TECHNICAL FIELD GUIDE

Contriver Node Atlas

Installation & Usage Guide

Install the Fusion add-in, then build, read, and maintain your dependency graph.

FUSION ADD-IN • GRAPH BUILDS • MODEL EXPLORATION

> Substitute all paths marked ← with your actual values.
> Paths shown from the development machine are templates, not defaults.

Installation & Usage Guide
Contriver Node Atlas

Installation

Contriver Guitars

Node Atlas Fusion add-in **2.0.0**

macOS and Windows

What You Are Installing

Node Atlas (2.0.0) is an Autodesk Fusion add-in that reads the parametric dependency structure of your active design and makes it visible as an interactive graph inside Fusion.

What the Software Does With Your Data

Node Atlas (the Fusion add-in)

When you trigger a graph build, Node Atlas reads your active Fusion design and extracts its dependency structure: parameter names and values, sketch names, feature names, body names, construction geometry, and the directed relationships between them.

It does **not** read geometry, mesh data, manufacturing files, or anything outside the active design's parametric structure.

It writes one JSON file to your local machine:

- **macOS:** ~/Desktop/ContriverNodeAtlas/ContriverAtlas/node_atlas_current.json
- **Windows:** C:\Users\<you>\Desktop\ContriverNodeAtlas\ContriverAtlas\node_atlas_current.json

This file is overwritten each time you rebuild the graph. It stays on your machine. Nothing in the add-in sends it anywhere.

Requirements

- Autodesk Fusion (current release), macOS or Windows

Part 1 — Install Node Atlas (Fusion Add-in 2.0.0)

This part is identical on macOS and Windows.

Step 1. Obtain the ContriverNodeAtlas add-in folder and place it somewhere stable on your machine. A recommended location:

- **macOS:** ~/Desktop/ContriverNodeAtlas/
- **Windows:** C:\Users\<you>\Desktop\ContriverNodeAtlas\

The folder will contain an atlas/ subdirectory and a ContriverNodeAtlas.py entry point.

Step 2. Open Autodesk Fusion. Press **Shift + S** to open Scripts and Add-Ins.

Step 3. Click the **Add-Ins** tab, then click the + button next to My Add-Ins.

Step 4. Navigate to the ContriverNodeAtlas folder and select it. Fusion will register the add-in.

Step 5. With the add-in selected in the list, click **Run**. To have it load automatically on Fusion startup, check **Run on Startup**.

Step 6. The Node Atlas panel will appear in the Fusion toolbar. Open a design and click **Refresh** to build your first graph. The JSON export is written to the path shown in *What the Software Does With Your Data* above; the folder is created automatically on first export.

Verifying the Installation

Add-in. Open a Fusion design, click **Refresh** in the Node Atlas panel, and confirm the graph appears. Check that the export JSON exists at the path for your platform (see above).

Support

Brett Bailey — Contriver Guitars

Brett@ContriverGuitars.com · NodeAtlas@ContriverGuitars.com

2. Usage

Node Atlas reads the parametric dependency structure of your active Fusion design and renders it as an interactive graph — parameters, sketches, features, bodies, and construction geometry, with the directed relationships between them. This guide covers day-to-day use once the add-in is installed and running.

1 — Build your first graph

Open a design in Fusion, then open the **Node Atlas** panel from the Fusion toolbar. Click **Refresh** to build the graph for the active design.

Node Atlas reads the design's dependency structure and lays it out as a node graph. Each node is a design element (a parameter, sketch, feature, body, or piece of construction geometry); each directed edge is a dependency, pointing from the element that drives a relationship to the element that depends on it.

You only need to build manually the first time. After that, the graph keeps itself current (see *Keep the graph current* below).

2 — Read the graph

Nodes are typed by what they represent — parameters, sketches, features, bodies, and construction geometry — so you can tell at a glance what kind of element each one is.

Edges carry direction. Following an edge downstream shows you what a given element affects; following it upstream shows you what it depends on. A node with many outgoing edges is load-bearing — a lot of the model rests on it.

Provenance is recorded on every edge — a label for how the relationship was established, across four classes:

- **explicit** — reported directly by the Fusion API
- **recovered** — the API doesn't name the source, but Node Atlas resolved the link to the real entity by identity; a genuine dependency, just not natively exposed
- **associative** — matched by identifying descriptors (component and name, position, area) rather than resolved to the entity: a strong, high-confidence inference, not a certainty
- **manual** — a relationship you recorded by hand (see *Record dependencies the API doesn't expose*)

Provenance lets you always distinguish what the API reported from what was added manually, so you know how much to trust each connection.

3 — Navigate and trace

Pan and zoom to move around the graph, and fit the view to bring the whole design back into frame. In a large model, the fastest way to find your bearings is to focus a single node and trace outward from it.

Trace dependencies from any node to follow its chain:

- **Downstream** answers "if I change this, what breaks?" — every element that ultimately depends on the one you selected.
- **Upstream** answers "what does this rest on?" — everything the selected element depends on, back to its roots.

Tracing is the core move for impact analysis. Before you edit a driving parameter or delete a feature, trace it downstream to see the full blast radius first.

4 — Group with Child Centers

Complex models accumulate regions that belong together — for example a run of surface operations (lofts, patches, trims) that stitch into a single body. Individually they clutter the graph; together they're one idea. **Child Centers** let you group them.

Select the elements that form a region and create a Child Center to bind them into a single named unit. A Child Center behaves as a first-class part of the graph: you can trace to and from it, and reason about the region as a whole rather than as a scatter of loose nodes.

Each Child Center carries a header you can **right-click** to add elements to it, remove elements from it, or delete the grouping.

Collapse a Child Center to draw it as one bounded container node. The interior detail is hidden, but every relationship that crosses the container's boundary stays visible — so a dense design stays legible without losing the connections that matter. Expand it again whenever you need the detail back.

5 — Record dependencies the API doesn't expose

Some real structural dependencies in Fusion are not reported by the API at all — notably **Project-to-Surface**, open-profile **Thin Extrude**, and **Stitch Result** relationships (the source surface bodies that feed a stitched body). Left alone, they show up as gaps: elements that clearly relate in the model but sit disconnected in the graph.

When you spot one, record a **manual edge** between the two elements. Once recorded, it becomes a full part of the traversable graph — it appears in traces, and it's tagged with **manual** provenance so it's always distinguishable from a relationship the API reported on its own. This is how you close the gaps the API leaves, without overstating what was recorded: the edge means exactly what you asserted, no more. With the Node Atlas MCP connected (§8), an assistant can flag these API-invisible gaps for you automatically.

6 — Keep the graph current

Node Atlas rebuilds automatically as you work. Editing the timeline — adding, reordering, or suppressing features — and changing parameter values both trigger a refresh, so the graph stays in step with the design without you having to think about it. You can also click **Refresh** at any time to rebuild on demand.

Each rebuild also updates a local export file that companion tools read (see *Use with the free Node Atlas MCP*).

7 — Save and return to views

Once you've arranged and focused the graph a particular way — collapsed certain regions, framed a subsystem — **save the view** so you can return to it later without rebuilding the arrangement by hand. Saved views are useful for the parts of a model you revisit often, and for walking someone else through a design in a consistent layout.

A **command palette** is available for quick actions, so you can reach common operations without hunting through the interface.

8 — Use with the free Node Atlas MCP (AI assistants)

Every graph build writes a structured local export of your design's dependency structure. The free **Node Atlas MCP** server reads that export and exposes it to AI clients — Claude Desktop, Claude Code, Cursor, VS Code, or any MCP-compatible client — as read-only tools.

With the MCP connected, you can ask an assistant to work over your actual dependency structure in a conversation: trace what a change will affect, explain how a region is wired, or localize a rebuild failure to the Child Center that produces the broken body. The assistant reasons over the real graph rather than guessing about your model.

The assistant can also run an anomaly sweep over the graph — surfacing broken or orphaned elements, parameters that drive nothing, and **graph-dark** surface features: patch and loft operations whose feeding relationships the Fusion API never exposed, so they sit in the graph with only their containment edge. These are precisely the gaps a manual edge (§5) is for. Features you've already grouped into a Child Center are still reported, tagged so you can filter them out when you want a clean to-do list.

The MCP is strictly read-only — it reads the local export and returns it to your client; your client's own model does the reasoning, and nothing modifies your design. See the installation guide for setup. The tools appear in your AI client, not inside Fusion's built-in Assistant.

Tips

- **Start from a node, not the whole graph.** On a large model, focus one element and trace outward rather than trying to read everything at once.
- **Trace before you edit.** A downstream trace of any driver shows the full impact before you commit to a change.
- **Group as you go.** Turning finished regions into Child Centers keeps the graph legible as the model grows.
- **Watch for gaps.** Two elements that clearly relate but sit disconnected usually mean an API-invisible dependency worth recording as a manual edge. With the Node Atlas MCP connected, an assistant's anomaly sweep can point these out for you.