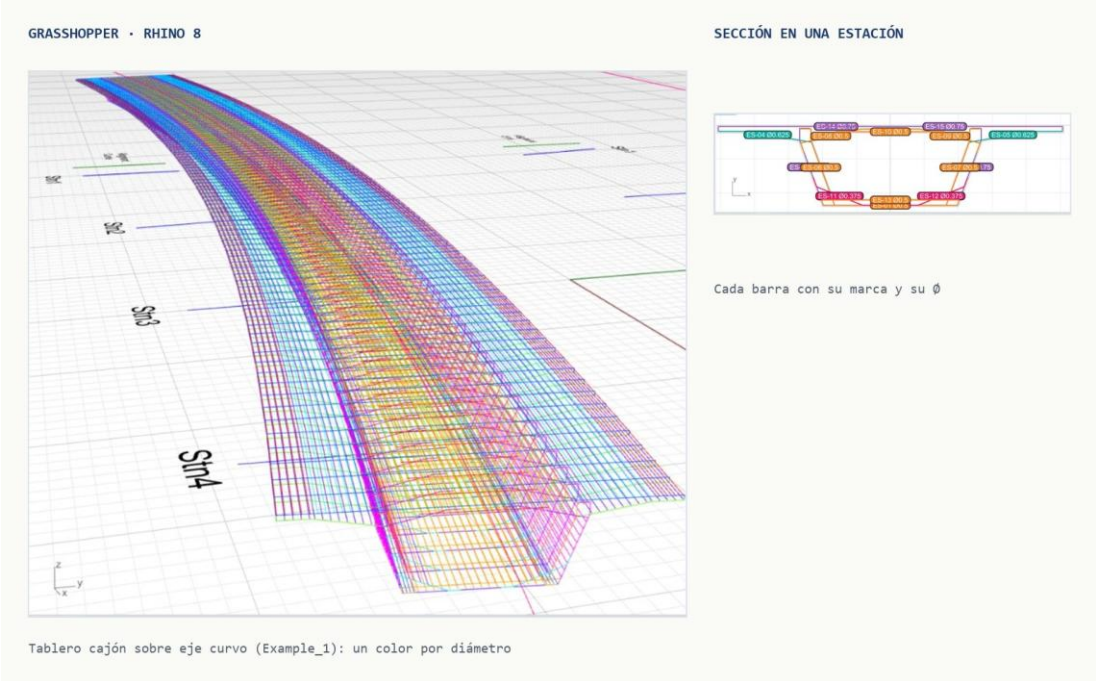
GH_BRIDGE

User guide · version 1.0.0 · Rhino 8

GH_Bridge is a bridge authoring system for Rhino and Grasshopper. Parametric cross sections are evaluated along a 3D alignment, swept into the solid deck, and reinforced with transversal and longitudinal bars that carry their own hooks, lap splices and marks. The bars can then be sent to Revit as native Rebar through Rhino.Inside.Revit.

It is not a generic rebar plugin. The core is the section and the alignment: the steel is defined against named points of the section, never in absolute coordinates, so it follows the deck when its depth, width or thicknesses change.



Box girder on a curved axis: transversal and longitudinal rebar, one standard color per diameter.

The Bridge tab, by workflow stage

1. Info · Plugin information
   - BrInfo
2. Alignment · Axis and stations
   - Axis StnCL StnD DivAx DivAxD DeStn
3. Section · Cross sections
   - ParamSec RdSecJ SecCrv SecVars WrSecJ DeSec
4. Model · Deck geometry
   - EvalSec SecSwp SecView PlnCut
5. Transversal · Transversal rebar
   - RbCat RbCov RbDef RbStir RbSec RbView
   - RbHook RbSeam
6. Longitudinal · Longitudinal rebar
   - RbLongN RbLongP RbLong RbLongAsm RbLongView RbArc
   - RbSplice RbLongJoin
7. Tools · Checks, take-off and output
   - DeBar RbGroup RbSched RbBend RbBD RbClash

```
    └── RbNudge  Bake  RbSplBar
8. Revit  ·  Native Revit rebar (inside Rhino.Inside.Revit only)
    └── Crv2Rvt  Rebar2Rvt
9. Curves  ·  Curve tools for manual rebar
    ├── CrvAng  DiscCrv  DivCrvN  DivCrv  ExtCrv  ExtCrv2
    ├── ExtPtCrv  LinePt  PrjCrv2  PrjCrv  OrientPl  RebuildCrv
    └── RebuildCrvH  SplCrv
```

1. INSTALLATION

Requirements

- Rhino 8 with Grasshopper (Windows).
- Revit panel only: Revit 2025 + Rhino.Inside.Revit.

Option A · Rhino Package Manager

In Rhino run `_PackageManager`, search for `gh-bridge` and install it. Restart Rhino.

Option B · Manual installation (.zip)

Download GH_Bridge-v1.0.0.zip from daapcloud.com/gh-bridge or Food4Rhino, then:

- Close Rhino.
- Copy the GH_Bridge folder (GH_Bridge.gha and its two .dll files) to:
  `%APPDATA%\Grasshopper\Libraries\GH_Bridge\`
- Optional, Revit: copy GH_Bridge.Revit\GH_Bridge.Revit.gha to:
  `%APPDATA%\Grasshopper\Libraries-Inside-Revit-2025\GH_Bridge.Revit\`
- Start Rhino and open Grasshopper: the Bridge tab appears. Place Bridge Info (BrInfo) to confirm the version.

Notes

- If Windows blocked the downloaded files: right-click each file > Properties > Unblock.
- Do not rename the .gha or its .dll files.
- The Revit .gha only loads inside Revit with Rhino.Inside.Revit; in plain Rhino it is ignored.

2. WORKFLOW

Every GH_Bridge definition follows the same order. Each stage feeds the next one:

1. Axis

```
Axis → StnCL / StnD → DivAx
```

A Rhino curve becomes the alignment. Stations come from control lines (abutments, piers, skew) or from explicit distances; DivAx subdivides each span for the bars.

2. Section

```
RdSecJ / SecCrv → ParamSec → SecVars
```

The cross section is read from JSON (or digitized from a closed curve). ParamSec exposes its variables as inputs; SecVars changes them span by span.

3. Model

```
EvalSec → SecSwp
```

The section is placed on every station plane and swept into the solid deck (Brep).

4. Transversal

```
RbCat + RbCov + RbDef → RbSec → RbStir
```

Bars are defined against the section points (anchors, waypoint, hooks) and resolved on the cover rails at every station.

5. Longitudinal

```
RbLong / RbLongN / RbLongP → RbLongAsm → RbSplice → RbLongJoin → RbArc
```

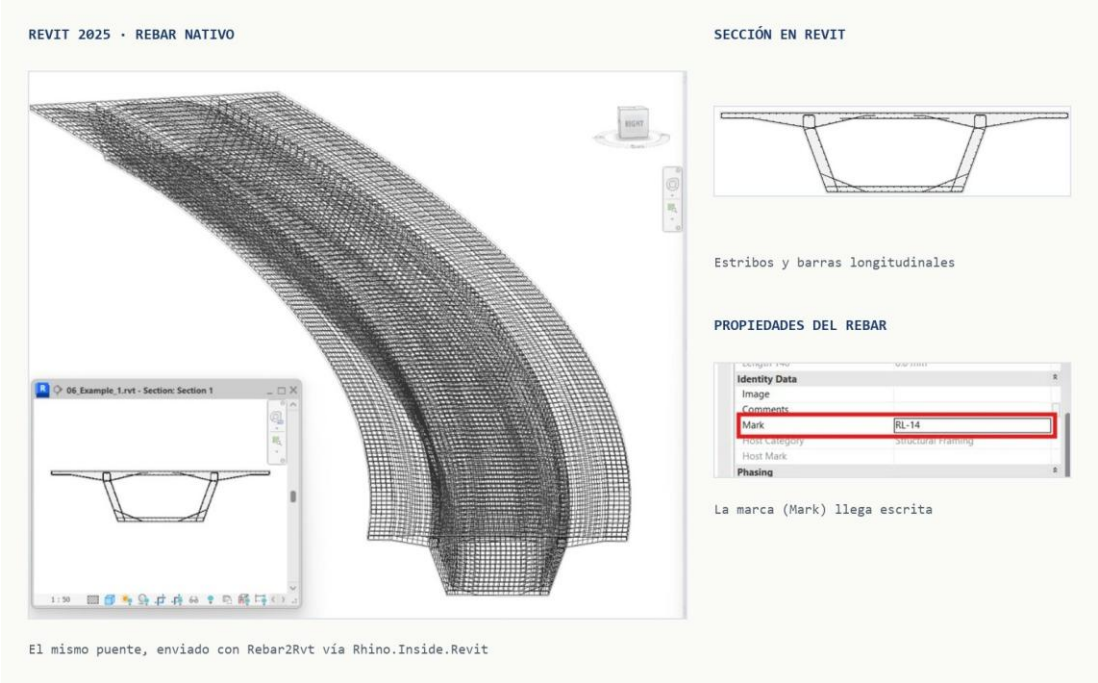
Packages of bars by spacing, count or points follow the axis in 3D; lap splices and fitting to true arcs come last.

6. Output

```
RbClash → RbNudge → RbGroup → RbSched → Bake / Rebar2Rvt
```

Clash check and correction, grouping by mark with standard colors and layers, steel take-off, and bake to Rhino or send to Revit.

Hover any component or port in Grasshopper for its built-in help. Example_1 and Example_2, included in the .zip, show the whole chain working.



The same bars in Revit as native Rebar, with their marks and hook types.

3. KEY CONCEPTS

3.1 The section JSON

A section is a JSON file with these main fields:

Field	Content
`Id, Name, Unit`	Number, name, and length unit (m, cm, mm, ft, in).
`Variables`	Name = value pairs: the dimensions you will drive (D, Wtop, Tweb...).
`Points`	Construction points; each has Id and Coord [Y, Z], numbers or expressions of the variables.
`Loops`	Closed outlines: Outer (concrete) and Inner (voids), each with its own points.
`Circles`	Round voids or piers: center point and radius.

- Z points down: TOPC = [0, 0] is the top center and BOTC = [0, D] the bottom. TOPC is the point that rides the axis.
- Naming: outer loop points E1, E2...; inner (void) points I1, I2.... The first letter tells the rebar which rail the point is on.
- Expressions accept + - * / ^ and parentheses, e.g. "0.5*Wtop - Tweb".
- A point can also be defined relative to others with Reference and Type:

Type	Reference	Meaning
`Euclidean`	`0 / 1 / 2`	Absolute; offset from one point; or Y from the first and Z from the second.
`Polar`	`2`	Local frame from ref[0] towards ref[1]; Coord = [along, across].
`CY / CZ`	`3`	On the line ref[0]–ref[1], at the same Y (or Z) as ref[2].

Composite decks: give the first section points named G, G1...Gn and add more sections to RdSecJ with "+"; each one is seated with its TOPC on those points. Example_2 ships a JSON section library to start from.

3.2 Anchors for transversal bars

RbDef places a bar on the cover rail between a Start and an End anchor. An anchor is written as:

Form	Example	Meaning
Point Id	`E3`	Exactly on that rail point.
From:To@dist	`E3:E4@0.25`	On the edge from E3 to E4, 0.25 (document units) from E3.
From:To@pct%	`I4:I5@30%`	On the edge from I4 to I5, at 30 % of its length.

- Same Start and End: a closed stirrup around the whole rail. Way forces the longer path around the rail.

– Hooks are Ids from the hook catalog (RbCat), e.g. H90, H135. A directional hook is written "Dir:Ref1:Ref2@leg1:leg2".

3.3 Splice and distribution tokens

Token	Example	Meaning
`F@val`	`F@6.0`	Fixed piece length.
`R@val`	`R@8.5`	Fills the rest with pieces up to val (at most one R token).
`P@val`	`P@0.27`	Lap centered at val: up to 1 is a fraction of the bar, above 1 a distance.
`N@val`	`3@0.5, R@2.0, 3@0.5`	DivAxD: N gaps of val, the rest at R, ordered from start to end.

In RbSplice, Alt staggers the splices of every second bar of the same mark (by dAlt, or half the pitch if empty); Sym mirrors the pattern from both ends.

3.4 Standard colors and layers

Bars get one standard color per diameter (with variants per mark), and RbGroup assigns standard layers Bridge::Rebar::Ø<mm>::<mark>. Preview, bake and file use the same colors, and concrete elements get standard greys per type.

4. COMPONENT REFERENCE

1. Info · Plugin information

a. Bridge Info (BrInfo)

GH_Bridge version, author and license

INPUTS:

- **(none)**

OUTPUTS:

- **Plugin (P)**: GH_Bridge version and author
- **License (L)**: License terms
- **Links (U)**: Source code, documentation and contact

2. Alignment · Axis and stations

a. Axis From Curve (Axis)

Builds a station-parametrized bridge alignment axis from a Rhino curve

INPUTS:

- **Curve (Crv)**: Alignment curve
- **Name (N)**: Axis name

OUTPUTS:

- **Axis (Ax)**: bridge axis

b. Stations From Control Lines (StnCL)

One station per control line crossing the Axis (e.g. abutment/pier/skew lines), ordered and measured along the axis

INPUTS:

- **Axis (Ax)**: bridge axis
- **Control Lines (CL)**: Lines/curves crossing the axis, one per station (e.g. abutment, pier or skew lines); named Stn1, Stn2, ... in the order given here
- **Tolerance (Tol)**: Intersection tolerance

OUTPUTS:

- **Stations (Stns)**: Stations, one per control line, ordered along the axis; Station = running length from the first one
- **Planes (Pl)**: Station planes (same order as Stations) - feed into EvaluateSectionComponent's Plane input
- **Gaps (Gaps)**: Running-length gap from each station to the next (Count - 1 values) - not the same thing as Divide Axis's own Sp input, which sets the rebar gaps WITHIN a tramo
- **Length (L)**: Total bridge length, from the first station to the last

c. Stations From Distances (StnD)

Stations an Axis at explicit running-length distances, oriented perpendicular to the Axis (no Control Lines)

INPUTS:

- **Axis (Ax)**: bridge axis
- **Distances (D)**: Running-length distances from the start of the Axis curve, e.g. 0, 25, 50, 75, 100; named Stn1, Stn2, ... in the order given here

OUTPUTS:

- **Stations (Stns)**: Stations at the requested distances, perpendicular to the Axis
- **Planes (Pl)**: Station planes (same order as Stations)
- **Gaps (Gaps)**: Running-length gap from each station to the next, in output order (Count - 1 values) - not the same thing as Divide Axis's own Sp input, which sets the rebar spacing WITHIN a tramo
- **Length (L)**: Total length covered, from the first distance given to the last

d. Divide Axis (DivAx)

Subdivides the domain between two named stations at a fixed spacing, for rebar placement along one tramo

INPUTS:

- **Axis (Ax)**: bridge axis (needed to evaluate points/tangents between the two named stations)
- **Stations (Stns)**: already-resolved stations to pick the domain from (e.g. StnCL/StnD output) - also used as direction-interpolation guides inside the domain
- **From (From)**: name of the station marking one end of the domain (e.g. "Stn1")
- **To (To)**: name of the station marking the other end of the domain (e.g. "Stn2") - order doesn't matter, the domain always runs low-to-high running length
- **Spacing (Sp)**: running-length spacing between infill stations inside the domain

OUTPUTS:

- **Stations (Stns)**: From, To (with their original names) and the fixed-spacing infill stations between them (unnamed)
- **Planes (Pl)**: Station planes (same order as Stations)

e. Divide Axis By Distribution (DivAxD)

Subdivides the domain between two named stations using a symmetric N@val/R@val token distribution

INPUTS:

- **Axis (Ax)**: bridge axis (needed to evaluate points/tangents between the two named stations)
- **Stations (Stns)**: already-resolved stations to pick the domain from (e.g. StnCL/StnD output) - also used as direction-interpolation guides inside the domain
- **From (From)**: name of the station marking one end of the domain (e.g. "Stn1")

- **To (To)**: name of the station marking the other end of the domain (e.g. "Stn2") - order doesn't matter, the domain always runs low-to-high running length
- **Distribution (Dist)**: Tokens ordered start-to-end, e.g. "3@0.5", "R@2.0", "3@0.5" (mirrored) - see component description

OUTPUTS:

- **Stations (Stns)**: From, To (with their original names) and the distribution's infill stations between them (unnamed)
- **Planes (Pl)**: Station planes (same order as Stations)

f. Deconstruct Station (DeStn)

Breaks a list of Stations down into their name, running length and plane

INPUTS:

- **Stations (Stns)**: Axis stations

OUTPUTS:

- **Name (Nm)**: Station name
- **Station (Stn)**: Running length along the axis
- **Plane (Pl)**: Station plane

3. Section · Cross sections

a. Parametric Section (ParamSec)

Exposes one input per Section Variable so its dimensions can be driven live. Accepts RdSecJ's assembled list: one block of inputs and one output per section (receiver on top, shared variables only in the receiver's block)

INPUTS:

- **Section (Sec)**: Parametric section template (Variables + expressions), or RdSecJ's receiver + adhered sections

OUTPUTS:

- **Section (Sec)**: Modified section (same geometry/topology, Variables overridden)

b. Read Section JSON (RdSecJ)

Reads a Section from a JSON string or file path (own format or a reference-schema section export). Zoom in and use "+" to adhere more sections: each one is seated (its TOPC) on every G/G1..Gn point of the first section

INPUTS:

- **File Path (Path)**: Receiver section: file path to a .json file, or a JSON description pasted in directly

OUTPUTS:

- **Section (Sec)**: Receiver section, followed by one copy of each adhered section per G point

c. Section From Curve (SecCrv)

Converts a closed planar curve (drawn in Rhino, or imported from a .dwg) into a Section

INPUTS:

- **Outer Curve (Out)**: Closed planar curve describing the outer boundary (straight segments only - no arcs)
- **Inner Curves (In)**: Closed planar curves describing voids/holes (e.g. a box girder cell)
- **Plane (Pl)**: Section plane; defaults to the outer curve's own planar frame
- **Name (Nm)**: Section name
- **Unit (U)**: reference-schema-style unit code (m/cm/mm/ft/in); defaults to the active Rhino document's model unit

OUTPUTS:

- **Section (Sec)**: cross section

d. Section Set Variables (SecVars)

Builds one Section per station from a template, carrying Variable overrides forward tramo by tramo

INPUTS:

- **Section (Sec)**: parametric template (Variables + expressions)
- **Stations (Stns)**: stations to build one Section per - order can be arbitrary, matched by Name
- **Overrides (Ov)**: "StationName: Var=Value, Var2=Value2" entries - one per station where a change starts; later stations carry it forward until the next override

OUTPUTS:

- **Sections (Sec)**: one Section per station, same order as Stations

e. Write Section JSON (WrSecJ)

Serializes a Section to a JSON string compatible with the reference schema's section JSON format

INPUTS:

- **Section (Sec)**: cross section
- **Indented (I)**: Pretty-print the JSON output

OUTPUTS:

- **Json Text (Txt)**: JSON description

f. Deconstruct Section (DeSec)

Breaks a Section down into its points, loops and circles for inspection

INPUTS:

- **Section (Sec)**: cross section

OUTPUTS:

- **Name (Nm)**: Section name
- **Unit (U)**: Section unit

- **Points (P)**: Top-level construction points
- **Loops (L)**: Section loops
- **Circles (C)**: Section circles

4. Model · Deck geometry

a. Evaluate Section (EvalSec)

Evaluates a Section's geometry (outline curves, points) for preview or downstream use

INPUTS:

- **Section (Sec)**: cross section
- **Plane (Pl)**: Where to place the section in 3D; defaults to the world XY plane

OUTPUTS:

- **Curves (C)**: Outline curves: one per loop, then one per circle (circular piers, round duct voids)
- **Points (P)**: All resolved section points (top-level and loop points)

b. Section Sweep (SecSwp)

Sweeps a Section's loops along the Axis, through stationed curves (from EvalSec fed a list of station planes), into a solid Brep

INPUTS:

- **Rail (R)**: The Axis alignment curve the sweep follows between stations
- **Curves (C)**: Per-station loop curves - one branch per station, one curve per loop in the same order at every station (EvalSec's Curves output, fed a list of station planes). Branches {piece; station} (grafted Sections) are swept piece by piece - the station index is the last path index that varies
- **Tolerance (Tol)**: Sweep/join/cap tolerance; defaults to the active document's absolute tolerance

OUTPUTS:

- **Solid (S)**: One solid per piece (one for a single Section; deck + each girder for a grafted assembly) - a capped tube, or (for a hollow section) the outer tube minus every inner tube
- **Walls (W)**: The per-loop swept skin surfaces (open, before capping), for inspection

c. Section Viewer (SecView)

Previews a Section with its own colors (Outer=green, Inner=red, TOPC/BOTC=purple, other points=navy) and every point Id labeled

INPUTS:

- **Section (Sec)**: cross section
- **Plane (Pl)**: Where to place the section in 3D; defaults to the world XY plane
- **Labels (Lbl)**: Show every point as a labeled dot (off by default - many dots slow the viewport down)

OUTPUTS:

- **Curves (C)**: Outline curves, one per loop (same as EvalSec)

- **Points (P)**: All resolved section points

d. Plane Cutter (PlnCut)

Builds a planar rectangular cutter Brep centered on each plane

INPUTS:

- **Planes (Pl)**: Planes to build cutters on
- **Size (Sz)**: Half-width of each square cutter
- **Tolerance (Tol)**: Tolerance for building the planar Brep

OUTPUTS:

- **Cutters (Cut)**: Planar cutter Breps, one per plane

5. Transversal · Transversal rebar

a. Hook Catalog (RbCat)

Reads the hook shape catalog (Id/angle/extension in bar diameters) from a JSON string or file path, or the built-in default if none is given

INPUTS:

- **File Path (Path)**: File path to a .json file, or a JSON description pasted in directly - leave empty to use the built-in default catalog (H90/H135/H180)

OUTPUTS:

- **Hooks (Hks)**: hook shapes in the catalog
- **Ids (Ids)**: hook Ids only, same order as Hooks - handy to feed a Value List

b. Rebar Cover (RbCov)

Bundles Cover Exterior/Interior into one Cover value for RbSec/RbLongAsm, passing the Section through

INPUTS:

- **Section (Sec)**: cross section, passed through unchanged (optional)
- **Cover Exterior (CovE)**: Concrete cover for Outer loop(s) - the rail moves inward by this much
- **Cover Interior (CovI)**: Concrete cover for Inner/void loop(s) - the rail moves outward by this much

OUTPUTS:

- **Section (Sec)**: the input Section, passed through
- **Cover (Cov)**: CoverExterior/CoverInterior bundled - wire into RbSec/RbLongAsm's Cover input

c. Transversal Bar (RbDef)

Defines one rebar bar: Id and diameter, Start/Waypoint/End anchors on the section rail, and hook

INPUTS:

- **Id (Id)**: bar identifier, e.g. "ES-01"
- **Diameter (Ø)**: bar diameter
- **Start (Sta)**: start anchor: a point Id, "From:To@dist", or "From:To@pct%"
- **Waypoint (Way)**: optional - an intermediate point Id forcing the LONGER of the two paths around the closed rail; leave empty to just take the shorter path (the usual case)
- **End (End)**: end anchor - same forms as Start; the same anchor as Start gives a closed stirrup around the whole rail
- **Hook (Hk)**: hook Id (or "Dir:Ref1:Ref2@leg1:leg2" for a directional hook aimed at a reference edge instead of the rail's own corner), matched against a Rebar Hook Catalog (Hooks/Ids output) - default for both ends unless overridden by HkS/HkE
- **Hook Start (HkS)**: override hook just for the Start end (same forms as Hook); leave empty to use Hook
- **Hook End (HkE)**: override hook just for the End end (same forms as Hook); leave empty to use Hook

OUTPUTS:

- **Bar (Bar)**: the resulting bar definition

d. Closed Stirrup (RbStir)

Opens closed stirrups at a corner with crossing overruns and inward hooks, the shape Revit recognizes as a closed stirrup

INPUTS:

- **Bars (Bars)**: Closed Rebar Bars (e.g. RbSec stirrups); open bars pass through unchanged
- **Corner (Cn)**: Index of the loop vertex where the stirrup closes
- **Overrun (Ov)**: How far each leg runs past the corner, in mm
- **Hook (Hk)**: Hook length at each end, in mm
- **Angle (A)**: Angle between each hook and its leg, in degrees (the hook turns back into the stirrup)

OUTPUTS:

- **Bars (Bars)**: Open stirrups ready for Rebar2Rvt
- **Geometry (G)**: The open stirrup polyline

e. Transversal Assemble (RbSec)

Builds cover-offset rails from a Section and resolves a list of bar definitions onto them, hooks included

INPUTS:

- **Section (Sec)**: cross section (one station)
- **Plane (Pl)**: Where the section is placed in 3D; defaults to the world XY plane
- **Cover (Cov)**: CoverExterior/CoverInterior bundle (RbCov output)
- **Bars (Bars)**: bar definitions (RbDef output)
- **Hooks (Hks)**: hook catalog (RbCat output)

OUTPUTS:

- **Section (Sec)**: the input Section, passed through - wire straight into the viewer's Section input (RbView / RbLongView) instead of cabling it separately
- **Bars (Bars)**: resolved bar geometry, hooks included

f. Transversal Viewer (RbView)

Previews a Section (muted) with its resolved Bars overlaid, one standard color per bar diameter

INPUTS:

- **Section (Sec)**: cross section, drawn muted for context (optional)
- **Plane (Pl)**: must match the plane Bars were resolved on; defaults to WorldXY
- **Bars (Bars)**: resolved bars (RbSec output)
- **Labels (Lbl)**: Label each bar with its Mark and diameter (off by default - many dots slow the viewport down)

OUTPUTS:

- **(none)**

g. Hook Preview (RbHook)

Draws one hook's shape in isolation (lead-in leg + the hook bend) for a given bar diameter

INPUTS:

- **Hook (Hk)**: one hook from a Rebar Hook Catalog (Hooks output)
- **Diameter (Ø)**: bar diameter, scales the hook's extension
- **Plane (Pl)**: where to draw it; the leg arrives along the plane's XAxis, the hook turns toward YAxis
- **Leg Length (Leg)**: length of the straight lead-in segment, for visualization only; 0 = auto (8x diameter)
- **Flip (Flip)**: turn toward -YAxis instead of +YAxis

OUTPUTS:

- **Curve (C)**: the hook's shape: leg + bend, as a polyline

h. Seam Match (RbSeam)

Normalizes winding direction and seam vertex across a set of closed curves for station-to-station correspondence

INPUTS:

- **Curves (C)**: Closed curves to normalize (one per station)
- **Planes (Pl)**: Reference plane per curve (single plane reused for all if only one given)
- **Flip (Flip)**: Flip the winding direction
- **Right (Right)**: Seam on the right side (false: left side)
- **Segments (Nseg)**: Advance the seam by this many extra vertices (can be negative)

OUTPUTS:

- **Curves (C)**: Seam-matched curves, same order as input
- **Bad (Bad)**: True where the curve needed a fallback repair

6. Longitudinal · Longitudinal rebar

a. Longitudinal By Count (RbLongN)

Defines a package of longitudinal bars (one Mark/Diameter) placed by a fixed count evenly spaced between two rail points

INPUTS:

- **Mark (Mk)**: shared mark for the whole package, e.g. "RL-01"
- **Diameter (Ø)**: bar diameter, shared by the whole package
- **Section (Sec)**: cross section used to resolve the real From-To distance and this package's offset rail
- **Offset Diameter (ØOfs)**: how far this package's rail sits off the raw section face (e.g. to clear the transversal bars) - not the structural cover; direction follows From/To's own E-/I- point Id
- **From (Fr)**: starting rail point Id (e.g. "E10") - the first bar lands here
- **To (To)**: ending rail point Id - the edge From->To the package is placed on
- **Count (Cnt)**: fixed number of bars, evenly spaced from From to To
- **Start Offset On (OffS)**: enable a start offset
- **Start Offset (dS)**: start offset distance
- **End Offset On (OffE)**: enable an end offset
- **End Offset (dE)**: end offset distance
- **Extend Start (ExtS)**: when Start Offset is on, dS pushes the first bar OUTWARD past From (same edge line) instead of trimming inward
- **Extend End (ExtE)**: when End Offset is on, dE pushes the last bar OUTWARD past To (same edge line) instead of trimming inward

OUTPUTS:

- **Bars (Bars)**: one longitudinal bar definition per point in the package
- **Count (N)**: how many bars this package generated

b. Longitudinal By Points (RbLongP)

Defines a package of longitudinal bars (one Mark/Diameter) placed at literal Section-local coordinates

INPUTS:

- **Mark (Mk)**: shared mark for the whole package, e.g. "RL-01"
- **Diameter (Ø)**: bar diameter, shared by the whole package
- **Points (Pts)**: one point per bar, already in the Rhino document's own units (no Section input to convert from) - X is the section's local horizontal axis, Y is vertical as seen on screen (positive up), origin = the section's TOPC; Z is ignored

OUTPUTS:

- **Bars (Bars)**: one longitudinal bar definition per point in the package
- **Count (N)**: how many bars this package generated

c. Longitudinal By Spacing (RbLong)

Defines a package of longitudinal bars (one Mark/Diameter) placed by a fixed spacing between two rail points

INPUTS:

- **Mark (Mk)**: shared mark for the whole package, e.g. "RL-01"
- **Diameter (Ø)**: bar diameter, shared by the whole package
- **Section (Sec)**: cross section used to resolve the real From-To distance and this package's offset rail
- **Offset Diameter (ØOfs)**: how far this package's rail sits off the raw section face (e.g. to clear the transversal bars) - not the structural cover; direction follows From/To's own E-/I- point Id
- **From (Fr)**: starting rail point Id (e.g. "E10") - distances are measured from here
- **To (To)**: ending rail point Id - the edge From->To the package is placed on
- **Spacing (Spc)**: fixed real spacing between bars
- **Mode (Mode)**: 0 = Beginning, 1 = Middle, 2 = End, 3 = Extreme (leftover distance distributed accordingly)
- **Start Offset On (OffS)**: enable a start offset
- **Start Offset (dS)**: start offset distance
- **End Offset On (OffE)**: enable an end offset
- **End Offset (dE)**: end offset distance
- **Extend Start (ExtS)**: when Start Offset is on, dS pushes the first bar OUTWARD past From (same edge line) instead of trimming inward
- **Extend End (ExtE)**: when End Offset is on, dE pushes the last bar OUTWARD past To (same edge line) instead of trimming inward

OUTPUTS:

- **Bars (Bars)**: one longitudinal bar definition per point in the package
- **Count (N)**: how many bars this package generated

d. Longitudinal Assemble (RbLongAsm)

Resolves each longitudinal bar's anchor against every station's cover-offset rail and joins the points into a polyline

INPUTS:

- **Section (Sec)**: cross section(s) - one Section broadcasts across every station, or a list matching Plane one-to-one for a tapering run
- **Plane (Pl)**: one Plane per station, in order along the run; with nothing wired, defaults to a single WorldXY station (same as RbSec)
- **Cover (Cov)**: CoverExterior/CoverInterior bundle (RbCov output)
- **Bars (Bars)**: longitudinal bar definitions (RbLong output)

OUTPUTS:

- **Section (Sec)**: the first station's Section, passed through - wire straight into the viewer's Section input (RbView / RbLongView) instead of cabling it separately

- **Cover (Cov)**: the cover-offset rail(s) actually used by the resolved bars, one branch per station - wire into Rebar Longitudinal Viewer's Cover input
- **Bars (Bars)**: resolved bar geometry, one per longitudinal bar

e. Longitudinal Viewer (RbLongView)

Previews a Section (muted) with a longitudinal package's Bars overlaid as per-station point dots, one standard color per bar diameter

INPUTS:

- **Section (Sec)**: cross section, drawn muted for context (optional)
- **Plane (Pl)**: plane the Section context is drawn on; defaults to WorldXY
- **Cover (Cov)**: cover-offset rail(s) (RbLongAsm's Cover output), drawn in magenta (optional)
- **Bars (Bars)**: resolved longitudinal bars (RbLongAsm output)
- **Labels (Lbl)**: Label each bar with its Mark and diameter (off by default - many dots slow the viewport down)

OUTPUTS:

- **(none)**

f. Fit Arcs (RbArc)

Rebuilds dense bar polylines as a few lines and true arcs within a tolerance (default 1 mm), for a clean export to Revit

INPUTS:

- **Curves (C)**: Bar curves to rebuild (polylines through every station, e.g. RbGroup's Geometry)
- **Tolerance (Tol)**: Maximum deviation from the original points, in millimetres

OUTPUTS:

- **Curves (C)**: Rebuilt curves: one line, one arc, or a chain of lines and arcs
- **Segments (N)**: Number of lines/arcs in each rebuilt curve

g. Lap Splice (RbSplice)

Defines a lap-splice rule (F@/R@ pattern, lap, staggering) for the given longitudinal bar Mark(s) - wire into RbLongJoin's Splice input

INPUTS:

- **Mark (Mk)**: Mark(s) this rule applies to (same Mk as RbLong/RbLongN/RbLongP)
- **Tokens (Tok)**: Ordered tokens: "F@val" fixed piece length, "R@val" fills the rest (at most one) - e.g. "R@8.5" or "F@6.0", "R@8.5". Or fixed lap positions only: "P@val" centers a lap at val (<= 1: fraction of the bar length, > 1: distance) - e.g. "P@0.27", "P@0.73"
- **Lap (Lap)**: Lap length between consecutive pieces, fixed, in document units
- **Symmetric (Sym)**: False: pattern applied start-to-end / True: mirror all but the last token at both ends
- **Alternate (Alt)**: Stagger splices: every 2nd bar of the same Mark gets its splice pattern shifted by dAlt

- **Alternate Shift (dAlt)**: Shift applied to alternate bars; leave empty for half the pattern pitch (first piece length - Lap)

OUTPUTS:

- **Splice (Spl)**: splice rule - wire into RbLongJoin's Splice input

h. Longitudinal Join (RbLongJoin)

Rebuilds each longitudinal bar (RbLongAsm output) as a clean interpolated trace, applies RbSplice lap splices by Mark and rediscretizes every piece to a minimum segment length

INPUTS:

- **Bars (Bars)**: longitudinal bars (RbLongAsm output)
- **Distance (Dist)**: minimum segment length of the final polylines, in document units; empty = 1 m, lowest allowed 0.30 m
- **Splice (Spl)**: optional splice rules (RbSplice output), matched by Mark; empty = bars come out whole
- **Max Length (MaxL)**: maximum commercial bar length, in document units; empty = 9 m. Bars with any piece longer than this are reported in Rej, and removed once any Splice rule is wired

OUTPUTS:

- **Bars (Bars)**: final bars (one per spliced piece), same Mark as the source bar - once Splice rules are wired, only bars within MaxL
- **Rejected (Rej)**: Marks removed because at least one of their pieces exceeds MaxL (with bar count and longest piece)

7. Tools · Checks, take-off and output

a. Deconstruct Rebar (DeBar)

Breaks a list of Rebar Bars down into their mark, diameter, steel grade and geometry

INPUTS:

- **Bars (Bars)**: Reinforcement bars

OUTPUTS:

- **Mark (Mk)**: Bar mark
- **Diameter (Ø)**: Bar diameter
- **Steel Grade (Grd)**: Reinforcement steel grade
- **Geometry (G)**: Bar centerline geometry

b. Group By Mark (RbGroup)

Groups resolved Rebar Bars (RbSec output, any tree shape) by Mark into one branch per distinct mark, for per-category Bake/export

INPUTS:

- **Bars (Bars)**: resolved bars (RbSec output) - any tree shape, e.g. one branch per station

- **Filter (Flt)**: optional - only keep these Mark(s) (e.g. "E-01"); leave empty to keep every mark. When given, output branch order follows THIS list's order instead of first-seen order

OUTPUTS:

- **Marks (Mk)**: distinct bar marks kept, one per branch of Geometry (same order) - use to build a per-branch Layer path downstream
- **Diameter (Ø)**: that mark's bar diameter, one per Mark
- **Geometry (G)**: bar geometry, grouped one branch per Mark
- **Bars (Bars)**: the kept bars themselves (Mark/Diameter/Geometry), grouped one branch per Mark - wire into Rebar2Rvt
- **Colors (Col)**: one color per Mark from the plugin's rebar standard (a fixed color per diameter, lighter/darker variants per mark), on the same branch as that Mark's Geometry - wire into Bake's Object Color
- **Layers (Ly)**: one standard layer path per Mark, "Bridge::Rebar::Ø<mm>::<Mark>", on the same branch as that Mark's Geometry - wire into Bake's Layer

c. Rebar Schedule (RbSched)

Steel take-off per Mark: diameter, pieces, total length and weight (kg), with an optional CSV

INPUTS:

- **Bars (Bars)**: Rebar Bars (any tree shape), e.g. RbSec and RbLongJoin outputs
- **CSV Path (Path)**: Optional .csv file to write the schedule to

OUTPUTS:

- **Marks (Mk)**: One row per Mark, in natural order
- **Diameter (Ø)**: Bar diameter, mm
- **Pieces (N)**: Number of pieces
- **Length (L)**: Total length, m
- **Weight (W)**: Total weight, kg
- **Table (T)**: The schedule as text, with totals

d. Bend Check (RbBend)

Flags Rebar Bars where a straight segment can't fit its own bend diameter at a vertex (too short, or too sharp a turn - same check)

INPUTS:

- **Bars (Bars)**: Reinforcement bars to check
- **Bend Diameter (Bd)**: Optional mandrel diameter in model units, for every bar. Leave empty to use each bar's own default: 4 x its diameter for closed stirrups/ties, 6 x for the rest

OUTPUTS:

- **Valid (V)**: True if that bar's bends all fit, one per input bar
- **Messages (M)**: Empty if valid, otherwise which segment(s) came up short and by how much
- **Points (P)**: Midpoints of every offending segment, across all bars, for viewport highlighting

e. Bend Diameter (RbBD)

Standard (ACI 318-style) minimum bend diameter for a bar diameter in inches, tiered by bar size

INPUTS:

- **Diameter (Ø)**: Bar diameter in inches (e.g. from the Rebar Size value list)
- **Stirrup Or Tie (St)**: True for a stirrup/tie (closed loop) bar, false for a straight/longitudinal bar

OUTPUTS:

- **Bend Diameter (Bd)**: Standard minimum bend (mandrel) diameter, in inches

f. Rebar Clash (RbClash)

Finds bars that cut into each other, using their real diameters: contact (overlap up to a few mm) vs crossing

INPUTS:

- **Bars A (A)**: Rebar Bars to check (any tree shape)
- **Bars B (B)**: Optional second set; empty = check A against itself
- **Contact (Ct)**: Overlap still counted as contact, in mm

OUTPUTS:

- **Points (P)**: Midpoint between the two bars at each clash
- **Clashes (Cl)**: "MarkA x MarkB: crossing|contact, overlap N mm" per clash
- **Crossing (X)**: True for a real crossing, false for contact - one per clash
- **Summary (S)**: Clash count per pair of Marks

g. Rebar Nudge (RbNudge)

Shifts clashing bars (e.g. stirrups) along the deck, a few mm at a time, until they clear the obstacles - or to the least-intrusion position

INPUTS:

- **Bars (Bars)**: Bars allowed to move (planar, e.g. stirrups) - any tree shape, kept in the output
- **Obstacles (Obs)**: Bars that stay where they are (e.g. longitudinal bars)
- **Max Shift (Max)**: Largest shift either way, in mm
- **Step (St)**: Shift increment, in mm
- **Contact (Ct)**: Overlap still accepted as contact, in mm

OUTPUTS:

- **Bars (Bars)**: Bars at their final position, same tree as the input
- **Shift (Sh)**: Shift applied to each bar, in mm (+ along its plane normal)
- **Overlap (Ov)**: Remaining overlap of each bar, in mm (0 = clear)
- **Summary (S)**: How many bars stayed, moved, or stay with an overlap

h. Bake Geometry (Bake)

Bakes geometry into the Rhino document under a given (nested) layer, with optional colors and grouping

INPUTS:

- **Geometry (G)**: Geometry to bake
- **Layer (Layer)**: Target layer path, e.g. "Bridge::Rebar::Stirrups" (created if missing)
- **Layer Color (LCol)**: Color to (re)apply to the target layer
- **Object Color (GCol)**: Per-object color override; leave unset to use the layer's color
- **Group (Group)**: Group the baked objects together
- **Bake (Bake)**: Set True to actually bake

OUTPUTS:

- **IDs (IDs)**: Object IDs of the baked geometry
- **Geometry (G)**: Passthrough of the input geometry, for wiring straight into Custom Preview
- **Object Color (GCol)**: Passthrough of the input object color, for wiring straight into Custom Preview

i. Splice Bars (RbSplBar)

Laps Rebar Bars longer than the commercial length into pieces (F@/R@ pattern or P@ fixed positions), keeping Mark and diameter

INPUTS:

- **Bars (Bars)**: Rebar Bars, any tree shape
- **Tokens (Tok)**: Piece pattern: e.g. "R@9.0" (fill with 9 m pieces), "F@6","R@9", or fixed lap centers "P@0.5"
- **Lap (Lap)**: Lap length, in model units
- **Symmetric (Sym)**: Mirror all but the last token at both ends (F@/R@ only)
- **Max Length (MaxL)**: Only bars longer than this are spliced, in model units
- **Marks (Mk)**: Optional: only these Marks are spliced

OUTPUTS:

- **Bars (Bars)**: Bars after splicing (one item per piece), same tree as the input
- **Spliced (N)**: How many bars were spliced

8. Revit · Native Revit rebar (inside Rhino.Inside.Revit only)

a. Send Curves To Revit (Crv2Rvt)

Creates native Revit Rebar elements from plain curves - polylines, lines and arcs; non-planar bars become free-form Rebar (manual rebar, or RbGroup's Geometry/Marks), with an optional Mark per branch; pinned and tracked like Rhino.Inside's own element components

INPUTS:

- **Curves (C)**: Curves to turn into Revit Rebar: polylines, lines and arcs (arcs go to Revit as true arcs; a non-planar bar, e.g. an arc with a leg out of its plane, becomes free-form Rebar) - any tree shape; each branch is one group sharing a Mark/Bar Type/Material. Disconnecting it removes the bars this component sent
- **Marks (Mk)**: optional - Revit Mark, one per branch of Curves (last one repeats; a single value marks everything)
- **Host (H)**: Revit element the rebar will be hosted on (a DirectShape works fine)
- **Bar Type (BT)**: Revit RebarBarType (e.g. #4 for a 1/2" bar), one per branch of Curves (last one repeats; a single one applies to every curve)
- **Material (Mat)**: optional - Revit material, one per branch (last one repeats). Revit keeps material per BAR TYPE, so this recolours every bar of that type in the project. Leave empty to keep the bar type's current material
- **Run (Run)**: optional - False leaves Revit untouched (no delete, no create); empty or True sends
- **Elements (E)**: Created Revit Rebar elements, same branches as Curves (null where a curve was skipped)

OUTPUTS:

- **(none)**

b. Send Rebar To Revit (Rebar2Rvt)

Creates native Revit Rebar elements from Rebar Bars (e.g. RbGroup's Bars output), writing each bar's Mark; pinned and tracked like Rhino.Inside's own element components

INPUTS:

- **Rebar Bars (Bars)**: GH_Bridge reinforcement bars to create in Revit - any tree shape (e.g. RbGroup's Bars, one branch per Mark); Elements output keeps the same branches. Catalog hooks (H90, H135...) and RbStir's closing hooks go to Revit as Rebar Hook Types (library shapes such as 02, 08, T1; a matching hook type is created when the project has none); other hooks go as drawn. Disconnecting it removes the bars this component sent
- **Host (H)**: Revit element the rebar will be hosted on - Rebar.CreateFromCurves only needs a generic Element (a DirectShape works fine), not a true HostObject
- **Bar Type (BT)**: Revit RebarBarType (e.g. #4 for a 1/2" bar), one per branch of Bars (last one repeats; a single one applies to every bar)
- **Material (Mat)**: optional - Revit material, one per branch (last one repeats). Revit keeps material per BAR TYPE, so this recolours every bar of that type in the project. Leave empty to keep the bar type's current material
- **Run (Run)**: optional - False leaves Revit untouched (no delete, no create); empty or True sends
- **Elements (E)**: Created Revit Rebar elements, same branches as Bars (null where a bar was skipped)

OUTPUTS:

- **(none)**

9. Curves · Curve tools for manual rebar

a. Curve Angle At End (CrvAng)

Tangent angle (degrees, CCW from the plane's X axis) at a curve's lower end in the plane

INPUTS:

- **Curve (C)**: Curve to measure
- **Plane (Pl)**: Reference plane

OUTPUTS:

- **Angle (Ang)**: Tangent angle in degrees, 0-360
- **Plane (Pl)**: Reference plane used

b. Discretize Curve (DiscCrv)

Discretizes a curve by segment type: straight segments keep just their endpoints, arcs are subdivided by distance

INPUTS:

- **Curves (C)**: Curves to discretize
- **Distance (D)**: Subdivision distance for arc segments
- **Vertex Mode (V)**: Add points just before/after every straight-segment vertex instead of only at it

OUTPUTS:

- **Points (P)**: Discretization points, one branch per curve
- **Polylines (Pl)**: Polyline through the discretization points, one branch per curve

c. Divide Curve By Count (DivCrvN)

Divides curves into N equal segments; outputs one branch per point index across all curves

INPUTS:

- **Curves (C)**: Curves to divide
- **Divisions (N)**: Number of divisions
- **Start Offset On (OffS)**: Enable a start offset
- **Start Offset (dS)**: Start offset distance
- **End Offset On (OffE)**: Enable an end offset
- **End Offset (dE)**: End offset distance

OUTPUTS:

- **Points (P)**: Division points, one branch per point index across all curves

d. Divide Curve By Length (DivCrv)

Divides curves by a fixed length step; outputs one branch per point index across all curves

INPUTS:

- **Curves (C)**: Curves to divide
- **Distance (D)**: Division step (arc length)
- **Mode (Mode)**: 0 = Beginning, 1 = Middle, 2 = End, 3 = Extreme (leftover distributed accordingly)
- **Start Offset On (OffS)**: Enable a start offset
- **Start Offset (dS)**: Start offset distance
- **End Offset On (OffE)**: Enable an end offset
- **End Offset (dE)**: End offset distance

OUTPUTS:

- **Points (P)**: Division points, one branch per point index across all curves

e. Extend Curve (ExtCrv)

Extends a curve at its start or end by a fixed distance

INPUTS:

- **Curves (C)**: Curves to extend
- **Distance (D)**: Distance to extend
- **End (E)**: False: extend the Start / True: extend the End
- **Plane (Pl)**: Reference plane for straight curves

OUTPUTS:

- **Curves (C)**: Extended curves

f. Extend Curve Both Ends (ExtCrv2)

Extends a curve at both its start and end by the same fixed distance

INPUTS:

- **Curves (C)**: Curves to extend
- **Distance (D)**: Distance to extend at each end
- **Plane (Pl)**: Reference plane for straight curves

OUTPUTS:

- **Curves (C)**: Extended curves

g. Extend Point To Curve (ExtPtCrv)

Casts a ray from each point at a fixed angle and trims it to where it meets a target curve

INPUTS:

- **Points (P)**: Ray origins
- **Target (T)**: Target curve
- **Angle (A)**: Ray angle in degrees, 0 = plane's X axis, CCW
- **Plane (Pl)**: Reference plane

OUTPUTS:

- **Curves (C)**: Line from each point to its hit on the target (null if no hit)

- **Distances (D)**: Distance from each point to its hit (null if no hit)

h. Line From Point (LinePt)

Draws a line from a point at a given in-plane angle and distance

INPUTS:

- **Points (P)**: Start points
- **Angle (A)**: Angle in degrees, 0 = plane's X axis, CCW
- **Distance (D)**: Line length; negative flips the direction 180 degrees
- **Plane (Pl)**: Reference plane

OUTPUTS:

- **Curves (C)**: Lines, one per point

i. Project Curve Between (PrjCrv2)

Extends both ends of a curve, each toward its own target curve

INPUTS:

- **Curve (C)**: Curve to extend
- **Start Target (TS)**: Target(s) for the Start end
- **End Target (TE)**: Target(s) for the End end
- **Plane (Pl)**: Reference plane

OUTPUTS:

- **Curves (C)**: Curves extended toward their Start/End targets

j. Project Curve To Curve (PrjCrv)

Extends a curve along its own tangent until it meets a target curve

INPUTS:

- **Curve (C)**: Curve to extend
- **Target (T)**: Target curve(s) to extend toward - one, or one per Crv1
- **From Start (S)**: False: extend from the End (default) / True: extend from the Start
- **Plane (Pl)**: Reference plane

OUTPUTS:

- **Curves (C)**: Extended curves (null where no hit was found)
- **Plane (Pl)**: Reference plane used

k. Orient Curves To Planes (OrientPl)

Copies template curves from a reference point onto every (filtered) target plane

INPUTS:

- **Curves (C)**: Template curves, defined around Point in world XY

- **Reference (R)**: Reference origin the template curves are built around
- **Planes (Pl)**: Target planes to copy the template onto
- **Side (S)**: 0 = All, 1 = Right half, 2 = Left half
- **Alternate (Alt)**: 0 = All, 1 = Even indices only, 2 = Odd indices only

OUTPUTS:

- **Curves (C)**: Copied curves, one branch per target plane index
- **Points (P)**: Reference point copied onto each target plane

l. Rebuild Curve (RebuildCrv)

Re-spaces a densely-discretized curve to uniform point spacing, keeping both endpoints

INPUTS:

- **Curves (C)**: Densely-discretized curves to rebuild
- **Spacing (S)**: Target spacing between points
- **Interpolated (I)**: False: rebuild as a Polyline / True: rebuild as an interpolated curve

OUTPUTS:

- **Curves (C)**: Rebuilt curves
- **Points (P)**: New points, one branch per curve
- **Count (N)**: Number of points generated, per curve
- **Actual Spacing (AS)**: Actual resulting spacing, per curve

m. Rebuild Curve With Hook (RebuildCrvH)

Re-spaces a densely-discretized curve to uniform spacing, preserving an auto-detected hook segment at one end

INPUTS:

- **Curves (C)**: Densely-discretized curves to rebuild
- **Spacing (S)**: Target spacing for the longitudinal (non-hook) span
- **Hook Length (Hk)**: Known length of the hook's single end segment, used to detect it
- **Tolerance (Tol)**: Length-match tolerance for hook detection
- **Prefer End (E)**: If both ends match the hook length: False prefers Start, True prefers End
- **Interpolated (I)**: False: rebuild the longitudinal span as a Polyline / True: as an interpolated curve (the hook always keeps its original points)

OUTPUTS:

- **Curves (C)**: Rebuilt curves (null where no hook was detected)
- **Points (P)**: New points, one branch per curve
- **Side (Side)**: "S" (hook at Start), "E" (hook at End), or null if not detected
- **Hook Point (HP)**: Vertex where the hook begins
- **Count (N)**: Number of points in the rebuilt curve
- **Actual Spacing (AS)**: Actual resulting spacing of the longitudinal span

n. Splice Curves (SplCrv)

Splits curves into lapped pieces following an F@val/R@val token pattern, sequential or symmetric

INPUTS:

- **Curves (C)**: Curves to splice
- **Tokens (Tok)**: Ordered tokens: "F@val" fixed length, "R@val" fills the rest (at most one, plain numbers are read as F@val). Or fixed lap positions only: "P@val" centers a lap at val (<= 1: fraction of the length, > 1: distance)
- **Lap (Lap)**: Overlap between consecutive pieces
- **Symmetric (Sym)**: False: sequential start-to-end / True: mirror all but the last token at both ends

OUTPUTS:

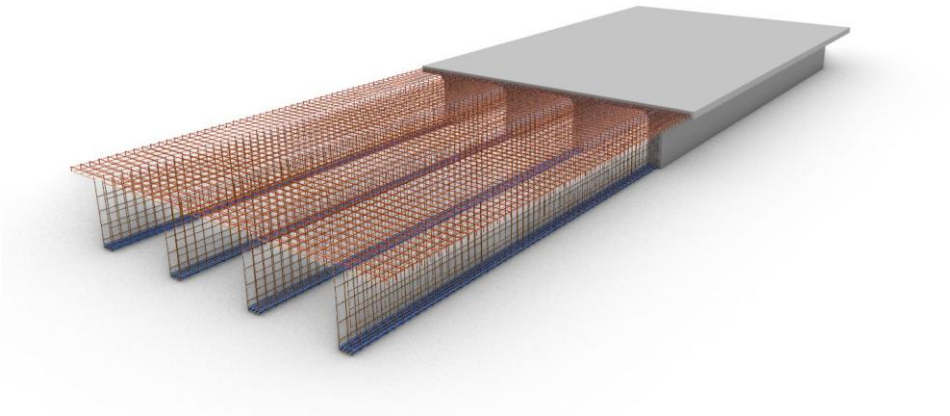
- **Curves (C)**: Resulting spliced pieces, in order
- **Token (Tok)**: "F" fixed, "R" fill piece, "R!" the fill piece adjusted to absorb rounding - same order as Curves
- **Length (L)**: Nominal length of each piece - same order as Curves
- **Check (Chk)**: Per input curve: covered length minus actual curve length (0 = exact fit)
- **Warning (W)**: Per input curve: True if something didn't fit exactly (see Check)

5. EXAMPLES

- **Example_1**: single-cell box girder with transversal and longitudinal rebar and Revit export.
- **Example_2**: seven decks with box, steel and precast girders, assembled by seating sections, with the JSON section library.

Both come in the Examples folder of the .zip. Open Example.3dm before the .gh: the definition references curves from that file. Paths to the JSON files are relative to the example folder.

The complete examples kit, from Example_3 on (a curved 3-span box-girder bridge and a 25 m T-beam bridge, end to end, with take-off), is available at daapcloud.com/gh-bridge.



Example_5 (kit): 25 m T-beam deck with its rebar.

Known limitations (v1.0)

- A 180° hook on a straight bar does not define a plane: it is sent to Revit drawn (free form) instead of as a Revit hook type.
- With Rhino.Inside.Revit, Revit processes the sends when its window is active: if a send does not progress, click once in the Revit view.

License and responsibility

Freeware: free for any use, including professional work. Selling, modifying or decompiling it is not allowed. Full text in LICENSE.txt. Copyright (c) 2026 Andrei Damian Pacheco.

GH_Bridge is a modeling tool. The geometry, rebar and take-off it produces must be reviewed by a qualified engineer, who remains responsible for the design.

Support and updates: daapcloud.com/gh-bridge