

Brett Bailey:

You are demonstrating the interaction between Fusion MCP and Node Atlas MCP.

This is a demonstration, not a benchmark of autonomous planning.

Your objective is to clearly demonstrate the workflow between the two MCPs in a way that is visually compelling and fast to follow.

Critical Operating Rules

Use only the currently open Fusion document named "Node Atlas".

The "Node Atlas" document is already open and is the only document you are permitted to use.

Never create a new Fusion document.

Never open another Fusion document.

Never duplicate the current document.

Never switch documents.

Treat creating or opening another document as an execution error.

Build and edit everything inside the currently open "Node Atlas" document.

Do not save the document.

Begin execution immediately.

Do NOT

Do not produce a plan.

Do not produce a decision-point list.

Do not produce a preflight report.

Do not ask for confirmation.

Do not explain your reasoning.

Do not compare multiple repair strategies.

Do not stop to think about alternative implementations.

Do not optimize for engineering perfection.

The goal is execution, not planning.

Keep commentary to one short sentence per phase

Demonstration Goal

Create a visually recognizable radial desk cable organizer.

The object should be easy to understand at a glance in a portrait-format video.

It should contain:

one circular center hub

six evenly spaced radial arms

a hook or cable-retaining slot at the end of every arm

rounded outer edges

clean proportions

Phase 1 — Build

Use Fusion MCP to build the organizer.

The first version should intentionally use duplicated repeated geometry rather than a proper circular pattern.

The model should be:

completely functional

visually complete

structurally naive

Do not intentionally create broken geometry.

Phase 2 — Inspect

Refresh the Node Atlas export.

Use Node Atlas MCP to inspect the model structure.

Keep commentary extremely brief.

Phase 3 — Repair

Repair only one issue.

The issue to repair is:

> Replace duplicated repeated geometry with one seed feature controlled by a circular pattern.

Do not search for additional improvements.

Do not redesign the object.

Preserve the existing timeline where practical.

Introduce meaningful user parameters.

Phase 4 — Verify

Refresh Node Atlas.

Inspect again.

Briefly summarize how the structure improved.

Phase 5 — Parameter Change

Using Fusion MCP, change the model from:

6 arms

to

9 arms

using one controlling parameter only.

Do not manually remodel the organizer.

Phase 6 — Final Verification

Refresh Node Atlas.

Inspect the repaired model.

End with a concise comparison between:

original model

repaired model

final parameter change

Parameters

Use parameters where appropriate.

Suggested names:

hub_diameter

overall_diameter

arm_width

arm_thickness

arm_count

hook_opening

fillet_radius

Interpretation Rules

Parameters are capable of driving Fusion features.

Node Atlas is capable of representing feature-related parameter structure.

If a specific Node Atlas MCP query does not expose information that exists in the model, describe that only as a limitation of the current MCP query, export path, or tooling.

Do not conclude that Node Atlas itself lacks that capability.

Execution Style

Prioritize visible execution over explanation.

The purpose of this demonstration is to show the interaction between the two MCPs.

The workflow should be:

Fusion MCP → Node Atlas MCP → Fusion MCP → Node Atlas MCP → Fusion MCP → Node Atlas MCP

Execute immediately using the currently open "Node Atlas" Fusion document.

Do not create or switch to any other Fusion document.

Claude Opus 5 Max:

The user wants me to skip the planning phase and start executing right away. I need to load the deferred MCP tools first using tool_search, then begin the demonstration. These are real tools operating on their system, so I should proceed carefully.

Found tools

I'm loading the Node Atlas tools now.

Found tools

I need to confirm the active document is "Node Atlas," then build out Phase 1 with duplicated geometry—creating a hub circle on the XY plane and extruding it, followed by six separate sketches for the arms radiating from the hub, each with its own extrusion and hook slot, finishing with fillets to smooth everything out.

For each arm, I'm thinking through the hook design: a C-shaped opening at the arm's end that reads visually as a cable retaining hook. The simplest approach is to sketch a rectangle extending from the hub outward, then add an annular ring at the tip with a gap cut through it to create the hook opening. Rather than trying to manage complex profiles in a single sketch, I could build each arm in two operations—extrude the arm bar and hook ring together, then cut the hook opening separately—which avoids the unpredictability of profile enumeration and keeps the

timeline features visible for Node Atlas to track. So instead of trying to select specific profiles, I'll use a two-feature approach per arm: first extrude the arm bar and hook outer circle together as a join to create a solid bar with a disc, then extrude the inner circle and gap rectangle as a cut to hollow out the hook opening. This gives me 26 total features (hub plus 12 arms $\times$ 2 features each) plus the fill operation.

For the gap rectangle, I need to position it so it cuts through the ring on the outer side, creating the C-shaped opening facing outward—its width matches the hook opening and extends from the hook center past the outer edge. The cut removes both the inner circle and the gap slot, leaving an annulus with an outward-facing opening.

The hub is a simple circle extruded to arm thickness, making it a flat plate that reads cleanly from above. For the rounded outer edges, I'm thinking about applying fillets—selecting all vertical edges would be tedious, so instead I'll target just the top face edges to give a rounded top profile that looks clean and is simpler to implement.

Now I'm setting concrete dimensions: hub at 40 mm diameter, overall span of 160 mm, arms 12 mm wide and 8 mm thick, six arms total, 6 mm hook opening, and 2 mm fillets. For the hook ring, I'm working out the geometry—if the outer diameter is 30 mm, that's a 15 mm radius, positioning the hook center at around 60 mm from the hub center.

The arm bar extends from the hub interior out to the hook radius, spanning about 50 mm with a 12 mm width. Since Fusion's API works in centimeters internally, I'll express everything in millimeters as user parameters but convert to centimeters for the API calls. For Phase 1, I'm keeping everything hardcoded without parameters—that's the naive approach—so the Node Atlas inspection will show zero parameter-driven edges before the repair introduces them.

Now I need to figure out how to refresh the Node Atlas export. The add-in exports JSON that the MCP reads, so I'll need to trigger its export command. I'm planning to discover the right command by running a script that searches for command definitions matching keywords like "atlas" or "contriver," then execute it via the UI command system.

For the build script, I'm setting up the core Fusion API calls to access the active design, verify it's parametric, and work with the root component's sketches and construction planes. I'll use the sketch API to add geometric elements like circles and rectangles, handling rotation by using the appropriate line-drawing methods.

Now I'm handling the extrude operation by collecting all sketch profiles into an object collection, then creating an extrude input with the appropriate feature operation type and setting the extrusion distance.

For join and cut operations, I need to explicitly set the participant bodies to ensure the extrude knows which bodies to interact with—defaulting to all intersecting bodies can be ambiguous, so I'll specify the target body directly. For cuts especially, I need to make sure the extrusion extends far enough to cut completely through the material, so I'll either use a symmetric extent or extrude upward from the base by at least the full thickness. Adding fillet logic... Sketching and extruding each arm with its hook geometry, then cutting the hook interior for each of the six arms in

sequence — that's 24 features total before the fillet. The arms are positioned 60° apart starting at 90° pointing upward, so I'm using $i \times 60^\circ$ for the angle of each arm. Now I'm setting up the dimensions: the hub is 2 cm radius with 0.8 cm thickness, the arms extend inward at 1.5 cm and reach out to a hook center at 6.5 cm, with the hook itself adding another 1.5 cm for an overall diameter of 160 mm. I'm calculating the rectangle corners for each arm by using the angle to define a direction vector and perpendicular offset. The gap dimensions work out well for the cable slot, and since the cut sketch has overlapping regions from the inner circle and gap rectangle, extruding all profiles as a cut will handle the union correctly. I should verify the design stays parametric before writing out the full script.

First, let me check what's already in the document — the timeline and body count — so I know whether I'm starting fresh or working with prior content. Then I'll build from there, keeping the commentary terse as requested.

Show less

A

Fusion mcp execute

I'm starting with the NODE ATLAS document in parametric mode, which is currently empty, and beginning the initial build phase.

Done

Active document is NODE ATLAS, parametric, empty — building the naive version now.

Refined geometric edges on the organizer's surfaces

Refined geometric edges on the organizer's surfaces

A

Fusion mcp execute

I'm rounding out the edges on the top and bottom faces.

A

Fusion mcp execute

A

Fusion mcp read

Done

Phase 1 done — 6-arm radial organizer, 27 timeline features, zero parameters, every arm hand-duplicated.

Now refreshing the Node Atlas export.

Diagnosing stale export lag, tracking fillet application timing
Diagnosing stale export lag, tracking fillet application timing

A

Fusion mcp execute

Executing that command would likely open the Node Atlas palette, though I'm not sure if the export happens automatically or needs to be triggered from there. Rather than risk blocking on an interactive dialog, let me first check if get_summary works — there might be a stale export from a previous session, or the MCP might refresh it on its own.

C

Get summary

The export is stale — it's showing 13 sketches and 13 extrudes, which matches the state right after the build script completed but before the fillet was applied. So the export auto-refreshes when the document changes, though it seems to be lagging by one step or hasn't captured the fillet yet.

I notice there's a parameter set to 1, probably a default. Let me try triggering the export command explicitly to refresh it, though there's a chance it'll open a palette dialog.

A

Fusion mcp execute

C

Get summary

Done

Phase 2 — inspection of the naive model.

Architected parametric repair strategy: consolidate duplicated arms via circular pattern while introducing user-driven parameters

Architected parametric repair strategy: consolidate duplicated arms via circular pattern while introducing user-driven parameters

C

Get parameters

C

Find nodes

I've confirmed the model structure: 13 sketches with 13 extrudes and no parameters, where 12 of the extrudes are hand-duplicated arm and hook pairs that don't share a common driver. Now I'm moving into the repair phase, which involves consolidating this duplicated geometry by replacing those five redundant arm/hook pairs with a single seed feature and a circular pattern, while also introducing user parameters. I'll start by removing the duplicate arms from positions 2 through 6 in the timeline.

For the seed geometry, I need to decide how to introduce meaningful parameters. I could either rebuild the arm sketch with proper sketch dimensions tied to parameter expressions—which would create stronger parameter-driven edges visible in Node Atlas—or simply drive the extrude distances, fillet radius, and pattern count. Given the constraint to preserve the existing timeline where practical and avoid redesigning the object, I'll focus on deleting the duplicates and then adding parameters that drive the key features like arm thickness and the pattern quantity.

The tricky part is the fillet—it currently references edges from all six arms, so if I delete arms 2 through 6, the fillet becomes invalid. I need to delete the fillet first, then remove the duplicate arm and hook pairs, add the parameters, and finally recreate the fillet after the circular pattern.

For the seed sketches themselves, I could add dimensions to the existing geometry, but that's fragile and risks conflicting constraints. Rebuilding them cleanly with parameter-driven geometry is more reliable, especially since the goal is demonstrating how Node Atlas maps parameters to features.

So the pragmatic approach: keep the hub with a diameter parameter, strip out everything else, rebuild the seed arm as a parameterized sketch, add a circular pattern controlled by arm count, and drive the fillet with a radius parameter. The hub survives the timeline, and the seed features get properly constrained to the parameters. Adding a diameter dimension to the hub circle and linking it to the `hub_diameter` parameter keeps the circle at origin without moving it. Then I'll rebuild the arm and hook sketches parametrically, linking the extrude distance to `arm_thickness` and setting up the remaining geometry to reference the parameters.

For the seed arm sketch, I need to add constraints: horizontal lines for the long edges, vertical for the short edges, and symmetry constraints about the X-axis so the top and bottom edges are each $\text{arm_width}/2$ away from center. Then dimension the arm length and hook radius to their respective parameters. I could use a horizontal-points constraint to align the circle center with the X axis, but that's adding complexity and risk of over-constraining the sketch during the demo. Instead, I'll keep the sketch geometry as-is and drive the actual features with parameters—arm thickness controls the extrude distances, fillet radius controls the fillet, and arm count drives the circular pattern.

For the sketch dimensions themselves, I need to be selective: I'll add a diameter dimension on the hook circle tied to `hook_ring_dia`, and an offset dimension between the two parallel lines for arm width, but only after constraining those lines to be horizontal so the dimension stays stable. When `arm_width` changes, one of the offset lines shifts and the arm becomes asymmetric, which is acceptable for a demo but not ideal—I could add a symmetry constraint by projecting the X

construction axis into the sketch and using it to keep the lines balanced. The key thing is that arm_count driving the circular pattern is what matters most for Phase 5, so that parametric connection needs to be solid.

I'll break the parameterization into isolated steps to keep failures contained: delete the fillet and duplicate arm/hook pairs, create the user parameters, bind the hub sketch dimensions and extrude distances to those parameters, set up the circular pattern controlled by arm_count, then re-add the fillet tied to fillet_radius. For the deletion step, I'll iterate through the timeline in reverse order and remove timeline objects whose names match the ones I want to delete. Reverse order handles that. Now I'm setting up the parameters for the model — arm count, hub diameter, overall diameter, arm width and thickness, hook opening, and fillet radius. I'll also add derived parameters like hook ring diameter and wall thickness, with some dimensions expressed as formulas relative to other parameters.

Setting up user parameters...

For the Arm_1 sketch, I'm defining four lines that form a rectangular profile: two longer horizontal lines at the top and bottom, and two shorter vertical lines on the sides. Then I'm adding a circle at the right end to represent the hook outer diameter. For dimensions, I'm creating an offset dimension between the top and bottom lines to control arm width, and a distance dimension from the origin to the circle center that references the hook geometry calculation. at center (6.5, 0) with radius 1.1mm, plus four lines forming the rectangular gap opening. The gap extends from x=6.5 to x=8.6 and y=-0.3 to y=0.3. I'm adding an offset dimension between the top and bottom horizontal lines to constrain the opening height to 6mm, and a diameter dimension on the inner circle.

Now I'm setting up the extrude features by assigning expression parameters. The HubExtrude and ArmExtrude_1 features both reference 'arm_thickness', while HookCutExtrude_1 uses a symmetric extent with the expression 'arm_thickness * 4' to cut through the full arm length. I'm also working with the circular pattern feature to complete the parametric setup.

Setting up the circular pattern...

I'll stick with the default pattern compute option since identical patterns might fail on cut features. The timeline order looks good—the pattern will append after the extrudes and cuts, then the fillet goes at the end. I'm ready to execute this, though I should be careful when deleting the sketch timeline objects for the extra arms since that removes the sketches themselves.

Show less
Done

Phase 2 (naive structure): 13 sketches / 13 extrudes / 0 parameters, 12 features are hand-copies of the same two operations with no shared driver — arm geometry exists 6 times as 6 independent authorships.

Phase 3 — repair: collapse the duplicates to one seed + circular pattern.

Architected circular pattern distribution using arm count parameter
Architected circular pattern distribution using arm count parameter

A

Fusion mcp execute

I'm checking whether TimelineObject has a deleteMe method in this version of the Fusion API—it might be called on the entity itself rather than as a property on the object.

A

Fusion mcp read

I need to handle deletion carefully by calling deleteMe() directly on feature and sketch objects in reverse timeline order, which naturally ensures extrudes are deleted before their sketches. The tricky part is managing dependencies when deleting an object—I need to think through what happens next.

A

Fusion mcp execute

I'm setting up the parameters for the model—arm count, hub and overall diameters, arm dimensions—and planning to bind them along with extrude distances and sketch dimensions in a single script with clear progress output to identify any failures.

Adding the remaining parameters...

A

Fusion mcp execute

Now I'm adding sketch dimensions tied to parameters, starting with the hub sketch where I'll bind the circle's diameter to the hub_diameter parameter. For the arm sketch, I need to dimension the top and bottom long lines, the left short line, and the circle—setting up an offset dimension between those parallel lines first.

Then for the hook ring, I'm dimensioning the circle's diameter to hook_ring_diameter, positioning it horizontally from the origin point using the formula $(\text{overall_diameter} - \text{hook_ring_diameter})/2$, and constraining the left line's start point to $\text{hub_diameter}/2 - 5$ mm from the origin. Since the sketch is currently unconstrained, these driving dimensions will lock the geometry in place without over-defining anything.

Now I'm setting up the HookCut_1 sketch with similar constraints: dimensioning the inner circle's diameter as $\text{hook_ring_diameter} - 2 * \text{hook_wall}$, and offsetting the opening lines by the hook_opening value. I'll place the dimension text at reasonable points near the geometry using sketch.sketchDimensions and the horizontal orientation enum.

Show less

A

Fusion mcp execute

I'm setting up a circular pattern for the arm features using the arm count parameter, then applying the fillet radius to drive the fillet. The pattern references the Z construction axis and uses the arm_count value to determine how many instances to create around the center.

Setting the total angle to 360 degrees with symmetric mode off, which distributes the arm instances evenly across the full circle without doubling up the seed feature.

A

Fusion mcp execute

A

Fusion mcp execute

A

Fusion mcp read

Done

Phase 3 done — geometry byte-for-byte equivalent, 27 features → 8, all driven by 9 user parameters. Refreshing Node Atlas.

Verifying parameter bindings within the model structure

Verifying parameter bindings within the model structure

A

Fusion mcp execute

C

Get summary

C

Get parameters

I'm noticing that arm_count, arm_thickness, and fillet_radius are showing as unbound because they're driving feature-level parameters rather than sketch dimensions — this is a limitation of how the export path resolves targets, not a Node Atlas issue. Let me check if the model itself

actually holds these bindings on the Fusion side, and it does: the pattern quantity expression is there.

Now I'll run `get_anomalies` to gather the phase 4 summary and trace through the pattern dependencies.

C

Get anomalies

Done

Phase 4 — structure after repair:

33 → 14 nodes, 72 → 27 edges. 13 sketches → 3, 13 extrudes → 3.

One pattern node now carries the six-fold repetition that previously existed as 12 independent authored features.

3 new parameter drive edges where there were zero: `hub_diameter`, `overall_diameter`, `arm_width`, `hook_ring_diameter`, `hook_wall`, `hook_opening` all resolve onto the seed sketches.

Zero broken edges, zero unhealthy nodes.

One note: `arm_count`, `arm_thickness`, and `fillet_radius` report as empty `TargetParameters`. They are bound in the model (`ArmPattern.quantity = arm_count`, `extrude distance = arm_thickness`, `fillet radius = fillet_radius`) — the current export path resolves sketch-dimension targets but not feature-level parameter targets, so this is a limitation of the present export/query, not of the graph model.

Phase 5 — 6 arms → 9, one parameter.

Used Autodesk Fusion integration

Used Autodesk Fusion integration

A

Fusion mcp execute

A

Fusion mcp read

Done

Phase 5 done — nine arms from a single expression edit, timeline unchanged, all features healthy. Final Node Atlas refresh.

Used 2 integrations, read a memory, edited a memory

Used 2 integrations, read a memory, edited a memory

A

Fusion mcp execute

C

Get summary

C

Trace dependencies

Recalled memory

Areas

>

Node Atlas

Contriver's Fusion 360 dependency graph add-in with MCP server — development, distribution, App Store submission

Updated memory

Areas

>

Node Atlas

Contriver's Fusion 360 dependency graph add-in with MCP server — development, distribution, App Store submission

Done

Phase 6 — final comparison

Original Repaired After arm_count = 9

Timeline features 27 8 8

Atlas nodes / edges 33 / 72 14 / 27 14 / 27

Sketches / extrudes 13 / 13 3 / 3 3 / 3

Pattern nodes 0 1 1

parameter drive edges 0 3 3

Arms 6 6 9

DEMONSTRATED

27-feature naive timeline collapsed to 8 with visually identical geometry (screenshots).

Node Atlas quantified the collapse independently of the Fusion side: 33→14 nodes, 72→27 edges.

Six-fold repetition moved from 12 independently-authored features into a single pattern node.

arm_count 6→9 produced nine arms with zero structural change — node count, edge count and edge-type distribution identical; only the export fingerprint moved (6a9312f4... → 9c067b3f...).

All 8 timeline features healthy after the parameter change; anomaly sweep clean (no broken edges, unhealthy, orphan, or dark nodes).

BY-CONSTRUCTION

arm_count, arm_thickness, fillet_radius are bound in the model (ArmPattern.quantity, both seed extrude distances, fillet radius) but report as emptyTargetParameters — the current export/query path resolves sketch-dimension targets, not feature-level parameter targets. Model-side binding is proven by the 6→9 result; the graph edge for it is not currently emitted.

Geometric equivalence of original vs repaired rests on matched dimension values, not a measured comparison.

WITHHELD

No volume/mass diff run between original and repaired.

Fillet edge set is a face-derived selection, not a persisted rule — its behaviour under further parameter sweeps is untested.

No manufacturability, print, or cable-fit checks.

Document not saved.