Framica

3D Space Frame Analysis Workbench

Developer Manual

Written in Wolfram Mathematica

This document describes the internal architecture, data structures, rendering pipeline, and extension patterns of the Framica interactive structural analysis workbench.

Version 5 | July 26, 2026

Contents

1 Overview

Framica is a 3D structural frame modelling and finite element analysis workbench written entirely in Wolfram Mathematica. It provides an interactive graphical interface for:

- constructing structural geometry by defining line segments;
- assigning section properties, member releases and node supports;
- defining load cases (primary and combination) and applying loads;
- running a 3D space frame FEA solver;
- viewing force diagrams, deformed shapes and support reactions;
- exporting member results to a searchable Dataset database.

The entire interface is contained in a single `DynamicModule` expression named `Framica`, supported by a library of standalone helper functions defined in cells that must be evaluated before the module is displayed.

2 Notebook and Cell Structure

A Framica notebook is organised into the following cells, which must be evaluated in order from top to bottom before `Framica` is displayed:

Cell	Contents	Description
1	Section library	Section property data and lookup functions
2	FEA solver	`fwRunFEA` and assembly routines
3	Geometry utilities	`fwCleanupGeometry`, `fwSanitizeLines`, etc.
4	Drawing primitives	`fwDrawNodeRelease`, `fwDrawLoadEntry`, etc.
5	Member detail viewer	`fwWBBuildDetail`, `fwWBOpenDetail`
6	Save/load project	`fwSaveProject`, `fwLoadProject`
7	Global DB helpers	`fwMemberDBSingle`, `fwMemberDBFlat`, `fwMemberDBNested` and `$fw*` initialisations
8	**Framica**	`Framica = DynamicModule[...]`

All cells above Cell 8 must be evaluated before `Framica` is displayed. Re-evaluating `Framica` from scratch resets all local state. Use the existing instance rather than re-evaluating when possible.

3 Global State Variables

Several dollar-sign globals are shared between the `Framica DynamicModule` and the outside kernel, allowing external cells to read current state without accessing module internals.

Variable	Purpose
`$fwWBCurrentLines`	Mirror of the current line list; updated silently whenever `lines` changes
`$fwWBCurrentSelected`	Mirror of selected member indices
`$fwWBGroups`	Association mapping group names to lists of line coordinates

Variable	Purpose
`$fwWBGroupVisible`	Association mapping group names to Boolean visibility flags
`$fwWBActiveGroups`	List of currently active group names (used for load assignment and Recall)
`$fwMemberDBAll`	Nested Association: load case name → member key → DB entry Association
`$fwMemberDB`	DB for the most recently built load case (shortcut reference)
`$fwMemberDBName`	String name of the most recently built load case
`$fwDBButtonRow`	The rendered `Pane` of DB access buttons shown in the Results tab

Table 1: Global state variables shared between Framica and external cells.

4 DynamicModule Structure

The entire interface is one `DynamicModule`. Its skeleton is:

```
Framica = DynamicModule[
  { (* local state variables *) },
  (
    Dynamic[...];                (* mirror lines/selection to globals *)
    grpPanel = Function[{}, ...];  (* group panel factory function *)
    Column[{
      Panel@Row[{...}],          (* top toolbar: mode, display, save/open *)
      Panel@Row[{...}],          (* view toolbar: viewpoint, clip, member slider *)
      Row[{
        Pane[                    (* left panel: tab view *)
          TabView[{
            "Geometry"   -> Column[...],
            "Properties" -> Column[...],
            "Loads"      -> Column[...],
            "File"       -> Column[...],
            "Results"    -> Column[...]
          }], ...],
        Spacer[10],
        Panel[Dynamic[           (* right panel: 3D viewport *)
          Which[
            Length[points] == 0,        Graphics3D[...],
            showResultsViewer,          Module[...],
            True,                       Module[...]
          ], TrackedSymbols :> {...}]]
      }]
    }]
  ),
  Initialization :> (...),
  SynchronousInitialization -> True
```

]

5 Local State Variables

All geometry, properties, loads and UI state live as local variables inside the `DynamicModule`. They persist across re-evaluations of inner `Dynamic[]` expressions because `DynamicModule` maintains its own local symbol store, identified by suffixed names such as `lines$$1234`.

5.1 Geometry

Variable	Type	Description
`lines`	List of `{{x1,y1,z1},{x2,y2,z2}}`	All member line segments; the primary geometry store
`points`	List of `{x,y,z}`	Canonical node list derived from `lines` by `fwCanonicalPoints`
`structureScale`	Real	Characteristic size used for scaling drawing primitives (sphere radii, arrow lengths, etc.)

5.2 Properties

Variable	Type	Description
`memberProps`	Association: index → Association	Per-member properties. Keys: `Library`, `Section`, `StartRelease`, `EndRelease`, `ForceType`, `Orientation`
`nodeProps`	Association: index → Association	Per-node properties. Keys: `Release`, `SpringKx`, `SpringKy`, `SpringKz`, `SpringType`

5.3 Loads

Variable	Type	Description
`loadCases`	Association: name → Association	All load cases. Each entry has `"Type"` (Primary or Combination) and either a `"Loads"` list or a `"Components"` Association
`currentLoadCase`	String or `None`	Currently selected load case name

5.4 FEA Results

Variable	Type	Description
`feaResults`	Association or `None`	Full output from `fwRunFEA`; `None` when no results
`feaReady`	Boolean	`True` when valid results are available
`feaLoadCase`	String or `None`	Load case selected for the next FEA run
`feaErrorMsg`	String or `None`	Error message if the last FEA run failed
`showResultsViewer`	Boolean	Whether the viewport shows FEA results or plain geometry

5.5 Undo and Redo

Variable	Type	Description
`undoStack`	List of state Associations	Each entry is `<|"L"->lines,"MP"->memberProps,"NP"->nodeProps,"LC`
`redoStack`	List of state Associations	Same structure; populated by Undo, cleared by any new operation

Every operation that modifies geometry, properties or loads follows this pattern:

```
undoStack = Append[undoStack, <|"L"->lines, "MP"->memberProps,
                                "NP"->nodeProps, "LC"->loadCases|>];
redoStack = {};
(* ... perform modification ... *)
```

5.6 UI Display Controls

Variable	Description
`editMode`	`"Members"` or `"Nodes"`; controls click behaviour in the viewport
`selectedMembers`	List of currently selected member indices
`selectedNodes`	List of currently selected node indices
`show3DMembers`	Boolean; render extruded 3D section profiles
`showDirectionArrows`	Boolean; render member direction arrows
`showPointLabels`	Boolean; show node number labels
`showLineLabels`	Boolean; show member number labels
`showDesignMembers`	Boolean; highlight design member groups with coloured overlay
`showLoads`	Boolean; render load arrows and symbols
`graySelection`	Boolean; gray out selected members for navigation
`checkConnectivity`	Boolean; colour members by connected component
`plotSize`	Integer; viewport height in pixels
`plotWidth`	Integer; viewport width in pixels
`feaViewPt`	`{vx,vy,vz}`; 3D camera viewpoint
`feaXClip`, `feaYClip`, `feaZClip`	`{min,max}` pairs; clip box ranges for each axis

Variable	Description
`feaResultType`	String; one of `"Deformed"`, `"Moment My"`, `"Moment Mz"`, `"Shear Vy"`, `"Shear Vz"`, `"Axial N"`, `"Torsion Tx"`
`feaDispScale`	Real; deflection magnification factor
`feaDiagScale`	Real; force diagram scale factor (metres per kN)
`feaSelMem`	Integer; currently highlighted FEA member index (0 = none)
`feaShowVals`	Boolean; show numerical values on force diagrams
`feaMomTension`	Boolean; plot moments on the tension side
`activeTab`	String; currently selected tab name
`dissolvedMembers`	List of member indices excluded from design group calculations

Table 2: UI display control variables.

6 The Two Dynamic Contexts

6.1 The Globals Mirror

The outermost `Dynamic` expression silently keeps global variables synchronised with local state:

```
Dynamic[
  $fwWBCurrentLines     = lines;
  $fwWBCurrentSelected = selectedMembers;
  Null,
  TrackedSymbols :> {lines, selectedMembers, selectedNodes, editMode}]
```

This fires whenever `lines` or `selectedMembers` change, making the current geometry and selection available to external notebook cells without exposing module internals.

6.2 The Viewport Dynamic

The 3D viewport is wrapped in a `Dynamic` with an explicit `TrackedSymbols` list:

```
Panel[Dynamic[
  Which[
    Length[points] == 0,
      Graphics3D[{Text[Style["Add␣lines...", 16, Gray]]}, ...],
    showResultsViewer && TrueQ[feaResults["Success"]],
      Module[{res, rPts, ...},    (* FEA results viewer *)
        ...
        Graphics3D[{wirePrims, diagPrims, nodePrims, ...}]],
    True,
      Module[{validLines, pts, ...},   (* geometry viewer *)
        ...
        Graphics3D[{mPrims, nPrims, ...}]]
  ],
  TrackedSymbols :> {lines, points, feaResults, feaResultType,
    feaDispScale, feaDiagScale, feaSelMem, feaShowVals, feaMomTension,
```

```
    feaViewPt, feaXClip, feaYClip, feaZClip,
    showResultsViewer, activeTab, selectedMembers, selectedNodes,
    editMode, showLoads, showPointLabels, showLineLabels, show3DMembers,
    showDirectionArrows, showDesignMembers, graySelection,
    checkConnectivity, structureScale, currentLoadCase, loadCases,
    plotSize}]]
```

> The `TrackedSymbols` list is critical for performance. Without it, Mathematica tracks all symbols and the viewport re-renders on every keystroke anywhere in the notebook. When adding a new reactive variable, add it to this list.

7 Tab Descriptions

7.1 Geometry Tab

Contains `OpenerView` panels for all geometry editing operations.

7.1.1 Add Line

Coordinate input fields for P1 and P2. **Add** appends a single line. **Add & Chain** additionally sets P1 ← P2 for rapid chaining.

7.1.2 Connect Nodes

Joins two selected nodes with a new line if not already connected.

7.1.3 Delete

Removes selected lines or nodes. Deleting a node removes all incident lines.

7.1.4 Subdivide

Splits selected members into sub-segments by number, ratio list, or length list. Calls `fwRedistributeLoadsAfter` to remap any existing loads onto the new segments.

7.1.5 Offset Lines

Copies selected members offset by $(\Delta x, \Delta y, \Delta z)$. Optionally copies section assignments and loads from all Primary load cases.

7.1.6 Extrude Nodes

Creates new lines from each selected node in direction $(\Delta x, \Delta y, \Delta z)$.

7.1.7 Move Nodes

Translates selected nodes; all incident lines follow automatically.

7.1.8 Intersect

Splits members at geometric intersection points using `fwSplitTargetsUntilStable`. Can operate on selected members only or all members.

7.1.9 General Geometry Operation Pattern

All geometry operations follow this invariant sequence:

```
(* 1. save undo state *)
undoStack = Append[undoStack,
  <|"L"->lines, "MP"->memberProps, "NP"->nodeProps, "LC"->loadCases|>];
redoStack = {};

(* 2. modify lines *)
lines = ...;

(* 3. clean up *)
lines = fwCleanupGeometry[lines];

(* 4. recompute derived data *)
points          = fwCanonicalPoints[fwSanitizeLines[lines]];
structureScale  = fwStructureScale[points];

(* 5. rematch properties to new indices *)
memberProps = fwMatchMemberProps[lines, origGP];
nodeProps   = fwMatchNodeProps[points, origNP];
```

7.2 Properties Tab

7.2.1 Section

Hierarchical library / type / section popup menus backed by the `sectionLibraries` Association. Displays a 2D profile sketch and a scrollable property table when a section is selected. **Assign** writes `Library` and `Section` keys to `memberProps` for all selected members.

7.2.2 Orientation

Rotation angle (degrees) of the section local y-axis about the member axis. Stored as `Orientation` in `memberProps`.

7.2.3 Member Releases

Start and end release conditions selected from `fwMemberReleaseTypes` (e.g. Fixed, Pinned, moment release). Stored as `StartRelease` and `EndRelease`.

7.2.4 Force Type

Assigns one of `"Tension+Compression"`, `"Tension Only"` or `"Compression Only"` to control which members carry which force types in the FEA assembly.

7.2.5 Node Supports

Assigns a support type from `fwNodeReleaseTypes` to selected nodes. Stored as `Release` in `nodeProps`.

7.2.6 Spring Supports

Assigns translational spring stiffnesses K_x, K_y, K_z (in kN/mm) and a spring type (Both / Tension only / Compression only) to selected nodes.

7.3 Loads Tab

7.3.1 Load Case Types

Type	Structure
Primary	`<|"Type"->"Primary", "Loads"->{...}|>`
Combination	`<|"Type"->"Combination", "Components"-><|"Case1"->1.35, "Case2"->1.5, ...|>|>`

7.3.2 Load Entry Types

Each entry in the `"Loads"` list of a Primary case is an Association:

Type	Required Keys
`"Point Force"`	`Node`, `Fx`, `Fy`, `Fz`
`"Point Moment"`	`Node`, `Mx`, `My`, `Mz`
`"UDL"`	`Member`, `wx`, `wy`, `wz`, `Coord`
`"Trapezoid"`	`Member`, `w1`, `w2`, `a`, `b`, `dx`, `dy`, `dz`, `Coord`
`"Member Point"`	`Member`, `Position` (fraction 0–1), `Fx`, `Fy`, `Fz`, `Coord`
`"Gravity"`	`gx`, `gy`, `gz` (acceleration in m/s^2)
`"Thermal"`	`Member`, `dT` (temperature change in °C)

`Coord` is either `"Global"` or `"Local"`.

7.4 File Tab

Save and load `.wl` project files via `fwSaveProject` and `fwLoadProject`. The project file serialises geometry, properties, load cases and UI state as a Wolfram Association. CSV import and export for nodes and elements is also available.

7.5 Results Tab

7.5.1 FEA Workflow

1. Select a load case from the popup menu.
2. Press **Run FEA** → calls `fwRunFEA[lines, memberProps, nodeProps, loadCases, feaLoadCase]`.
3. On success: `feaResults` is populated, `feaReady` is set `True`, `showResultsViewer` is set `True`, and the clip box is reset to fit the model.
4. The viewport switches to the FEA results viewer.
5. Diagram type, scale sliders and the member selector control the display.

7.5.2 Build DB Workflow

1. Run FEA for load case A, then press **Build DB**.
2. Run FEA for load case B, then press **Build DB** again.
3. Repeat for all desired load cases.
4. DB buttons appear in the Results panel for each built case.
5. The flat and nested buttons access all accumulated cases together.

> Do *not* press **Clear results** between FEA runs when building the DB across multiple load cases. Clear results resets the DB globals. The DB accumulates across runs only when

results are not cleared.

8 FEA Results Structure

`fwRunFEA` returns an Association with the following top-level keys:

Key	Content
`"Success"`	Boolean
`"LoadCase"`	String
`"Error"`	Error message string (if `Success` is `False`)
`"nNodes"`	Integer
`"nMembers"`	Integer
`"pts"`	$n \times 3$ real matrix of node coordinates
`"U"`	Displacement vector, length $6n$ (order: $u_x, u_y, u_z, r_x, r_y, r_z$ per node, in metres/radians)
`"reactions"`	Reaction force/moment vector, length $6n$
`"bcDOFs"`	List of constrained DOF indices (1-based)
`"lines"`	List of line coordinate pairs in the same order as input `lines`
`"members"`	List of per-member result Associations (see below)
`"Fvec"`	Applied force vector, length $6n$
`"SpringKdiag"`	Spring stiffness diagonal vector, length $6n$

Table 3: Top-level keys of the FEA result Association.

Each entry in `"members"` is an Association with:

Key	Content
`"m"`	Member index (1-based)
`"a"`, `"b"`	Start and end node coordinate vectors
`"L"`	Member length in metres
`"T"`	12×12 transformation matrix (local $\rightarrow$ global DOFs)
`"axes"`	`{ex, ey, ez}` — local axis unit vectors in global frame
`"dofE"`	List of 12 global DOF indices for this member
`"forces"`	List of force Associations sampled along the member length

Table 4: Per-member result Association keys.

Each entry in `"forces"` contains:

Key	Units	Description
`N`	kN	Axial force (positive = tension)
`Vy`	kN	Shear force in local y
`Vz`	kN	Shear force in local z
`My`	kN·m	Bending moment about local y
`Mz`	kN·m	Bending moment about local z
`Tx`	kN·m	Torsional moment
`dy`	mm	Lateral deflection in local y
`dz`	mm	Lateral deflection in local z

9 Member Database Structure

`$fwMemberDBAll` is a nested Association:

```
$fwMemberDBAll
  "LoadCase1"
    1  -> <|"LeadMember" -> 1,
            "Group"          -> {1, 2, 3},
            "IsDesignGroup" -> True,
            "Section"        -> "457x152x52␣UB",
            "Library"        -> "UK",
            "LoadCase"       -> "LoadCase1",
            "Forces"         -> <|"N"  -> {Nmin, Nmax},
                                  "Vy" -> {Vymin, Vymax},
                                  "Vz" -> {Vzmin, Vzmax},
                                  "My" -> {Mymin, Mymax},
                                  "Mz" -> {Mzmin, Mzmax},
                                  "Tx" -> {Txmin, Txmax}|>,
            "Detail"         -> DynamicModule[...]|>
    6  -> <|...|>
    ...
  "LoadCase2"
    1  -> <|...|>
    ...
```

Integer keys match the `LeadMember` index (the first member of each design group). For non-design-group structures every member is its own entry.

Forces stores rounded `{min, max}` pairs: N, Vy, Vz, My, Mz to 2 decimal places (kN or kN·m); Tx to 4 decimal places.

Detail stores a live `DynamicModule` containing `fwWBBuildDetail` with an interactive displacement scale slider. In a `Dataset`, this renders as `...` dots that expand the full force diagram widget inline on click.

> Always use `Dataset[$fwMemberDBAll[lc]]` to display the DB for load case `lc`. Do not reconstruct the Association via `Table` and then wrap in `Dataset` — this causes premature evaluation of the `DynamicModule` and the `...` expansion dots disappear.

10 Viewport Rendering Pipeline

10.1 Geometry Viewer (Normal Mode)

When no FEA results are available, or when `showResultsViewer` is `False`, the viewport renders the structural model:

```
validLines     = fwSanitizeLines[lines]
pts            = fwCanonicalPoints[validLines]
hiddenIndices  = members belonging to invisible groups
resolvedLoads  = fwResolveLoadCase[currentLoadCase, loadCases]

mPrims  <- per-member primitives:
             colour by ForceType / selection / section type
             EventHandler Tube for click-to-select
             optional 3D extrusion via fwExtrudeMember
             release markers via fwDrawMemberRelease
             direction arrows via fwDrawMemberDirection
             Tooltip showing member index and section

nPrims  <- per-node primitives:
             fwDrawNodeRelease (support symbol)
             fwDrawNodeSprings (spring symbol if springs assigned)
             clickable Sphere (red when selected, blue in Nodes mode)

lPrims  <- optional node number and member number labels

ldPrims <- fwDrawLoadEntry per resolved load entry

dPrims  <- optional design group overlay dashed lines

Graphics3D[{mPrims, nPrims, lPrims, ldPrims, dPrims}, ...]
```

10.2 FEA Results Viewer

When `showResultsViewer` is `True` and results are available:

```
visM     = members whose midpoint lies inside the clip box
wirePrims  <- undeformed ghost lines: res["lines"][[visM]]
dLines     <- deformed shape (when feaResultType == "Deformed"):
                cubic Hermite interpolation, visM members only
                dvy = feaDispScale*(h1*uL[[2]]+h2*uL[[6]]
                                    +h3*uL[[8]]+h4*uL[[12]])
                dvz = feaDispScale*(h1*uL[[3]]-h2*uL[[5]]
                                    +h3*uL[[9]]-h4*uL[[11]])
nodePrims  <- node spheres at rest positions
supPrims   <- support symbols from fwDrawNodeRelease
springPrims<- spring symbols from fwDrawNodeSprings
diagPrims  <- force/moment diagram primitives from
              fwResultOneMemberDiagram (non-Deformed modes)
memberClickLines <- EventHandler Line for member selection
resLPrims  <- node and member number labels
resDPrims  <- design group overlay (if showDesignMembers)
```

```
nodeTooltipPrims <- transparent Spheres with Tooltip
                    showing displacements and (optionally)
                    support reactions on hover

Graphics3D[{wirePrims, diagPrims, nodePrims, supPrims,
            springPrims, dLines, memberClickLines,
            resLPrims, resDPrims, nodeTooltipPrims}, ...]
```

The clip box (`feaXClip`, `feaYClip`, `feaZClip`) filters both `visM` and all label and symbol primitives independently, so clipping is consistent across all rendered elements.

10.2.1 Hermite Interpolation

The deformed shape uses cubic Hermite basis functions to give C^1 continuity (displacement and slope match at nodes). For a position parameter $\xi \in [0, 1]$ along a member of length L:

$$h_1 = 1 - 3\xi^2 + 2\xi^3, \quad h_2 = L\,\xi(1-\xi)^2, \quad h_3 = 3\xi^2 - 2\xi^3, \quad h_4 = L\,\xi^2(\xi - 1)$$

The y- and z-direction deflections in the local frame are:

$$\delta_y = h_1 u_2 + h_2 u_6 + h_3 u_8 + h_4 u_{12}$$
$$\delta_z = h_1 u_3 - h_2 u_5 + h_3 u_9 - h_4 u_{11}$$

where u_i are the local DOF displacements. The result is transformed to global coordinates using the member local axis vectors.

11 Key Utility Functions

Function	Purpose
`fwCleanupGeometry[lines]`	Remove duplicate lines and near-zero-length members
`fwSanitizeLines[lines]`	Remove degenerate (zero-length) lines
`fwCanonicalPoints[lines]`	Extract a sorted, deduplicated node list from line endpoints
`fwStructureScale[points]`	Compute a characteristic length for drawing primitive sizing
`fwRebuildModel[lines, mp, np]`	Recompute `points`, reindex properties
`fwMatchMemberProps[lines, origGP]`	Rematch member properties to new line indices after geometry change
`fwMatchNodeProps[points, origNP]`	Rematch node properties to new point indices after geometry change
`fwDesignMembers[lines, memberProps]`	Compute the list of design member groups (collinear members with same section)
`fwDesignGroupOf[mi, dmg, dissolved]`	Find the design group containing member index `m`
`fwOrderDesignGroup[grp, lines]`	Sort the members of a design group end-to-end in physical order
`fwAggregateDesignMember[grp, res]`	Combine FEA force results across all members of a design group

Function	Purpose
`fwGroupToIndices[name, lines]`	Map a named group to current member indices (by coordinate matching)
`fwResolveLoadCase[lc, loadCases]`	Expand a combination load case to a flat list of scaled primary loads
`fwRedistributeLoadsAfterSplit[old, new, lc]`	Remap member loads from old line indices to new indices after subdivision or intersection
`fwRunFEA[lines, mp, np, lc, lcName]`	Assemble and solve the 3D space frame stiffness system
`fwResultOneMemberDiagram[mem, type, scale, vals]`	Build force diagram `Graphics3D` primitives for on member
`fwWBBuildDetail[grp, res, dispScale, mt]`	Build the multi-panel force diagram display for a member or design group
`fwWBOpenDetail[grp, res, mt]`	Open `fwWBBuildDetail` in a floating notebook window
`fwDrawNodeRelease[pt, release, scale]`	Draw a support symbol at a node
`fwDrawNodeSprings[pt, props, scale]`	Draw spring symbols at a node
`fwDrawMemberRelease[pt, dir, release, scale]`	Draw an end-release marker at a member end
`fwDrawMemberDirection[p1, p2, scale]`	Draw a direction arrow on a member
`fwDrawLoadEntry[ld, pts, lines, scale, mp]`	Draw a load arrow or symbol for a single load entry
`fwSaveProject[fn, ...]`	Serialise the full model state to a `.wl` file
`fwLoadProject[fn]`	Deserialise model state from a `.wl` file

Table 5: Key utility functions called by Framica.

12 Adding New Features

12.1 New Geometry Operation

1. Add an `OpenerView` in the Geometry tab.
2. Follow the push-undo / modify / cleanup / rebuild pattern (§**??**).
3. Add any new UI variables to the `DynamicModule` variable list.
4. Add symbols to `TrackedSymbols` only if the viewport must redraw when they change.

12.2 New Load Type

1. Add a menu item to the `newLoadType PopupMenu`.
2. Add a `Which` branch for the input fields `Dynamic`.
3. Add a `Which` branch in the **Apply** button action.
4. Add a rendering branch in `fwDrawLoadEntry`.
5. Handle the new type in `fwRunFEA`.

12.3 New Result Diagram

1. Add a `RadioButton` in the Results tab diagram row.
2. Add a branch in `fwResultOneMemberDiagram`.

12.4 New Reactive Variable

1. Declare it in the `DynamicModule` variable list with a default value.
2. Add it to the `TrackedSymbols` list of the viewport `Dynamic` if the viewport should redraw when it changes.
3. Initialise it in the `Initialization :>` block at the end of the `DynamicModule`.

12.5 New Global Variable

1. Add an initialisation line to the global helpers cell (Cell 7).
2. Also initialise it in the `Initialization :>` block.
3. If it should be cleared by **Clear results**, add it to that button's action.

13 Modal, Thermal, and Programmatic Subsystems

This section documents the subsystems added after the original release.

13.1 Mass and Section Data

`fwGetFEAProps` now also returns `"Mu"` (mass per length in tonne/m), `"h"` (section depth in m, for thermal gradient) and `"Alpha"` (thermal expansion coefficient). Mass per length is taken from the section's tabulated `"Mass"` field (kg/m, converted to tonne/m); if absent it falls back to ρA with `$fwRho`$= 7.85$ tonne/m^3. Units are chosen so that with $\mathbf{K}$ in kN/m and $\mathbf{M}$ in tonnes, eigenvalues come out directly as ω^2 in s^{-2}.

The torsion constant lookup accepts either `"It"` or `"IT"` (RHS/SHS tables use the latter), falling back to an estimate $0.05(I_y + I_z)$ when neither is present.

13.2 Modal Analysis

`fwLocalMass3D[mu, L, rx2]`
Consistent 12×12 element mass matrix in local axes (translational, rotational and torsional terms; `rx2`$= (I_y + I_z)/A$ is the polar radius of gyration squared).

`fwRunModal[lines, mProps, nProps, "Modes"->n]`
Assembles global $\mathbf{K}$ and consistent $\mathbf{M}$ (rigid joints; supports and grounded springs respected), applies boundary conditions, and solves `Eigensystem[{Kff, Mff}]` on the free DOFs. Returns an association with `"Frequencies"` (Hz), `"Periods"` (s), `"Omega"`, `"Points"` and `"Modes"` (each a mode shape with per-node translations, normalised to unit maximum).

`fwOpenModeViewer[res, lines]`
Opens a separate resizable window (`CreateDocument` + `Manipulate`) animating the mode shapes. The mode data is captured by value so the window is self-contained.

`FramicaModal["id"]`
Public wrapper that runs modal analysis on a stored run.

Member end-releases are treated as rigid for modal analysis (exact for rigidly-jointed frames). The lowest cantilever/beam frequencies match Euler–Bernoulli theory to better than 0.1 %.

13.3 Thermal Loads

`fwThermalFEF3D[EA, EIz, alpha, h, dT, grad]` returns a 12-vector local equivalent-nodal-load vector: an axial term $N = EA\,\alpha\,\Delta T$ (ends pushed apart) and gradient end-moments $M = EI_z\,\alpha\,\Delta T_{\text{grad}}/h$ about local z. In the solver it is condensed and transformed through the same

path as UDL/point fixed-end forces, so it acts along the member's *local* axis for any orientation. A `"Thermal"` load entry carries `"dT"` and `"Gradient"` keys.

13.4 Run Store and IDs

`$FramicaRuns` is a session-persistent association keyed by run id. `Framica[fwArg_:None]` coerces its argument to a string id (`fwRunId`); the `Initialization` restores that run's state from `$FramicaRuns` (and self-loads the package + clears any `Global‘Framica` shadow before rendering). Public helpers: `FramicaRuns[]`, `FramicaRuns["id"]`, `FramicaReset[...]`. `FramicaImport["id", nodes, members]` builds `lines` from node/member tables (reusing `fwCleanupGeometry`/`fwRebuildModel`) and writes them into the run.

13.5 DXF and CSV I/O

`fwExportDXF`/`fwImportDXF` write and parse R12 ASCII DXF (`LINE`, `LWPOLYLINE`, `POLYLINE`/`VERTEX`). The CSV exporters were corrected so nodes are `#,X,Y,Z,Release` and members are `#,N1,N2,...`, which round-trip through the existing validators and `fwImportGeometryCSV`.

13.6 Public Symbols

The paclet now exports `Framica`, `FramicaImport`, `FramicaRuns`, `FramicaReset`, `FramicaModal`, `FramicaManual` and `FramicaExamples`. The manuals and example projects are bundled as paclet *assets* and opened via `FramicaManual[...]` / `FramicaExamples[]`.

14 Known Design Constraints

Single DynamicModule
All state is local to one module. This gives clean encapsulation but means the entire interface resets if `Framica` is re-evaluated. Use the existing rendered instance; do not re-evaluate unless a full reset is intended.

`Method -> "Queued"` on all computation buttons
Every button that performs non-trivial computation must use `Method -> "Queued"` to avoid blocking the Mathematica front end. Omitting this causes the interface to freeze during long operations.

TrackedSymbols on every Dynamic
Always specify `TrackedSymbols :> {...}` explicitly on `Dynamic[]` expressions inside the viewport. Without it, Mathematica tracks all symbols and performance degrades severely.

Property reindexing after geometry changes
`lines` indices change whenever geometry is modified (lines may be reordered or renumbered by `fwCleanupGeometry`). Always use `fwMatchMemberProps` and `fwMatchNodeProps` after any operation that modifies `lines`. Direct index assignment to `memberProps` or `nodeProps` will produce incorrect results after the next geometry operation.

DynamicModule in Dataset
Do not reconstruct the DB Association via `Table` before wrapping in `Dataset`. This evaluates the `DynamicModule` stored in `"Detail"`, causing either premature rendering or silent dropping of the entry. Always call `Dataset[$fwMemberDBAll[lc]]` directly.

DB accumulation across load cases
`$fwMemberDBAll` accumulates one key per load case as Build DB is pressed. Pressing **Clear results** resets the DB. Run FEA and press Build DB for all desired load cases *before* pressing Clear results.

Section library dependency
`fwRunFEA` reads elastic modulus E, cross-sectional area A, and second moments I_{yy}, I_{zz}, I_t from `sectionLibraries` via `fwGetSectionProperties`. Members without an assigned section cannot be analysed; the solver will fail with an appropriate error message.

A Load Case Data Structure Summary

```
loadCases = <|
  "Dead" -> <|
    "Type"  -> "Primary",
    "Loads" -> {
      <|"Type"->"UDL", "Member"->3, "wx"->0, "wy"->0, "wz"->-5,
        "Coord"->"Global"|>,
      <|"Type"->"Point Force", "Node"->7, "Fx"->0, "Fy"->0, "Fz"->-10|>,
      <|"Type"->"Gravity", "gx"->0, "gy"->0, "gz"->-9.81|>
    }|>,
  "Live" -> <|
    "Type"  -> "Primary",
    "Loads" -> {
      <|"Type"->"Trapezoid", "Member"->5, "w1"->-3., "w2"->-6.,
        "a"->0., "b"->4., "dx"->0, "dy"->0, "dz"->1,
        "Coord"->"Global"|>
    }|>,
  "ULS" -> <|
    "Type"        -> "Combination",
    "Components" -> <|"Dead"->1.35, "Live"->1.5|>
  |>
|>
```

B Member Properties Data Structure Summary

```
memberProps = <|
  1 -> <|"Library"        -> "UK",
         "Section"        -> "457x152x52 UB",
         "StartRelease"   -> "Fixed",
         "EndRelease"     -> "Pinned",
         "ForceType"      -> "Tension+Compression",
         "Orientation"    -> 0|>,
  2 -> <|"Library"        -> "UK",
         "Section"        -> "152x152x23 UC",
         "StartRelease"   -> "Fixed",
         "EndRelease"     -> "Fixed",
         "ForceType"      -> "Tension+Compression",
         "Orientation"    -> 0|>
|>
```

C Node Properties Data Structure Summary

```
nodeProps = <|
  1 -> <|"Release"     -> "Fixed"|>,
```

```
  5 -> <|"Release"     -> "Pinned (Free Rotation)"|>,
  9 -> <|"Release"     -> "Fixed (Internal)",
         "SpringKx"    -> 10.,
         "SpringKy"    -> 0.,
         "SpringKz"    -> 25.,
         "SpringType" -> "Both"|>
|>
```