causals.
    \__numodel_lua_emit_causals:
    \tl_gput_right:NV \g__numodel_graphic_tl \l__numodel_late_causals_tl
    \typeout{BUILD:~step~8~done}
    \tl_gput_right:Nn \g__numodel_graphic_tl
      { \end{tikzpicture} }
  }

% --- Node placement ---
\cs_new_protected:Npn \__numodel_place_node:n #1
  {
    \bool_set_true:N \l_tmpa_bool
    \tl_set:Ne \l__numodel_scratch_tl { \use:c { #1 type } }
    \exp_args:NV \str_if_eq:nnT \l__numodel_scratch_tl { system }
      { \bool_set_false:N \l_tmpa_bool }
    \bool_if:NT \l_tmpa_bool
      {
        \int_compare:nNnT { \use:c { #1 gridx } } = { -1 }
          { \bool_set_false:N \l_tmpa_bool }
      }
    \bool_if:NT \l_tmpa_bool
      {
        \tl_set:Ne \l__numodel_tmp_tl { \use:c { #1 text } }
        % With forrester|edu, valve vars get a dual emission:
        % the natural aux/const node + a phantom valve next to the stock.
        \bool_set_false:N \l_tmpb_bool   % is this a valve var?
        \prop_if_in:NnT \l__numodel_valve_for_prop {#1}
          { \bool_set_true:N \l_tmpb_bool }
        \prop_if_in:NnT \l__numodel_outvalve_for_prop {#1}
          { \bool_set_true:N \l_tmpb_bool }
        \prop_if_in:NnT \l__numodel_between_valve_prop {#1}
          { \bool_set_true:N \l_tmpb_bool }
```

```
        \bool_lazy_and:nnTF
          { \l__numodel_keep_natural_bool } { \l_tmpb_bool }
          { \__numodel_emit_natural_and_phantom:n {#1} }
          {
            \prop_if_in:NnTF \l__numodel_valve_for_prop {#1}
              { \__numodel_emit_valve:n {#1} }
              {
                \prop_if_in:NnTF \l__numodel_outvalve_for_prop {#1}
                  { \__numodel_emit_outvalve:n {#1} }
                  {
                    \prop_if_in:NnTF \l__numodel_between_valve_prop {#1}
                      { \__numodel_emit_between_valve:n {#1} }
                      {
                        \tl_set:Ne \l__numodel_scratch_tl { \use:c { #1 type } }
                        \exp_args:NV \str_if_eq:nnT \l__numodel_scratch_tl { stock }
                          { \__numodel_emit_stock:n {#1} }
                        \exp_args:NV \str_if_eq:nnT \l__numodel_scratch_tl { aux }
                          { \__numodel_emit_aux:n {#1} }
                        \exp_args:NV \str_if_eq:nnT \l__numodel_scratch_tl { constant }
                          { \__numodel_emit_const:n {#1} }
                      }
                  }
              }
          }
      }
  }

% --- Helper: emit a coordinate node at the valve position ---
% The flow arrow starts/ends at this coordinate (or at an offset
% relative to it, for the open-end without cloud).  The visible
% white-filled valve is added later (valves_tl) at the same
% coordinates so it overlays the arrow.
% #1 = coord id   #2 = x   #3 = y
\cs_new_protected:Npn \__numodel_emit_valve_coord:nnn #1 #2 #3
  {
    \tl_gput_right:Ne \g__numodel_graphic_tl
      {
        \exp_not:N \coordinate ~
        (#1) ~ \exp_not:n{at} ~ (#2 , ~ #3) ;
      }
  }

% --- Helper: append a valve node to the deferred valves_tl ---
% Renders \node[<valve-style>] (id) at (x,y) {<label>}; later in the
% picture so the white fill overlays the flow arrow underneath.
% #1 = node id   #2 = x coord   #3 = y coord   #4 = label tl name
\cs_new_protected:Npn \__numodel_defer_valve:nnnn #1 #2 #3 #4
  {
    \tl_put_right:Ne \l__numodel_valves_tl
      {
        \exp_not:N \node ~ [ \tl_use:N \l__numodel_valve_node_tl ] ~
        (#1) ~ \exp_not:n{at} ~ (#2 , ~ #3) ~
        {
          \bool_if:NTF \l__numodel_valve_label_bool
            { \exp_not:N $ \exp_not:V #4 \exp_not:N $ }
            { }
        } ;
      }
  }

% --- Forrester/edu-mode emission: natural node + phantom valve ---
% The variable sits as an ordinary aux/const node at its own gridy.
% The phantom valve (id #1__v) sits next to the stock at gridy=0; its
% appearance follows the resolved valve-style (label included only
% when valve-style=edu).  A causal arrow from #1 -> #1__v is drawn
% (deferred to late_causals_tl so it lands on top of the white-
```

```
% filled valve).
\tl_new:N   \l__numodel_phantom_label_tl
\cs_new_protected:Npn \__numodel_emit_natural_and_phantom:n #1
  {
    % --- 1. Natural node (aux or constant) ---
    \tl_set:Ne \l__numodel_scratch_tl { \use:c { #1 type } }
    \exp_args:NV \str_if_eq:nnT \l__numodel_scratch_tl { aux }
      { \__numodel_emit_aux:n {#1} }
    \exp_args:NV \str_if_eq:nnT \l__numodel_scratch_tl { constant }
      { \__numodel_emit_const:n {#1} }
    % --- 2. Locate the phantom valve x/y position ---
    \prop_get:NnNT \l__numodel_vpos_x_prop {#1} \l__numodel_phantom_x_tl
      {
        \prop_get:NnNF \l__numodel_vpos_y_prop {#1} \l__numodel_scratch_y_tl
          { \tl_set:Nn \l__numodel_scratch_y_tl { 0 } }
        \tl_set:Ne \l__numodel_phantom_label_tl { \use:c { #1 text } }
        % --- 3. Coordinate at the phantom valve position ---
        \__numodel_emit_valve_coord:nnn { #1 __vc }
          { \tl_use:N \l__numodel_phantom_x_tl }
          { \tl_use:N \l__numodel_scratch_y_tl }
        % --- 4. Flow arrow + (optional) cloud, one segment ---
        \prop_if_in:NnTF \l__numodel_valve_for_prop {#1}
          {
            \tl_set:Ne \l__numodel_tmp_tl
              { \prop_item:Nn \l__numodel_valve_for_prop {#1} }
            \__numodel_eff_flowcloud:V \l__numodel_tmp_tl
            \str_if_eq:VnTF \l__numodel_flowcloud_eff_tl { true }
              {
                \tl_put_right:Ne \l__numodel_flows_tl
                  {
                    \exp_not:N \flowarrow [#1 __cl] {#1 __vc}
                      { \tl_use:N \l__numodel_tmp_tl }
                  }
              }
              {
                \tl_put_right:Ne \l__numodel_flows_tl
                  {
                    \exp_not:N \flowarrow {#1 __vc}
                      { \tl_use:N \l__numodel_tmp_tl }
                  }
              }
            \__numodel_emit_valve_extras:n {#1}
          }
          {
            \prop_if_in:NnTF \l__numodel_outvalve_for_prop {#1}
              {
                \tl_set:Ne \l__numodel_tmp_tl
                  { \prop_item:Nn \l__numodel_outvalve_for_prop {#1} }
                \__numodel_eff_flowcloud:V \l__numodel_tmp_tl
                \str_if_eq:VnTF \l__numodel_flowcloud_eff_tl { true }
                  {
                    \tl_put_right:Ne \l__numodel_flows_tl
                      {
                        \exp_not:N \flowoutarrow [#1 __cl]
                          { \tl_use:N \l__numodel_tmp_tl } {#1 __vc}
                      }
                  }
                  {
                    \tl_put_right:Ne \l__numodel_flows_tl
                      {
                        \exp_not:N \flowoutarrow
                          { \tl_use:N \l__numodel_tmp_tl } {#1 __vc}
                      }
                  }
                \__numodel_emit_outvalve_extras:n {#1}
              }
```

```
              {
                % between (no cloud: both ends are stocks)
                \tl_set:Ne \l__numodel_tmp_tl
                  { \prop_item:Nn \l__numodel_between_valve_prop {#1} }
                \tl_set:Ne \l__numodel_scratch_tl
                  { \prop_item:Nn \l__numodel_between_target_prop {#1} }
                \tl_put_right:Ne \l__numodel_flows_tl
                  {
                    \exp_not:N \flowbetweenarrow
                      { \tl_use:N \l__numodel_tmp_tl }
                      {#1 __vc}
                      { \tl_use:N \l__numodel_scratch_tl }
                  }
              }
          }
        % --- 5. Defer the phantom valve drawing (overlays flow) ---
        \__numodel_defer_valve:nnnn { #1 __v }
          { \tl_use:N \l__numodel_phantom_x_tl }
          { \tl_use:N \l__numodel_scratch_y_tl }
          \l__numodel_phantom_label_tl
        % --- 6. Causal arrow natural -> phantom valve (deferred) ---
        \tl_put_right:Ne \l__numodel_late_causals_tl
          {
            \exp_not:N \draw ~ [\exp_not:n{causal}] ~
            (#1) ~ \exp_not:n{to} ~ (#1 __v) ;
          }
      }
  }
\tl_new:N \l__numodel_phantom_x_tl
\cs_generate_variant:Nn \__numodel_eff_flowcloud:n { V }

\cs_new_protected:Npn \__numodel_emit_stock:n #1
  {
    \tl_gput_right:Ne \g__numodel_graphic_tl
      {
        \exp_not:N \node ~ [\exp_not:n{stock}] ~
        (#1) ~ \exp_not:n{at} ~
        ( \use:c{#1 gridx} , ~ \use:c{#1 gridy} ) ~
        { \exp_not:N $ \exp_not:V \l__numodel_tmp_tl \exp_not:N $ } ;
      }
  }

% Tight-mode helpers: the valve sits at the variable's own gridx/
% gridy.  We emit a coordinate node (#1__vc) early so the flow
% arrow has a positioning anchor, and defer the visible valve node
% (#1) to valves_tl so it overlays the arrow with white fill.
\cs_new_protected:Npn \__numodel_emit_tight_valve_setup:n #1
  {
    \__numodel_emit_valve_coord:nnn { #1 __vc }
      { \use:c{#1 gridx} } { \use:c{#1 gridy} }
    \__numodel_defer_valve:nnnn {#1}
      { \use:c{#1 gridx} } { \use:c{#1 gridy} }
      \l__numodel_tmp_tl
  }

\cs_new_protected:Npn \__numodel_emit_valve:n #1
  {
    \__numodel_emit_tight_valve_setup:n {#1}
    \tl_set:Ne \l__numodel_scratch_tl
      { \prop_item:Nn \l__numodel_valve_for_prop {#1} }
    \__numodel_eff_flowcloud:V \l__numodel_scratch_tl
    \str_if_eq:VnTF \l__numodel_flowcloud_eff_tl { true }
      {
        \tl_put_right:Ne \l__numodel_flows_tl
          {
            \exp_not:N \flowarrow [#1 __cl] {#1 __vc}
```

```
              { \tl_use:N \l__numodel_scratch_tl }
          }
      }
      {
        \tl_put_right:Ne \l__numodel_flows_tl
          {
            \exp_not:N \flowarrow {#1 __vc}
              { \tl_use:N \l__numodel_scratch_tl }
          }
      }
    \__numodel_emit_valve_extras:n {#1}
  }

% Emit one curved inflow branch per shared-inflow extra target
% (\l__numodel_valve_extras_prop[fv]).  Each branch starts at the
% valve coordinate and sweeps over the primary stock to the extra
% target's north edge.  No-op when the valve is not shared.
\cs_new_protected:Npn \__numodel_emit_valve_extras:n #1
  {
    \prop_get:NnNT \l__numodel_valve_extras_prop {#1} \l__numodel_scratch_tl
      {
        \clist_set:NV \l__numodel_scratch_clist \l__numodel_scratch_tl
        \clist_map_inline:Nn \l__numodel_scratch_clist
          {
            \tl_put_right:Nn \l__numodel_flows_tl
              { \flowarrowbent {#1 __vc} {##1} }
          }
      }
  }

% Mirror of \__numodel_emit_valve_extras:n for shared outflow valves.
% Each extra source stock gets a curved branch arcing over the
% intermediate stocks into the shared valve's open end.
\cs_new_protected:Npn \__numodel_emit_outvalve_extras:n #1
  {
    \prop_get:NnNT \l__numodel_outvalve_extras_prop
      {#1} \l__numodel_scratch_tl
      {
        \clist_set:NV \l__numodel_scratch_clist \l__numodel_scratch_tl
        \clist_map_inline:Nn \l__numodel_scratch_clist
          {
            \tl_put_right:Nn \l__numodel_flows_tl
              { \flowoutarrowbent {##1} {#1 __vc} }
          }
      }
  }
\clist_new:N \l__numodel_scratch_clist

\cs_new_protected:Npn \__numodel_emit_outvalve:n #1
  {
    \__numodel_emit_tight_valve_setup:n {#1}
    \tl_set:Ne \l__numodel_scratch_tl
      { \prop_item:Nn \l__numodel_outvalve_for_prop {#1} }
    \__numodel_eff_flowcloud:V \l__numodel_scratch_tl
    \str_if_eq:VnTF \l__numodel_flowcloud_eff_tl { true }
      {
        \tl_put_right:Ne \l__numodel_flows_tl
          {
            \exp_not:N \flowoutarrow [#1 __cl]
              { \tl_use:N \l__numodel_scratch_tl } {#1 __vc}
          }
      }
      {
        \tl_put_right:Ne \l__numodel_flows_tl
          {
            \exp_not:N \flowoutarrow
```

```
              { \tl_use:N \l__numodel_scratch_tl } {#1 __vc}
          }
      }
    \__numodel_emit_outvalve_extras:n {#1}
  }

\cs_new_protected:Npn \__numodel_emit_between_valve:n #1
  {
    \__numodel_emit_tight_valve_setup:n {#1}
    \tl_set:Ne \l__numodel_scratch_tl
      { \prop_item:Nn \l__numodel_between_valve_prop {#1} }
    \tl_set:Ne \l__numodel_tmp_tl
      { \prop_item:Nn \l__numodel_between_target_prop {#1} }
    \tl_put_right:Ne \l__numodel_flows_tl
      {
        \exp_not:N \flowbetweenarrow
          { \tl_use:N \l__numodel_scratch_tl }
          {#1 __vc}
          { \tl_use:N \l__numodel_tmp_tl }
      }
  }

\cs_new_protected:Npn \__numodel_emit_aux:n #1
  {
    \tl_gput_right:Ne \g__numodel_graphic_tl
      {
        \exp_not:N \node ~ [\exp_not:n{aux}] ~
        (#1) ~ \exp_not:n{at} ~
        ( \use:c{#1 gridx} , ~ \use:c{#1 gridy} ) ~
        { \exp_not:N $ \exp_not:V \l__numodel_tmp_tl \exp_not:N $ } ;
      }
  }

\cs_new_protected:Npn \__numodel_emit_const:n #1
  {
    \tl_gput_right:Ne \g__numodel_graphic_tl
      {
        \exp_not:N \constnode
        {#1}
        { \use:c{#1 gridx} , ~ \use:c{#1 gridy} }
        { \exp_not:N $ \exp_not:V \l__numodel_tmp_tl \exp_not:N $ }
      }
  }

% --- Stock-as-flow valve emission ---
\cs_new_protected:Npn \__numodel_emit_stock_valve:nn #1 #2
  {
    % #1 = target stock (e.g. modH), #2 = source stock (e.g. modV)
    % Pull gridx from the saved property
    \prop_get:NnNTF \l__numodel_stock_valve_prop { #1 __svgx }
      \l__numodel_scratch_tl
      {
        % Look up the wrap-shifted y-slot; default to 0 (the row that
        % the auto layout used before any gridmaxx wrap was applied).
        \prop_get:NnNF \l__numodel_stock_valve_prop { #1 __svgy }
          \l__numodel_scratch_y_tl
          { \tl_set:Nn \l__numodel_scratch_y_tl { 0 } }
        % Coordinate at the valve position (used by the flow arrow)
        \__numodel_emit_valve_coord:nnn { #1 __svc }
          { \tl_use:N \l__numodel_scratch_tl }
          { \tl_use:N \l__numodel_scratch_y_tl }
        % Flow arrow from source-stock through valve into target-stock
        % (no cloud: both ends are stocks).  The arrow runs in one
        % continuous segment; the visible valve will be drawn ON TOP
        % later via valves_tl.
        \tl_put_right:Ne \l__numodel_flows_tl
```

```
          {
            \exp_not:N \flowbetweenarrow {#2} {#1 __svc} {#1}
          }
        % Defer the visible valve node (white fill overlays arrow)
        \tl_set:Ne \l__numodel_tmp_tl { \use:c { #2 text } }
        \__numodel_defer_valve:nnnn { #1 __sv }
          { \tl_use:N \l__numodel_scratch_tl }
          { \tl_use:N \l__numodel_scratch_y_tl }
          \l__numodel_tmp_tl
        % Causal arrow source-stock -> valve.  Bend it (curved) only
        % when the source stock and the valve sit on the same row -
        % then the straight causal would coincide with the flow pipe
        % and disappear behind it.  When they are on different rows
        % (e.g. after a gridmaxx wrap) the causal is vertical-ish and
        % stays visible without a bend.  Deferred so it lands on top
        % of the white-filled valve.
        \str_if_eq:eeTF { \tl_use:N \l__numodel_scratch_y_tl }
                        { \use:c { #2 gridy } }
          {
            \tl_put_right:Ne \l__numodel_late_causals_tl
              {
                \exp_not:N \draw ~ [\exp_not:n{causal}] ~
                (#2) ~ \exp_not:n{to} ~ [\exp_not:n{bend~left=30}] ~ (#1__sv) ;
              }
          }
          {
            \tl_put_right:Ne \l__numodel_late_causals_tl
              {
                \exp_not:N \draw ~ [\exp_not:n{causal}] ~
                (#2) ~ \exp_not:n{to} ~ (#1__sv) ;
              }
          }
      }
      { }  % no gridx found: skip
  }

% --- Stock phantom-valve emission (source stock as rate factor) ---
% Source stock acts as a rate factor for the target stock without a
% matching outflow term; render with a cloud-fed inflow valve next to
% the target stock and link the source stock via a causal arrow.
\cs_new_protected:Npn \__numodel_emit_stock_phantom_valve:nn #1 #2
  {
    % #1 = target stock (e.g. ballY), #2 = source stock (e.g. ballV)
    \prop_get:NnNTF \l__numodel_stock_phantom_valve_prop { #1 __svgx }
      \l__numodel_scratch_tl
      {
        % Look up the wrap-shifted y-slot; default to 0.
        \prop_get:NnNF \l__numodel_stock_phantom_valve_prop { #1 __svgy }
          \l__numodel_scratch_y_tl
          { \tl_set:Nn \l__numodel_scratch_y_tl { 0 } }
        % Coordinate at the valve position
        \__numodel_emit_valve_coord:nnn { #1 __svc }
          { \tl_use:N \l__numodel_scratch_tl }
          { \tl_use:N \l__numodel_scratch_y_tl }
        % Flow arrow: cloud (open end) -> valve -> target stock
        \__numodel_eff_flowcloud:n {#1}
        \str_if_eq:VnTF \l__numodel_flowcloud_eff_tl { true }
          {
            \tl_put_right:Ne \l__numodel_flows_tl
              {
                \exp_not:N \flowarrow [#1 __svcl] {#1 __svc} {#1}
              }
          }
          {
            \tl_put_right:Ne \l__numodel_flows_tl
              {
```

```
              \exp_not:N \flowarrow {#1 __svc} {#1}
            }
        }
      % Defer the visible valve node, labelled with the source stock
      \tl_set:Ne \l__numodel_tmp_tl { \use:c { #2 text } }
      \__numodel_defer_valve:nnnn { #1 __sv }
        { \tl_use:N \l__numodel_scratch_tl }
        { \tl_use:N \l__numodel_scratch_y_tl }
        \l__numodel_tmp_tl
      % Causal arrow source-stock -> valve.  Bend (curved) only when
      % source and valve sit on the same row - see the equivalent
      % comment in \__numodel_emit_stock_valve:nn for the rationale.
      \str_if_eq:eeTF { \tl_use:N \l__numodel_scratch_y_tl }
                      { \use:c { #2 gridy } }
        {
          \tl_put_right:Ne \l__numodel_late_causals_tl
            {
              \exp_not:N \draw ~ [\exp_not:n{causal}] ~
              (#2) ~ \exp_not:n{to} ~ [\exp_not:n{bend~left=30}] ~ (#1__sv) ;
            }
        }
        {
          \tl_put_right:Ne \l__numodel_late_causals_tl
            {
              \exp_not:N \draw ~ [\exp_not:n{causal}] ~
              (#2) ~ \exp_not:n{to} ~ (#1__sv) ;
            }
        }
    }
    { }  % no gridx found: skip
}

% ---------------------------------------------------------------------
% Pillar A - Lua-side layout writeback.  A single directlua call
% that runs compute_layout (build_deps, classify_flows,
% populate_occupied, auto_layout, causals) and writes the resulting
% state back to the TeX-side props (\<var>gridx/gridy and the
% flow-props) that the downstream emitters (place_node,
% flow-builders, emit_natural_and_phantom, emit_stock_valve) read.
% Lua is the single source of truth for the decision logic; TeX
% only renders.
% ---------------------------------------------------------------------
\cs_new_protected:Npn \__numodel_lua_layout_writeback:
{
  % 1. Clear the props that Lua refills, so repeated \graphicmodel
  %    calls don't produce duplicate entries.
  \prop_clear:N  \l__numodel_valve_for_prop
  \prop_clear:N  \l__numodel_valve_extras_prop
  \prop_clear:N  \l__numodel_outvalve_for_prop
  \prop_clear:N  \l__numodel_outvalve_extras_prop
  \prop_clear:N  \l__numodel_between_valve_prop
  \prop_clear:N  \l__numodel_between_target_prop
  \prop_clear:N  \l__numodel_stock_valve_prop
  \prop_clear:N  \l__numodel_stock_phantom_valve_prop
  \prop_clear:N  \l__numodel_vpos_x_prop
  \prop_clear:N  \l__numodel_vpos_y_prop
  % 2. Run the Lua pipeline.
  \directlua{ numodel.compute_layout(
    "\g_numodel_current_prefix_tl",
    "\g__numodel_diagram_style_tl",
    \int_use:N \g_numodel_gridmaxx_int) }
  % 3. Write gridx/gridy and all flow-props back.
  %    tex_writeback emits a list of expl3 statements via tex.print.
  \directlua{ numodel.tex_writeback(
    "\g_numodel_current_prefix_tl") }
}
```

```
\cs_new_protected:Npn \__numodel_lua_emit_causals:
  {
    \directlua{ numodel.emit_causals(
      "\g_numodel_current_prefix_tl") }
  }

\NewDocumentCommand{\graphicmodel}{ O{} }{%
  \tl_clear:N \l__numodel_cmd_prefix_tl
  \tl_clear:N \l__numodel_cmd_diagstyle_tl
  \tl_clear:N \l__numodel_cmd_flowarrow_tl
  \tl_clear:N \l__numodel_cmd_valve_tl
  \tl_clear:N \l__numodel_cmd_flowcloud_tl
  \keys_set:nn { numodel / cmd } {#1}
  \__numodel_resolve_eff_prefix:
  % Temporary overrides: save the global state, apply the per-call
  % key values before build, restore afterwards.  This way
  % \graphicmodel[diagram-style=forrester] flips one render without
  % touching the global \numodelsetup state.
  \tl_set_eq:NN \l__numodel_saved_diagstyle_tl \g__numodel_diagram_style_tl
  \tl_set_eq:NN \l__numodel_saved_flowarrow_tl \g__numodel_flowarrow_style_tl
  \tl_set_eq:NN \l__numodel_saved_valve_tl     \g__numodel_valve_style_tl
  \tl_set_eq:NN \l__numodel_saved_flowcloud_tl \g__numodel_flowcloud_tl
  \tl_if_empty:NF \l__numodel_cmd_diagstyle_tl
    { \tl_gset_eq:NN \g__numodel_diagram_style_tl \l__numodel_cmd_diagstyle_tl }
  \tl_if_empty:NF \l__numodel_cmd_flowarrow_tl
    { \tl_gset_eq:NN \g__numodel_flowarrow_style_tl \l__numodel_cmd_flowarrow_tl }
  \tl_if_empty:NF \l__numodel_cmd_valve_tl
    { \tl_gset_eq:NN \g__numodel_valve_style_tl \l__numodel_cmd_valve_tl }
  \tl_if_empty:NF \l__numodel_cmd_flowcloud_tl
    { \tl_gset_eq:NN \g__numodel_flowcloud_tl \l__numodel_cmd_flowcloud_tl }
  % Resolve the effective values for this render and propagate them
  % to \nmflowbody / \nmflowtip used by the flow macros.
  \__numodel_resolve_flowarrow:
  \__numodel_resolve_valve:
  \__numodel_resolve_flowcloud:
  \__numodel_apply_flowarrow_style:
  \__numodel_apply_valve_style:
  \__numodel_with_prefix:Nn \l__numodel_eff_prefix_tl
    {
      \__numodel_build_graphic:
      \tl_use:N \g__numodel_graphic_tl
    }%
  \tl_gset_eq:NN \g__numodel_diagram_style_tl    \l__numodel_saved_diagstyle_tl
  \tl_gset_eq:NN \g__numodel_flowarrow_style_tl  \l__numodel_saved_flowarrow_tl
  \tl_gset_eq:NN \g__numodel_valve_style_tl      \l__numodel_saved_valve_tl
  \tl_gset_eq:NN \g__numodel_flowcloud_tl        \l__numodel_saved_flowcloud_tl
}
\tl_new:N \l__numodel_saved_diagstyle_tl
\tl_new:N \l__numodel_saved_flowarrow_tl
\tl_new:N \l__numodel_saved_valve_tl
\tl_new:N \l__numodel_saved_flowcloud_tl

% ====================================================================
% Plot machinery
% ====================================================================
% \diagrammodel calls the pgfplots/calcplotdims infrastructure
% through internal wrappers \__numodel_calc_plot_dims: and
% \__numodel_draw_plot:n.  Those wrappers forward to \calcplotdims
% and \drawplot from the sibling package numodel-plot.
%
% The wrappers are not \cs_new_eq:NN because numodel-plot can load
% after this package; at load time \calcplotdims and \drawplot may
% not yet exist.  With a wrapper they are looked up at expansion
% time.

```

```
\cs_new:Npn  \__numodel_calc_plot_dims: { \calcplotdims }
\cs_new:Npn  \__numodel_draw_plot:n   #1 { \drawplot {#1} }

% ======================================================================
% \diagrammodel -- convenience command for model graphs
% ======================================================================
% Usage: \diagrammodel{xvar}{yvar}[extra drawplot code]{label}
%        \diagrammodel{xvar}{yvar1,yvar2,...,yvarn}[extra]{label}
%
% Builds a complete figure with model points.  Sets axis dimensions
% from min/max, axis labels from text/unit, and emits caption and
% label automatically.
%
% When the second argument is a comma-separated list, every y-variable
% whose unit matches the first one is drawn into the same diagram with
% the y-axis scaled to the joint min/max range.  Variables with a
% mismatching unit are dropped with a warning.  Every series is drawn
% as discrete model points (no connecting lines); the seven colours
% come from Okabe & Ito's colour-blind safe palette (yellow omitted)
% and are ordered so consecutive series differ in luminance, which
% keeps them distinguishable on a greyscale printout.  The legend
% lists each kept variable's display text.

\tl_new:N \l__numodel_xname_tl
\tl_new:N \l__numodel_yname_tl
\tl_new:N \l__numodel_yfull_tl
\tl_new:N \l__numodel_yunit_first_tl
\tl_new:N \l__numodel_yqty_tl
\tl_new:N \l__numodel_plotbody_tl
\clist_new:N \l__numodel_yshort_clist
\int_new:N \l__numodel_yidx_int

% Colour-blind safe palette (Okabe & Ito; yellow #F0E442 omitted for
% poor white-background contrast).  Ordered so consecutive series
% differ in luminance as well as hue, keeping them distinguishable
% on a greyscale printout.
\definecolor{numodelseriesa}{HTML}{000000} % black            (L ~ 0.00)
\definecolor{numodelseriesb}{HTML}{E69F00} % orange           (L ~ 0.69)
\definecolor{numodelseriesc}{HTML}{0072B2} % dark blue       (L ~ 0.36)
\definecolor{numodelseriesd}{HTML}{56B4E9} % sky blue        (L ~ 0.74)
\definecolor{numodelseriese}{HTML}{D55E00} % vermillion      (L ~ 0.50)
\definecolor{numodelseriesf}{HTML}{CC79A7} % reddish purple  (L ~ 0.62)
\definecolor{numodelseriesg}{HTML}{009E73} % bluish green    (L ~ 0.52)

\clist_new:N \g__numodel_series_colors_clist
\clist_gset:Nn \g__numodel_series_colors_clist
  {
    numodelseriesa , numodelseriesb , numodelseriesc , numodelseriesd ,
    numodelseriese , numodelseriesf , numodelseriesg
  }

% Return the cycling colour name for series #1 (1-based), modulo 7.
\cs_new:Npn \__numodel_series_color:n #1
  {
    \clist_item:Nn \g__numodel_series_colors_clist
      { 1 + \int_mod:nn { #1 - 1 } { 7 } }
  }

\NewDocumentCommand{\diagrammodel}{ O{} m m O{} m }
  {
    \tl_clear:N \l__numodel_cmd_prefix_tl
    \keys_set:nn { numodel / cmd } {#1}
    \__numodel_resolve_eff_prefix:
    \tl_set:Ne \l__numodel_xname_tl { \l__numodel_eff_prefix_tl #2 }
    % --- Parse comma-separated y-variable list ----------------------
    % Keep only variables whose unitraw matches the first one's; warn
```

```
  % about the rest.
  \clist_clear:N \l__numodel_yshort_clist
  \tl_clear:N    \l__numodel_yunit_first_tl
  \int_zero:N    \l__numodel_yidx_int
  \clist_map_inline:nn {#3}
    {
      \int_incr:N \l__numodel_yidx_int
      \tl_set:Ne \l__numodel_yfull_tl { \l__numodel_eff_prefix_tl ##1 }
      \int_compare:nNnTF { \l__numodel_yidx_int } = { 1 }
        {
          \tl_set:Ne \l__numodel_yunit_first_tl
            { \use:c { \l__numodel_yfull_tl unitraw } }
          \clist_put_right:Nn \l__numodel_yshort_clist {##1}
        }
        {
          \tl_set:Ne \l_tmpa_tl
            { \use:c { \l__numodel_yfull_tl unitraw } }
          \str_if_eq:VVTF \l_tmpa_tl \l__numodel_yunit_first_tl
            { \clist_put_right:Nn \l__numodel_yshort_clist {##1} }
            {
              \msg_warning:nnee { numodel } { unit-mismatch }
                { \tl_use:N \l__numodel_yfull_tl }
                { \tl_use:N \l__numodel_yunit_first_tl }
            }
        }
    }
  % Canonical y-name = first kept variable (legacy single-series
  % accessor used by the caption when only one series is plotted).
  \clist_get:NN \l__numodel_yshort_clist \l_tmpa_tl
  \tl_set:Ne \l__numodel_yname_tl
    { \l__numodel_eff_prefix_tl \l_tmpa_tl }
  % --- Axis ranges ------------------------------------------------
  \def\xmin{\use:c{\l__numodel_xname_tl min}}
  \def\xmax{\use:c{\l__numodel_xname_tl max}}
  \def\xlabelqty{\use:c{\l__numodel_xname_tl text}}
  \def\xlabelunit{\use:c{\l__numodel_xname_tl unitraw}}
  % Reduce over the kept y-variables.  Use \edef + \fpeval so the
  % per-variable min/max accessors are expanded to numeric literals
  % before the fp parser sees them.
  \def\ymin{inf}
  \def\ymax{-inf}
  \clist_map_inline:Nn \l__numodel_yshort_clist
    {
      \tl_set:Ne \l__numodel_yfull_tl { \l__numodel_eff_prefix_tl ##1 }
      \edef\ymin{\fpeval{min(\ymin,\use:c{\l__numodel_yfull_tl min})}}
      \edef\ymax{\fpeval{max(\ymax,\use:c{\l__numodel_yfull_tl max})}}
    }
  % --- y-axis label: comma-joined display texts, common unit -----
  \tl_clear:N \l__numodel_yqty_tl
  \int_zero:N \l__numodel_yidx_int
  \clist_map_inline:Nn \l__numodel_yshort_clist
    {
      \int_incr:N \l__numodel_yidx_int
      \int_compare:nNnF { \l__numodel_yidx_int } = { 1 }
        { \tl_put_right:Nn \l__numodel_yqty_tl { ,~ } }
      \tl_put_right:Ne \l__numodel_yqty_tl
        { \use:c { \l__numodel_eff_prefix_tl ##1 text } }
    }
  \edef\ylabelqty{\tl_use:N \l__numodel_yqty_tl}
  \edef\ylabelunit{\tl_use:N \l__numodel_yunit_first_tl}
  \begin{figure}[H]
  \centering
  \group_begin:
  \__numodel_apply_dsep:
  % --- Build the plot body ----------------------------------------
  \int_compare:nNnTF
```

```
        { \clist_count:N \l__numodel_yshort_clist } = { 1 }
      {
        % Single-series: preserve the legacy mark-only rendering.
        \edef\dm@coords
          { \mcoordsp{\l__numodel_eff_prefix_tl}{#2}{\tl_use:N \l_tmpa_tl} }
        \__numodel_draw_plot:n
          {
            \addplot[only~marks,~mark=*,~mark~size=0.5pt]~coordinates~{\dm@coords};
            \addlegendentry{model~point}
            #4
          }
      }
      {
        % Multi-series: one mark-only \addplot per kept variable, with
        % cycling colour-blind safe colour, and a text legend.  The
        % legend's mark is enlarged so the colour stays visible there.
        \tl_clear:N \l__numodel_plotbody_tl
        \int_zero:N \l__numodel_yidx_int
        \clist_map_inline:Nn \l__numodel_yshort_clist
          {
            \int_incr:N \l__numodel_yidx_int
            \tl_put_right:Ne \l__numodel_plotbody_tl
              {
                \exp_not:N \addplot
                  [ only~marks ,~mark=* ,~mark~size=0.5pt ,
                    color = \__numodel_series_color:n
                              { \int_use:N \l__numodel_yidx_int } ,
                    legend~image~post~style = { mark~size=2pt } ]
                coordinates {
                  \mcoordsp { \l__numodel_eff_prefix_tl } { #2 } { ##1 }
                } ;
                \exp_not:N \addlegendentry
                  { $ \use:c { \l__numodel_eff_prefix_tl ##1 text } $ }
              }
          }
        \tl_put_right:Nn \l__numodel_plotbody_tl {#4}
        \exp_args:NV \__numodel_draw_plot:n \l__numodel_plotbody_tl
      }
    \group_end:
    \caption{$\tl_use:N \l__numodel_yqty_tl$($\use:c{\l__numodel_xname_tl text}$)\text{-diagram}}
    % If the project-specific worksheet system is loaded, pick the
    % '-ws' label inside a worksheet; otherwise always the bare label.
    \cs_if_exist:NTF \ifinworksheet
      { \ifinworksheet{\label{fig:#5-ws}}{\label{fig:#5}} }
      { \label{fig:#5} }
    \end{figure}
  }

\ExplSyntaxOff

% ============================================================
% GRAPHIC MODEL - STYLES AND MACROS
% ============================================================
%
% Forrester-diagram conventions:
%
%   Stock              [stock]    Rectangle              v, s, T, x
%   Valve / inflow     [valve]    Circle on a thick      a, dv, ds
%                                 flow pipe
%   Auxiliary          [aux]      Plain circle           F_res, F_air
%   Constant           \constnode Circle + dashes        m, k, g
%
% Use \dgridx and \dgridy as the grid spacings for consistent positioning.
% ============================================================

% === Grid spacing -- defaults set via \numodelsetup {graphscalex, graphscaley, stockwidth} ===
```

```
% === Styles ===
\tikzset{
 % Scale coordinates to the grid spacing
 gridscale/.style={x=\dgridx cm, y=\dgridy cm},
 % Stock (state variable): rectangle
 stock/.style={
  rectangle, draw, thick,
    minimum width=3 em,
    minimum height=2 em,
  font=\small
 },
 % Auxiliary (intermediate variable): plain circle
 aux/.style={
  circle, draw, thick,
  inner sep=2pt,
    minimum size=2 em,
  font=\scriptsize
 },
 % Constant: circle (dashes drawn by the \constnode macro)
 const/.style={
  circle, draw, thick,
  inner sep=2pt,
    minimum size=2 em,
  font=\scriptsize
 },
 % Valve (legacy alias): circle on the flow pipe.  Kept for
 % backwards compatibility with user code that calls \flowarrow
 % manually.  The package's emit helpers now pick one of the
 % style variants below based on \numodelsetup{valve-style=...}.
 valve/.style={
  circle, draw, thick, fill=white,
  minimum size=2 em,
  font=\small
 },
 % Valve variant: Forrester valve symbol -- two filled triangles
 % (white inside) whose tips meet at the centre.  The symmetry
 % axis stands perpendicular to the (horizontal) flow.  The
 % bounding box is tall and narrow so the apex angle at the
 % centre is roughly 60 degrees (sharper than the 90 degrees of
 % a square box), and the horizontal "ribs" along the top and
 % bottom edges close the two triangles.
 valve-forrester/.style={
  shape=rectangle,
  draw=none, fill=none,
  inner sep=0pt, outer sep=0pt,
  %minimum width=\dgridx*0.18 cm,
    minimum width=1 em,
  %minimum height=\dgridy*0.30 cm,
    minimum height=2 em,
  path picture={%
   \draw[thick, fill=white]
    (path picture bounding box.north west) --
    (path picture bounding box.north east) --
    (path picture bounding box.center) --
    cycle;
   \draw[thick, fill=white]
    (path picture bounding box.south west) --
    (path picture bounding box.south east) --
    (path picture bounding box.center) --
    cycle;
  }
 },
 % Valve variant: empty circle (no label).
 valve-circle/.style={
  circle, draw, thick, fill=white,
```

```
 minimum size=2 em,
 font=\small
},
% Valve variant: circle with the variable's display label inside.
valve-edu/.style={
 circle, draw, thick, fill=white,
 minimum size=2 em,
 font=\small
},
% Flow pipe variants (body without arrow tip, used for the
% segment from open-end / cloud to the valve):
flowpipe/.style={line width=3pt},                    % legacy alias
flowpipe-filled/.style={line width=3pt},
flowpipe-hollow/.style={
 line width=1pt,
 double=white, double distance=3pt
},
% Flow pipe variants WITH arrow tip (used for the segment from
% valve to stock, or the open-end side of an outflow).  Both
% variants use the Stealth arrow tip.  The hollow form
% uses the double-line magic length 0pt-3-0 so that the arrow
% tip merges cleanly with the gap between the two strokes.
flowpipe-filled-tip/.style={line width=3pt, arrows={-Stealth[inset=0pt, angle=30:1em]}},
flowpipe-hollow-tip/.style={
 line width=1pt,
 double=white, double distance=3pt,
 arrows={-Stealth[inset=0pt, angle=30:1em]}
},
% Flow pipe variants with the Cloud arrow tip on the open end:
% inflow keeps the Stealth arrow head at the stock side.
flowpipe-filled-cloudin/.style={
 line width=3pt,
 arrows={Cloud[fill=white, line width=0.8pt]-Stealth[inset=0pt, angle=30:1em]}
},
flowpipe-hollow-cloudin/.style={
 line width=1pt,
 double=white, double distance=3pt,
 arrows={Cloud[fill=white, line width=0.8pt]-Stealth[inset=0pt, angle=30:1em]}
},
% Outflow with cloud at the open end: first a Stealth tip at the
% line end, then the cloud behind it (in arrows.meta spec, list
% multiple tips separated by a dot so the shaft stops at the
% Stealth tip and does not extend into the cloud; order = from
% line outwards, hence Stealth.Cloud).
flowpipe-filled-cloudout/.style={
 line width=3pt,
 arrows={-{Stealth[inset=0pt, angle=30:1em].Cloud[fill=white, line width=0.8pt]}}
},
flowpipe-hollow-cloudout/.style={
 line width=1pt,
 double=white, double distance=3pt,
 arrows={-{Stealth[inset=0pt, angle=30:1em].Cloud[fill=white, line width=0.8pt]}}
},
% Causal arrow (thin arrow for dependencies)
causal/.style={thick, arrows={-Stealth[length=0.5em]}}
}

% Default values for the body / tip macros so a manual call to
% \flowarrow (outside \graphicmodel) keeps working.  The
% \graphicmodel pipeline overwrites these per render.
\providecommand{\nmflowbody}{flowpipe-filled}
\providecommand{\nmflowtip}{flowpipe-filled-tip}
\providecommand{\nmflowtipcloudin}{flowpipe-filled-cloudin}
\providecommand{\nmflowtipcloudout}{flowpipe-filled-cloudout}

% === Macro: constant node with dashes ===
```

```
% Usage: \constnode{name}{(x,y)}{$label$}
\newcommand{\constnode}[3]{%
 \node[const] (#1) at (#2) {#3};%
 \draw[thick] ([xshift=-3mm]#1.west) -- (#1.west);%
 \draw[thick] (#1.east) -- ([xshift=3mm]#1.east);%
}

% === Macro: flow pipe with valve (inflow) ===
% Draws ONE continuous flow arrow ending at the stock.  The starting
% position is the same regardless of cloud presence; passing any
% non-blank optional argument switches to the cloud-tipped variant
% so the arrow's tail becomes a white-filled cloud (drawn as a real
% PGF arrow tip, not a separate node - moves with the line).  The
% valve overlays the middle of the arrow as a separate white-filled
% node drawn later by the build pipeline.
% Style keys: \nmflowtip / \nmflowtipcloudin.
% Usage: \flowarrow[<any>]{<valve-coord>}{<stock-node>}
\NewDocumentCommand{\flowarrow}{ O{} m m }{%
 \IfBlankTF{#1}{%
  \draw[\nmflowtip] ([xshift=-3em]#2) -- ([xshift=0.5mm]#3.west);%
 }{%
  \draw[\nmflowtipcloudin] ([xshift=-3em]#2) -- ([xshift=0.5mm]#3.west);%
 }%
}

% === Macro: flow pipe with valve (outflow) ===
% Mirror of \flowarrow.  The arrow always ends at the same offset
% past the valve coordinate; passing a non-blank optional argument
% switches to the cloud-tipped variant (cloud at the open end).
% Usage: \flowoutarrow[<any>]{<stock-node>}{<valve-coord>}
\NewDocumentCommand{\flowoutarrow}{ O{} m m }{%
 \IfBlankTF{#1}{%
  \draw[\nmflowtip] (#2.east) -- ([xshift=4em]#3);%
 }{%
  \draw[\nmflowtipcloudout] (#2.east) -- ([xshift=5em]#3);%
 }%
}

% === Macro: flow pipe between two stocks ===
% One continuous arrow from source stock through valve coord into
% target stock.  No cloud option (no open end exists when both
% sides are stocks).
% Usage: \flowbetweenarrow{<source-stock>}{<valve-coord>}{<target-stock>}
\newcommand{\flowbetweenarrow}[3]{%
 \draw[\nmflowtip] (#1.east) -- ([xshift=0.5mm]#3.west);%
}

% === Macro: curved inflow branch for a shared valve ===
% Used when one inflow valve feeds more than one stock.  The primary
% target is drawn with the standard straight \flowarrow; every
% secondary target is drawn with this macro, which arcs UNDER the
% intervening stock(s).  Stocks sit on the bottom row of the
% Forrester diagram, so the area below is empty -- routing the
% branch below avoids collisions with the aux/const rows and the
% causal arrows that live above.  Both endpoints are passed
% unanchored so TikZ picks the border-intersection that best
% matches the curve's tangent at each end.
% Usage: \flowarrowbent{<valve-coord>}{<target-stock>}
\newcommand{\flowarrowbent}[2]{%
 \draw[\nmflowtip] (#1) to[bend right=30] (#2);%
}

% === Macro: curved outflow branch for a shared valve ===
% Mirror of \flowarrowbent.  Used when one outflow valve drains more
% than one stock.  Routes the branch BELOW the intervening stocks
% (same rationale as the inflow variant: nothing else lives below
```

```
% the stocks row).  Endpoints unanchored so TikZ chooses the
% cleanest border intersection.
% Usage: \flowoutarrowbent{<source-stock>}{<valve-coord>}
\newcommand{\flowoutarrowbent}[2]{%
  \draw[\nmflowtip] (#1) to[bend right=30] (#2);%
}
```
⟨/package⟩

9.1 Translation files

Each `numodel-<LANG>.def` file populates the lookup table consulted by `\__numodel_kw:n`. The package loads exactly the file matching the current `syntax` key, via `\InputIfFileExists` (and hence `kpse`), so additional languages can be supplied through `TEXMFHOME/tex/latex/numodel/numodel-<LANG>.def` with no package rebuild. The two shipped files follow.

9.1.1 `numodel-EN.def` – English (XMILE)

XMILE v1.0 OASIS style: ALL-CAPS, case sensitive. Sets the default decimal mark to `point`.

⟨∗EN⟩
```
\ProvidesExplFile{numodel-EN.def}{2026/05/23}{0.5.0}
  {numodel~keyword~table~---~English~(XMILE)}
\cs_new:Npn \__numodel_kw_EN_if:           { IF~ }
\cs_new:Npn \__numodel_kw_EN_then:         { ~THEN~ }
\cs_new:Npn \__numodel_kw_EN_then_nl:      { ~THEN }
\cs_new:Npn \__numodel_kw_EN_else:         { ~ELSE~ }
\cs_new:Npn \__numodel_kw_EN_else_nl:      { ELSE }
\cs_new:Npn \__numodel_kw_EN_endif:        { ~ENDIF }
\cs_new:Npn \__numodel_kw_EN_endif_nl:     { ENDIF }
\cs_new:Npn \__numodel_kw_EN_and:          { ~AND~ }
\cs_new:Npn \__numodel_kw_EN_or:           { ~OR~ }
\cs_new:Npn \__numodel_kw_EN_not:          { NOT~ }
\cs_new:Npn \__numodel_kw_EN_sign:         { SIGN }
\cs_new:Npn \__numodel_kw_EN_abs:          { ABS }
\cs_new:Npn \__numodel_kw_EN_sqrt:         { SQRT }
\cs_new:Npn \__numodel_kw_EN_exp:          { EXP }
\cs_new:Npn \__numodel_kw_EN_ln:           { LN }
\cs_new:Npn \__numodel_kw_EN_sin:          { SIN }
\cs_new:Npn \__numodel_kw_EN_cos:          { COS }
\cs_new:Npn \__numodel_kw_EN_tan:          { TAN }
\cs_new:Npn \__numodel_kw_EN_asin:         { ARCSIN }
\cs_new:Npn \__numodel_kw_EN_acos:         { ARCCOS }
\cs_new:Npn \__numodel_kw_EN_stop:         { ~STOP~ }
\cs_new:Npn \__numodel_kw_EN_th_model:     { model }
\cs_new:Npn \__numodel_kw_EN_th_initvals:  { initial~values }
\cs_new:Npn \__numodel_kw_EN_contfoot:     { Continued~on~next~page }
\cs_new:Npn \__numodel_kw_EN_conthead:     { (Continued) }
\cs_new:Npn \__numodel_kw_EN_dsep_default: { point }
```
⟨/EN⟩

9.1.2 `numodel-NL.def` – Dutch (CoachTaal)

CoachTaal convention. Sets the default decimal mark to `comma`.

⟨∗NL⟩
```
\ProvidesExplFile{numodel-NL.def}{2026/05/23}{0.5.0}
  {numodel~keyword~table~---~Dutch~(CoachTaal)}
\cs_new:Npn \__numodel_kw_NL_if:           { Als~ }
\cs_new:Npn \__numodel_kw_NL_then:         { ~Dan~ }
\cs_new:Npn \__numodel_kw_NL_then_nl:      { ~Dan }
\cs_new:Npn \__numodel_kw_NL_else:         { ~Anders~ }
\cs_new:Npn \__numodel_kw_NL_else_nl:      { Anders }
\cs_new:Npn \__numodel_kw_NL_endif:        { ~EindAls }
\cs_new:Npn \__numodel_kw_NL_endif_nl:     { EindAls }
\cs_new:Npn \__numodel_kw_NL_and:          { ~EN~ }
\cs_new:Npn \__numodel_kw_NL_or:           { ~OF~ }
\cs_new:Npn \__numodel_kw_NL_not:          { NIET~ }
\cs_new:Npn \__numodel_kw_NL_sign:         { Teken }
```

```
\cs_new:Npn \__numodel_kw_NL_abs:          { Abs }
\cs_new:Npn \__numodel_kw_NL_sqrt:         { Sqrt }
\cs_new:Npn \__numodel_kw_NL_exp:          { Exp }
\cs_new:Npn \__numodel_kw_NL_ln:           { Ln }
\cs_new:Npn \__numodel_kw_NL_sin:          { Sin }
\cs_new:Npn \__numodel_kw_NL_cos:          { Cos }
\cs_new:Npn \__numodel_kw_NL_tan:          { Tan }
\cs_new:Npn \__numodel_kw_NL_asin:         { Arcsin }
\cs_new:Npn \__numodel_kw_NL_acos:         { Arccos }
\cs_new:Npn \__numodel_kw_NL_stop:         { ~Stop~ }
\cs_new:Npn \__numodel_kw_NL_th_model:     { model }
\cs_new:Npn \__numodel_kw_NL_th_initvals:  { startwaarden }
\cs_new:Npn \__numodel_kw_NL_contfoot:     { Wordt~vervolgd~op~volgende~pagina }
\cs_new:Npn \__numodel_kw_NL_conthead:     { (Vervolg) }
\cs_new:Npn \__numodel_kw_NL_dsep_default: { comma }
⟨/NL⟩
```