Atlas Meridian Project Handoff

Current concept and Meridian conversation record

Version 0.9 | Handoff through 4 October 2026 05:51 UTC

Atlas Meridian is a planned 2D canvas application for creating, exploring and sharing vast networks of information. Objects placed by the user on a canvas are called nodes. At its simplest, a node represents a piece of media, such as text, an image, a video or a document. Nodes can be linked, nested and arranged in different ways so the user can express how information is connected and how a system is organized.

A canvas can hold both an overview and the detail behind it. A node may contain a collection of other nodes, which the user can reveal or hide by expanding or collapsing their parent. Moving through these layers and zooming around the canvas allows the user to explore a large body of information without having to present every part at once. Research, stories and other complex subjects can be organized into connected spaces that are easier to follow and understand.

Meridian is also intended to support the creation of functional, live systems. Specialized connections and configurable operations allow arrangements of nodes to express electrical circuits, generate and combine waveforms, or evaluate mathematical expressions. The user can supply inputs, control the system and observe its output directly. Explanatory text, images and documents can sit alongside these working systems, making the canvas a place to understand an idea as well as interact with it.

Canvases themselves are represented by nodes, allowing the user to browse a connected collection of saved workspaces and open the one they want to explore. A workspace can also lead into other canvases, while collections of nodes can be saved and reused as building blocks for new work. This lets individual pieces of information and smaller systems contribute to larger projects.

Users create and alter canvases in an editor, then make finished canvases available for interactive viewing. The editor is intended to be available online and downloadable for local use, with projects stored locally when desired. Users can selectively publish their work, embed it in webpages, share reusable material and collaborate in shared workspaces. Downloaded canvases and selections can become starting points for further work.

How to use this handoff

The component sections that follow explain the confirmed behavior and open choices in this planned product. [U] marks user decisions; [P] marks proposals, including acceptance examples. N01–N64 identify functional requirements, D01–D22 identify open decisions, and D1–D4 and M1–M10 identify source references. Part Two preserves the Meridian conversation verbatim, including superseded ideas as history. The current specification governs when earlier wording conflicts.

1 Nodes media and universal identity

N01 Treat the canvas as a root node

Basis: [U9, U30] canvas root and special canvas nodes

A canvas is itself a node and acts as the root for its canvas space. It has a title and may have an associated image. U30 adds a special canvas-node representation for browsing and opening canvases; the workspace content belongs to the opened canvas. The exact identity relationship between that root object and its canvas-node appearances remains D18.

Acceptance [P]: Identify a canvas root and its title/image, browse its canvas-node representation and open the workspace. Do not assume separate or shared GUIDs for those appearances before D18 is resolved.

N02 Represent media as nodes

Basis: [U9] minimum node definition; [U1, U2] media scope

At minimum, a node is one piece of media placed on the canvas. Text, images, video, SVG/graphics and complete documents are part of the stated media scope. A node can also be a parent or container rather than being restricted to a passive media card.

Acceptance [P]: Place each supported media item as a node and use its spatial and relationship behavior without an intermediate catalog-entry model.

N03 Identify the node object universally

Basis: [U9, U14, U32, U34] node identity and selected external packaging

Every node receives a GUID that identifies the node object across Meridian, rather than identifying its media contents. Any node may also have an alias, such as a title or handle. The node preserves and delivers a model of its content for references. A GUID alone does not enforce the separate no-writeback requirement. Cross-canvas identity within an existing environment does not imply portable global resolution after download; external packaging follows the downloader’s choice under N62–N63. Identity handling for whole imported canvases remains open; fresh node GUIDs are confirmed for modules and snippets. A future public-hosted universal reference ID is only a possibility.

Acceptance [P]: Resolve a GUID to the intended original node object, even when another node has identical media. References expose the model without creating a mutable original.

N07 Use aliases to open appropriate viewers

Basis: [U9] aliases and media viewers

Certain media types, including PDF and text documents, may appear on the canvas through an alias: a user-assigned text title acting as a hyperlink. Activating that alias launches the appropriate viewer or loader in a separate pane. Any node may have an alias, but an alias is not mandatory for every node.

Acceptance [P]: Activate a document alias and open the actual media in the appropriate separate viewer/loader pane. Other media can retain their suitable canvas representation.

Complete documents

Complete PDF, DOCX and TXT documents are explicitly requested. An alias or summary must still lead to the actual document. The separate viewer/loader pane is confirmed for suitable media; DOCX fidelity, editing and the exact controls remain open. Images, video, text, SVG and other graphics remain in scope, without assuming a full editor for every type. [U1, U2, U9; D07]

2 Canvas repository and workspace entry

N43 Browse saved canvases in a canvas space layer

Basis: [U30] canvas repository navigation

Provide a canvas space layer that acts as the repository where users browse saved canvases and choose which to open. This layer is itself a canvas comprised of special canvas nodes. Opening a selected canvas launches its primary canvas workspace, where users spend most of their time.

Acceptance [P]: Browse several saved canvases as canvas nodes, choose one and open its content workspace. Keep the repository navigation layer distinguishable from the workspace being edited or viewed.

N44 Give canvas nodes a title image and navigation behavior

Basis: [U30] special canvas node properties

Canvas nodes are a special node type with a title and an optional descriptive image. They can collapse or expand to hide or reveal child canvas nodes, and can launch a new window containing their canvas contents. Canvases can contain subcanvases and can be linked. Expansion within the repository hierarchy and opening a content workspace are distinct experiences.

Acceptance [P]: Create a titled canvas node with or without an image. Expand it to reveal child canvas nodes, collapse it to hide those children, then open its canvas contents in a new window. No mandatory image or general title-and-thumbnail rule is imposed on other nodes.

N45 Keep repository layer links visual only

Basis: [U30] noncomputational canvas layer

At the canvas repository layer, links only indicate relationships between canvases. They cannot perform operations, carry electric current or implement logic-gate behavior. Canvas nodes at this layer offer the stated title/image, hierarchy and workspace-launch behavior without distinct computational functions.

Acceptance [P]: Link two canvas nodes to communicate a relationship. The repository connection has no reactive value, joint operation or signal execution. Opening an ordinary workspace still allows its parent-scoped working systems under the existing requirements.

3 Canvas nodes inside ordinary workspaces

N46 Place canvas nodes beside ordinary media and systems

Basis: [U30] embedded canvas nodes

Canvas nodes can also be placed inside a normal canvas workspace alongside other node types. They retain their role as special canvas nodes that can expose child canvases or open a window containing canvas contents. This lets a research workspace lead into another focused workspace without flattening its contents into media.

Acceptance [P]: Place a canvas node alongside text, pictures, documents and video. Activate it to open its associated canvas workspace in a new window while preserving the distinction between the launcher and the content workspace.

Aphrodite example

A node containing the text “Aphrodite” expands to reveal several child nodes: a picture, several documents, a video and a canvas node. Opening that canvas node launches a workspace that plots a story about Aphrodite using a collection of relational nodes. The media remain nodes in the current workspace; the special canvas node provides entry into the story workspace. [U30]

Three experiences to keep distinct

The repository canvas is a navigable collection of saved canvases. Its links are visual relationships only. Expanding a canvas node there reveals its child canvas nodes. Opening it launches the main content workspace in a new window. [U30]

An ordinary workspace can contain heterogeneous media and functional systems under the established parent-scoped rules. The repository restriction does not disable operations throughout all workspace roots. A canvas node embedded in that ordinary workspace remains a canvas launcher; computational connection behavior for such embedded canvas nodes is not specified. [U10, U27, U30; D18]

The canvas object is still its workspace root. “Canvas node” identifies the special browsing/launch representation in this experience; it does not settle whether repeated appearances share a GUID, receive their own GUIDs or use the existing reference mechanism. No new cycle, permission or ownership rule is assumed. [U9, U14, U30; D18]

4 Hierarchy collapse and child layouts

N04 Contain child nodes within parents

Basis: [U9] subnodes

A node can contain other nodes as its children or subnodes. Parent-child containment is a structural relationship. The user can define how those children appear and behave on the canvas rather than being restricted to one fixed arrangement.

Acceptance [P]: Create a parent with child nodes and recognize their containment and configured layout. Simultaneous containment parents remain open in D01.

N05 Use collapse to hide children

Basis: [U9] superseding collapse definition

A collapsed node is one whose child nodes are hidden. Collapse is not defined by a mandatory title-and-image appearance. The parent’s own media or alias representation is separate from the visibility of its children.

Acceptance [P]: Collapse a parent to hide its children, then expand to reveal them again without deletion. No title/thumbnail-only display restriction is enforced.

N06 Keep adjacency and sibling behavior distinct

Basis: [U9] adjacency and siblings

Nodes may be adjacent in the same canvas with no association beyond residing there. Sibling nodes remain visible independently: collapsing one does not hide the other. Co-location, structural membership and explicit relation links must not be treated as the same thing.

Acceptance [P]: Place unrelated neighbors without creating a semantic link. Collapse one sibling while the other remains visible.

N08 Offer sticky edges and overlap

Basis: [U9] configurable subnode behavior

Child nodes of the same generation may use sticky edges so their edges snap together when placed adjacent to one another. The user may instead allow overlap. These are configurable spatial behaviors, not an automatic assertion that the nodes are semantically related.

Acceptance [P]: Snap adjacent same-generation child edges where enabled. Permit overlap under the chosen configuration without silently creating a semantic relationship.

N09 Support cell based and nested containers

Basis: [U9, U13] spreadsheet-like and nested containers

A parent node may itself be a container that behaves like a spreadsheet surface, with subnodes placed in cells identified by two-dimensional coordinates. A container can itself have another container as a child. This does not settle multiple-parent containment. Formulas or spreadsheet calculation behavior have not been specified.

Acceptance [P]: Place child nodes in distinct coordinate-addressed cells and nest one container inside another. Cell occupancy, spanning and resizing details remain open.

Every node participates in the parent-child hierarchy. Root/leaf treatment, simultaneous parents, group semantics and contained movement/resizing remain open in D01 and D06; no required child count or root cycle is implied. [U11, U13]

5 References and original state

N17 Keep referenced state owned by the original context

Basis: [U14] model projection and no competing input

The original context directly determines a node's content/state, including changing values in a live system. A reference in another system presents a model or projection of that original. It cannot define a competing input that directly changes the original value and cannot write back to it. Referencing does not transfer ownership or create an independently executable clone.

Acceptance [P]: Observe a transistor reference in another system. Its value comes from the original context, and competing input through the reference cannot change that original.

N18 Use live local references and saved external snapshots

Basis: [U15, U16, U32, U34] reference freshness and export controls

A reference embedded in the same canvas workspace continuously reflects the original node's changes. A reference whose source is outside the canvas contains a snapshot from the last time the source canvas was saved or closed; it does not consume that external canvas's unsaved live state. When an outside source canvas is saved, consuming references automatically refresh to that saved snapshot. Both remain read-only projections with no competing input or writeback. These ownership and freshness rules apply within the existing environment; U34 governs download inclusion. Imported-package identity and reconnection remain D17.

Acceptance [P]: Change a local original and see continuous updates. Unsaved external changes remain hidden; saving the external source automatically refreshes its consuming references.

The original owns the state

The transistor example establishes a read-only projection of original state, not an executable clone or ownership transfer. The GUID identifies the node object, not its media bytes; the no-writeback rule is a separate behavior requirement. [U14]

Location of reference	Freshness	Mutation boundary
Same canvas workspace	Continuously reflects the original	The original context directly changes the original value
Outside source canvas	Last saved/closed snapshot; automatically refreshed on source save	No unsaved live external state or writeback through the reference

Snapshot boundaries still apply to calculations

An external reference continues to expose its saved snapshot even if the source has unsaved value changes or an unsaved undefined result. Downstream calculations use the value currently exposed to them; propagation does not bypass the snapshot boundary. The exact meaning of closing without saving remains open. Automatic refresh on save was chosen as a practical default, not a strong general preference. [U15, U16, U25]

Reference identity, deletion, broken targets and movement remain D02–D03. Modules create independent fresh-GUID instances (N47–N49). Containment cardinality and broader copying remain open.

6 Relation meaning and link behavior

N10 Distinguish causal mutual and cyclical relations

Basis: [U9] explicit visual distinctions

Links express information through a variety of relation types and visual forms. The specified causal, mutual and cyclical forms must remain distinct rather than being collapsed into one generic bidirectional edge.

Acceptance [P]: Create all three forms and distinguish one two-headed mutual line from two opposing cyclical arrows.

Relation	Required visible form	Stated meaning
Causal	One directed arrow	Causal relationship
Mutual	One line with arrowheads at both ends	Mutual relationship
Cyclical	A pair of opposing arrows	Feedback loop

N20 Distinguish visual links from value reactive links

Basis: [U18] generic and specialized link types

Provide a generic link that is insensitive to its endpoints and serves a visual purpose, such as an infographic. Provide specialized link types that respond to the values of input nodes. This distinction concerns functional response; it does not decide whether a visual line stays geometrically attached when nodes move.

Acceptance [P]: Changing endpoint values does not execute a generic link. A specialized connection responds through its configured joint operation under the circuit evaluation rules.

Meaning appearance and execution

Adjacency is not a semantic relationship. Containment is a structural relationship, not an implied drawn link. Sticky edges describe layout. A generic line can communicate information without computation. The causal, mutual and cyclical forms do not by themselves select an operation or start execution. Generic endpoint insensitivity concerns value-driven behavior, not a decision about geometric attachment when a node moves. [U9, U18–U21]

Additional labels, styles, relation types, direction-dependent functions and feedback semantics remain open. Systems may share a workspace without automatic conversion between their values or permitted connection types. [D04, D05]

7 Flexible routes and explicit joints

N11 Join connections into shared lines

Basis: [U9] converging lines

The user can extend lines from a set of nodes so the lines meet, then continue as one line to a related node. The routing must accommodate a joined structure rather than requiring a separate complete line for every source-destination pair.

Acceptance [P]: Join routes from several nodes at a meeting point, continue one line to the destination and adjust the resulting route.

N12 Attach lines to edges and other segments

Basis: [U9, U32] flexible visual snapping and precise functional endpoints

A line from a node can snap to any point on the edge of a destination node or to any point on a separate line segment. The design must not restrict the user to a small predetermined set of edge attachment points. Visual connections retain this free placement. A functional connection must also identify the exact input or output it addresses under N49; visual position alone must not leave its target ambiguous.

Acceptance [P]: Choose different points along a node boundary and an existing segment, then recognize the intended attachment location.

N22 Keep visual crossings separate until explicitly joined

Basis: [U21] crossing is not joining

When two links merely cross visually, they remain separate. A joint is created only when the user explicitly joins them. Geometric intersection alone must not combine their signals, connect their paths or create an operation.

Acceptance [P]: Cross two links without connecting them. Explicitly join them and identify the resulting joint and its properties.

N19 Define operations at joints

Basis: [U17, U19, U20] joint operations and user configuration

A joint is the meeting of two or more links. Joints define operations and must support the confirmed capability to combine signals or values under the applicable system rules. The user selects and configures a joint’s operation through its properties, subject to inherited constraints. Links connect; nodes define input values and output destinations. A merely geometric crossing remains separate until explicitly joined under N22.

Acceptance [P]: Explicitly join at least two links, configure the joint’s operation and observe its result from defined inputs. Inherited constraints still apply.

Keeping routes understandable

[P] Connected endpoints should remain visibly attached when the relevant node moves, while users can adjust routes and joints. Exact automatic rerouting, line curvature and manual controls remain open. Snapping to a segment is supported, but the precise gesture that confirms an explicit join still needs definition. A crossing alone remains separate. [U9, U21; D06]

Whether a joint is itself a node and has a GUID remains open. Its confirmed role is to define an operation chosen and configured by the user through its properties, within inherited constraints. [U19, U20; D02]

8 Working systems and inherited rules

N13 Provide functional node types and working systems

Basis: [U9, U19, U20, U31] functional toolset and clarified scope

Provide a library of node types for working systems as well as information maps. Nodes supply input values and output destinations; links connect and may react to values; joints define user-selected operations. The examples include logic systems, electric circuits and coding-related systems.

Acceptance [P]: Select appropriate node types, connect values and destinations, configure joint operations and observe the agreed behavior in a chosen example domain.

N21 Use nodes for input values and output destinations

Basis: [U19] corrected node role

Nodes define input values and output destinations in the working system. Links connect them through joints that define the operations. This clarification replaces the earlier broad calculation-node wording; it does not assign the calculation operation to a node.

Acceptance [P]: Identify value and destination nodes, their connecting links and the joint that defines the operation. Each role remains distinguishable.

N15 Scope types and behavior through parent nodes

Basis: [U10] parent-defined descendant types and system behavior

A parent node can define the node and relation types available to its descendants and the behavior of that system. Behavior is not locked into a mandatory mode at the canvas root. A workspace can therefore contain differently defined subsystems alongside explanatory images and text. Every node is bound by its inheritance and cannot override it. A node may add non-conflicting rules for descendants while retaining inherited rules.

Acceptance [P]: Place differently defined subsystems beside explanatory media. Each presents its permitted descendant types while retaining inherited constraints. Evaluation follows N23 for ordinary calculations and N34 for continuous signals.

N16 Add rules without overriding inheritance

Basis: [U11, U12] binding inheritance and non-conflicting additions

Every node participates in the parent-child hierarchy and is bound by its inheritance. No node can override inherited rules. A node may add non-conflicting rules for its descendants. This also applies to nodes that define a subsystem’s behavior.

Acceptance [P]: Add a compatible rule while retaining inherited rules. A contradictory rule cannot override them. No required child count or multiple-inheritance mechanism is implied.

A workspace can mix purposes

Different working systems and explanatory media coexist under binding inheritance. Rule conflicts, changes to existing rules, cross-system conversions and specialist fidelity remain open; compatible additions do not override inherited rules. [U10–U12; D05, D10]

9 Circuit evaluation switches and variables

N23 Evaluate on completion and connected operand changes

Basis: [U22, U23] initial and value-change evaluation

For the currently defined calculation behavior, as soon as a circuit becomes complete its operation occurs once. Thereafter, changing an operand value in the connected circuit immediately affects the solution; no open-close cycle is required. This combines initial completion evaluation with reactive updates on operand changes, without assuming free-running repeated execution while values remain unchanged. Completion criteria and feedback behavior remain open. The confirmed parent-scoped continuous-signal behavior is N34; ordinary calculations retain these rules.

Acceptance [P]: Complete a valid circuit and observe its solution. Change an operand while connected and see an immediate update without toggling a switch or repeatedly running unchanged inputs.

N24 Toggle circuits and clear open circuit solutions

Basis: [U22, U24] switches and undefined output

Provide switch objects that let the user toggle circuits between open and closed states. Closing may establish circuit completion, subject to the eventual completion criteria. Opening a circuit clears its current solution or makes it undefined. It does not retain the last solution as the current result or substitute numeric zero. Undefined then propagates to dependent calculations and clears their results under N26.

Acceptance [P]: Open a previously complete circuit. Its solution becomes undefined rather than the old value or zero; dependent results clear under N26. Reclosing a valid circuit evaluates again.

N26 Propagate undefined to dependent results

Basis: [U25] downstream undefined propagation

When an opened circuit clears its solution to undefined, calculations that depend on that output also receive undefined and clear their results. This follows dependency relationships rather than clearing unrelated calculations. The exact presentation of undefined and feedback-cycle handling remain open.

Acceptance [P]: Feed one result into another calculation, then open the source circuit. Both its solution and dependent results clear to undefined rather than displaying stale values.

N25 Use variable GUIDs as mathematical input references

Basis: [U22] GUID-addressable variables

Variables can be given GUIDs. Mathematical operations can use these GUIDs as input references to the variables and their values, rather than treating the identifier bits as arithmetic operands. Whether a variable must be a separate node, how variable GUIDs relate to node GUIDs and their namespace or lookup behavior remain to clarify.

Acceptance [P]: Use a variable’s GUID as a mathematical input reference and obtain its value, not the GUID string. A connected operand-value change updates the solution under N23.

Evaluation boundaries

Completion criteria, feedback and propagation order remain open. “Immediate” defines the intended experience, not a clock or numerical latency guarantee. Whether a variable is a separate node, and how variable and node GUIDs relate, remains unresolved. [U22–U25; D02, D04]

10 Waveforms layers and signal observation

N27 Generate pure tones and raw waveforms

Basis: [U26] tone and waveform creation

Meridian should let the user create pure tones and raw waveforms and control their inputs and output directly. Tone generation is a product capability; it must fit the existing division of input/destination nodes, connecting links and operation-defining joints. Parent-scoped continuous output follows N34; remaining timing and control details are in D11.

Acceptance [P]: Create a pure tone, change its frequency and level through visible controls, and inspect the resulting signal. Other waveform shapes and control details in the library below are proposed choices.

N28 Stack signal layers and apply effects

Basis: [U26] layering and effects

The user can combine waveform or signal layers and apply effects, while retaining direct control over contributing inputs and the resulting output. A meeting of links does not silently choose a mixing operation; a joint defines the user-configured operation.

Acceptance [P]: Layer two tones, control each contribution and observe the combined output. Apply a chosen effect and compare the result. Per-layer gain, mute/solo, panning and bypass are proposed controls.

N29 Display waveform and signal output

Basis: [U26] visual output

Show a visual representation of generated waveforms and their output, including the effect of stacking layers. The display should help the user understand how inputs and operations change the result. Spectrum, level meters and clipping indicators are useful proposed complements.

Acceptance [P]: Inspect individual layers and their combined waveform, then change an input or effect and see the relevant visual result. Display timing, scaling and channel selection remain to be chosen.

N34 Keep closed signal circuits producing continuous output

Basis: [U27] parent-scoped continuous-signal behavior

A parent-scoped continuous-signal system runs generators and effects while its circuit is closed. Closing starts continuous output, not a single arbitrarily bounded result. Opening stops the signal and clears dependent results to undefined. Ordinary calculations elsewhere retain N23–N26. The mode respects inherited rules and is not a mandatory canvas-wide mode. Explicit Run and Stop All controls govern deliberate activation under N53; opening a community canvas must not itself start sound or costly processing.

Acceptance [P]: Close a valid tone-generating circuit and observe sustained output without an arbitrary duration cutoff. Open it and hear output stop while its solution and dependent results clear to undefined. Ordinary calculation circuits keep their completion/operand-change behavior.

Max distinguishes event messages from a continuously evaluated signal network. [M1] Meridian now explicitly supports both its existing calculation behavior and parent-scoped continuous signals. This does not choose a global clock, scheduling scheme, feedback policy or reconnection phase behavior. [U27; D11]

11 Logic gates wiring symbols and outputs

N30 Include the basic Boolean logic gates

Basis: [U26] all basic logic gates

Include AND, OR, NOT, NAND, NOR, XOR and XNOR as available logic capabilities. Their familiar visible gate symbols must map onto Meridian's existing roles; the symbol does not create a second calculation-object model. Boolean and bitwise meanings must be distinct product choices.

Acceptance [P]: Configure each gate behavior and check its Boolean output. For two-input XOR exactly one true input gives true; two-input XNOR is true when the inputs match. Keep undefined distinct from a valid false result.

Gate	Boolean behavior
AND	True when every input is true
OR	True when at least one input is true
NOT	Inverts one Boolean input
NAND	Inverts the AND result
NOR	Inverts the OR result
XOR	For two inputs, true when exactly one is true
XNOR	For two inputs, true when the inputs match

[P] Useful companions include arithmetic, comparisons, min/max, range mapping, conditional selection, a buffer/identity operation and separately selected bitwise operators. [M9]

N31 Use a consistent wiring symbol vocabulary

Basis: [U26] wiring diagram symbols

Use recognizable symbols from wiring diagrams for logic and connected systems. Keep actual connections and explicit joints visually distinct from unconnected crossings. The symbol standard, families and mapping onto node/link/joint roles remain to select; no particular standard or electrical simulation fidelity is assumed.

Acceptance [P]: Represent a basic gate, source, switch, connection and output in a consistent vocabulary. A viewer can distinguish a crossing from an explicit joint. These example symbol families are proposals for the initial palette.

N32 Provide multiple forms of output

Basis: [U26] varied output and direct control

Support varied output experiences, including audible output and visual waveform/signal output. Users should understand which destination receives which result and be able to control the route. The broader output palette is proposed in the next section, not all mandatory at once.

Acceptance [P]: Route a generated result to an agreed audible destination and an observation display. Show destination and channel status. Numeric/Boolean displays, files, MIDI, external control and video outputs require their stated scope choices.

12 Optional component library and resource use

The user's direction is broad Max/MSP-inspired capability without burdensome resource demands. The following is a proposed feature catalog, not an accepted parity checklist. Max object names are research references, not a requirement to reproduce its runtime, patch-file compatibility, ecosystem or interface. Processing capabilities map to joint operations rather than silently becoming calculation nodes. [U26; M1–M10]

Family	Proposed capabilities
Generators	Sine/pure tone, saw/ramp, triangle, square/pulse, noise, constants and low-frequency modulation; wavetable/custom waveforms as an extension
Mix and route	Per-layer gain, pan, mute/solo, crossfade, channel selection and split/merge; explicit sum, multiply/modulate and selector operations
Samples and files	Audio-file playback, looping, position/rate controls, input/output recording and render/export destinations
Effects	Gain/invert, clamp, waveshaping, filters/EQ, delay/echo and dynamics; optional reverb, chorus/flanger and richer modulation
Control and timing	Knobs/sliders, triggers, envelopes/ADSR, ramps, LFOs, counters, clocks/tempo, step sequencing, random values and smoothing
Observe and output	Numeric/text/Boolean readouts, state lights, waveform and spectrum displays, meters and plots; signal-responsive visual properties
Input and integration	Audio devices/channels, MIDI, files; separately enabled network/OSC, serial hardware, cameras and external controls
Advanced extensions	Multichannel/polyphony, granular/spectral tools, video/image composition, 3D, custom signal operations and plug-in hosting

N33 Keep breadth bounded by usable resource demands

Basis: [U26] optional breadth with resource restraint

Favor a broad, useful component library while keeping the workspace responsive and avoiding burdensome resource use. Advanced capabilities are optional scope choices, not a requirement that every workspace run every component. No numerical performance or component-count budget is yet agreed.

Acceptance [P]: Use a representative layered signal workspace and keep navigation, editing, observation and stop/mute controls usable. Agree its scale and limits before treating a library family as accepted; expose resource pressure rather than silently degrading meaning.

[P] Consider visible workload indicators, adjustable display quality, bounded recordings/buffers and clear opt-in for expensive spectral/video features. Any automatic pausing, voice limits or dropped processing needs an explicit user-visible policy. Avoid selecting such policies by implication. [D13]

13 Canvas workflow and preserving work

N14 Navigate and arrange the canvas

Basis: [U1, U2, U9, U32] canvas zoom and confirmed discovery tools

The user can zoom in and out, arrange media nodes and work with relationships and child layouts. Search, backlinks, a synchronized tree outline and keyboard navigation are confirmed under N50–N51. Panning and restoring the current view remain proposed conveniences. Exact selection, collapse and activation gestures still need definition.

Acceptance [P]: Zoom between overview and detail without changing content or losing intended connections. Navigate to a chosen node and distinguish layout actions from functional actions.

Research walkthrough

Create a canvas for Greek mythology. Place descriptive text, images of artistic representations and complete source documents as media nodes. Use parent nodes to contain related material, expand them to reveal children and collapse them to reduce visible detail. Keep neighboring or sibling material visible independently. Use relation links to express genealogy or a chronological story; the workspace need not be an executable system. [U2, U9, U18]

Reference a node from another canvas by its universal identity and inspect its exposed model under the reference rules. This use does not restore the withdrawn encyclopedia-subject catalog model or assume that visual relationships are automatically copied to other canvases. [U9, U14–U16]

Working system walkthrough

Place input-value nodes and output destinations inside a parent-defined system. Connect specialized links and explicitly join them at operation-defining joints. Set joint properties, complete the circuit and observe the solution. Change an operand to update the result; open a switch to clear it and dependent results to undefined. Explanatory media can remain beside the system. [U10, U19–U25]

Preserving work

- [P] Save and reopen node identity, aliases, media references, containment, layout, routes, relation types and joint configuration without losing meaning
- [U32] Undo, named checkpoints and recovery of a previously working arrangement are confirmed in N54. [P] Also distinguish unsaved work, saved work and a failed save
- [P] Preserve an identifiable placeholder when media or a referenced target is missing, with a clear repair path
- [P] Make navigation, selection and inspection understandable at both overview and detail scales; provide discoverable controls rather than relying only on hidden gestures

File packaging, import/export formats, history retention, offline use and platform details remain open. Shared-canvas collaboration is confirmed; its permissions and interaction rules still need definition. These safeguards do not select a storage format or technical architecture. [D07–D10]

14 Viewing editing and reusable canvases

N35 Separate viewing mode from editing mode

Basis: [U28, U29] modes and local web-editor clarification

Provide a viewing mode for experiencing a finished canvas and an editing mode for creating and altering canvases. Viewing remains interactive, but the exact permitted controls versus structural or content editing still need definition. The editor is a downloadable, locally runnable web app under N42. Earlier standalone-versus-browser wording is superseded by U29.

Acceptance [P]: Open a finished canvas in viewing mode, interact with its permitted controls and clearly distinguish entry to editing mode. Do not treat viewing access alone as permission to author or publish changes.

N36 Distribute interactive finished canvas files

Basis: [U28] standalone finished files

A finished canvas can be distributed as a standalone file that remains viewable and interactive. Standalone describes the desired file-based experience; it does not yet require a native executable, guarantee offline operation or settle which viewer or external resources are needed.

Acceptance [P]: Open a distributed finished canvas and use the interactions allowed in viewing mode. Clearly identify any required viewer or unavailable resources instead of silently losing canvas content.

N37 Embed a finished canvas in a webpage

Basis: [U28] embedded viewing

Provide a way to embed a finished canvas in a webpage while retaining its permitted interactive viewing experience. Embedding does not by itself grant editing rights, device permissions or publication authority. The host interaction and media behavior remain to specify.

Acceptance [P]: Load an embedded finished canvas and interact within the approved viewing scope. Canvas content and permitted controls remain understandable without assuming access to the full editor.

N38 Download canvases and reopen them for editing

Basis: [U28, U32, U34] editable downloads with external inclusion choice

A user can download a Meridian canvas from a website and open it in editing mode to make changes. The file supports authoring rather than only a flattened screenshot. The downloader chooses external-reference inclusion under N62, with author-designated required canvases included by default under N63. This supersedes U32’s absolute local-only restriction. Reuse permissions and the identity or lineage of whole downloaded canvases remain open.

Acceptance [P]: Download a permitted canvas with the chosen dependency inclusion, inspect its contents and missing requirements, and edit it in the approved editor. The workflow does not overwrite the online original. Downloading a snippet creates a new blank canvas with fresh copied-node GUIDs under N64.

Viewing interaction still needs a boundary

Explicit Run and always-accessible Stop All are confirmed for viewers under N53. Pan/zoom, expansion, media playback, switch controls and changing exposed inputs remain candidate interactions, not yet a complete approved list. Define transient versus saved state and which actions require editing mode. Parent-scoped signal rules remain binding; remaining activation details are D20. [U32; P; D15]

15 Repositories publication and collaboration

N39 Publish selected canvases to personal and community repositories

Basis: [U28] selective online publication

Preserve personal repositories and an online community collection of canvas files. Users create and alter canvases in their editor, then selectively publish ready work to their personal online repository. Community discovery, audience and access controls remain to define. Publication is a deliberate choice, not assumed automatic synchronization or a default public audience.

Acceptance [P]: Keep unfinished work separate from an explicitly selected publication. Publish a chosen ready canvas through an approved workflow and make the intended audience and published item clear. Audience, versions and withdrawal rules remain open.

N40 Support a shared collaborative canvas workspace

Basis: [U28] collaboration

Preserve the ability for people to work together in a shared canvas workspace. The shared workspace must respect the distinction between viewing and editing and the established node, reference and system rules. Roles, invitations, concurrent changes, presence and shared execution state remain to define.

Acceptance [P]: Authorized collaborators can access the same agreed workspace and perform their permitted tasks. The experience distinguishes a viewing participant from an editing participant and explains changes or access limits without silently choosing conflict or runtime-ownership rules.

N41 Plan the editor and repository within the Atlas environment

Basis: [U28, U29] product destinations and deferred integrations

The intended eventual home for Meridian’s editor and repository is atlasuniversalis.com. The user also wants the editor embedded in MyNet on Hypershell. The downloadable local web-editor direction is now confirmed under N42. Hypershell, MyNet and Hyperlib integration details are deferred for a future agent; they do not restrict the editor to those environments. These are future product destinations, not a claim that deployment exists.

Acceptance [P]: Review a user journey from the approved editor entry point to a personal repository, a published canvas and its download/embed experience. Treat deeper Hypershell/MyNet/Hyperlib integration as deferred work rather than a prerequisite for local authoring. Hosting is a future product destination; this handoff does not deploy it.

N42 Run the downloadable web editor with local projects

Basis: [U29] hosted distribution and local authoring

Host and distribute the editor at atlasuniversalis.com while allowing users to download the web app and run it locally. Users can store Meridian projects locally rather than hosting every file online, and need not visit the website for every creation or editing session. Selective publication remains a separate action. This supersedes the earlier unresolved standalone-only/browser access wording.

Acceptance [P]: Download the editor, run it locally and create or edit a locally stored project without requiring an online repository for that project. Publish only when the user chooses. Complete offline operation with external media/references and update behavior remain to define.

16 Packing containers and reusable modules

N47 Save an assembly in a packing container

Basis: [U32, U33] packing container and node-based module definition

Let users wrap and save a collection of nodes, links and joints as a reusable node module. The packing and module-definition tool is itself a node under N60. Stored modules have a GUID, name and listed contents. Looking up a saved module to create a new instance is a different operation from observing an existing node through a read-only reference. The precise mapping between the definition node's GUID and the instantiation lookup ID remains open; no non-node recipe object or separate mandatory identifier namespace is introduced.

Acceptance [P]: Save an assembly through its module-definition node, find the stored module by its identifier and identify the collection to instantiate. Its name, GUID and contents are browsable. Invoking instantiation creates fresh nodes under N48 rather than a mutable reference to the source.

N48 Instantiate a fresh independent collection from its module ID

Basis: [U32] unique new node GUIDs and otherwise identical module

Searching the packing container's module ID in the dedicated tool generates a new instance of the node module. Every node in the new collection receives a unique new GUID. Preserve the saved collection's content, configuration, layout and internal relationships, remapping internal node references to the corresponding new GUIDs so they remain inside the new instance. The intended result is otherwise identical to the saved original. External dependencies are not resolved by that identity rule; their treatment remains D19.

Acceptance [P]: Instantiate one saved module twice. Each collection has fresh node GUIDs, retains the saved arrangement and internal connections, and can change independently. No internal reference accidentally points back to an original node. Do not silently clone or reconnect external source objects.

N49 Provide named interfaces through ordinary nodes

Basis: [U32, U33] approved interfaces and sibling output clarification

Reusable assemblies retain named inputs, outputs and adjustable controls without requiring every internal component to remain visible. These use the common node model rather than mandating a specialized wrapper display object. Output nodes can be siblings of a collapsed parent while linked to its hidden children under N59. This behavior applies to ordinary systems as well as packed modules. Functional connections identify their exact input or output, visual links retain free edge-snapping, and operations remain on joints.

Acceptance [P]: Collapse a waveform-generating parent while ordinary sibling output nodes remain visible and display the children's output. Address a named input/output or control without introducing a non-node display exception. Expanding the parent reveals its internal composition and joints.

Packing and referencing serve different purposes: a reference observes the original under its state-ownership rules; a saved module creates a new independent assembly. Destination inheritance remains binding. How to handle a saved assembly that conflicts with its destination's rules, or depends on outside objects, remains open. [U14, U32; D19]

17 Ordinary output nodes and module definition nodes

N59 Keep sibling output nodes visible when a parent collapses

Basis: [U33] ordinary node solution for independent displays

A parent may contain child nodes that generate a waveform or other output. Separate output nodes can be siblings of that parent while linked to its children. Collapsing the parent hides its children but leaves the sibling output nodes visible, displaying the connected output or selected properties. This is ordinary node behavior, available whether or not the system has been packed as a module. Collapse is a visibility action, not an instruction to open its circuit or discard the displayed result.

Acceptance [P]: Connect sibling output nodes to waveform-generating children, then collapse the children's parent. The sibling displays remain visible and show the output under the existing execution rules. Opening a circuit still clears its dependent results; merely hiding its nodes does not perform that action.

N60 Make the module definition tool a composable node

Basis: [U33] all-node module tooling

The tool used to define a reusable module is itself a node and follows the common node rules. Module-definition nodes can link to other nodes and form systems of modules that are stored and loaded together. The design must avoid specialized non-node canvas objects for this role. This does not assign calculation operations to the wrapper or choose an execution meaning for every module link; existing node/link/joint distinctions still apply.

Acceptance [P]: Place and link module-definition nodes into a system of modules, save that system and load it again with its connections and definitions intact. Creating a fresh module instance remains a distinct action from loading or referencing its definition.

N61 Browse stored modules by name contents and GUID

Basis: [U33] module library list

Present the user's stored module library as a browsable list. A stored module has a name and GUID, and lists its contents so the user can understand what it represents before loading or instantiating it. Dedicated identifier lookup under N47–N48 remains available. Thumbnail/tag browsing and version-management behavior are separate proposals.

Acceptance [P]: Browse stored modules, inspect one module's name, GUID and contents, and select the intended module. Distinguish loading its definition from creating a new independent instance where those actions differ.

Ordinary nodes provide composition, display and module tooling. No new exception to inheritance, reference ownership or circuit semantics follows from being an output or module-definition node. The exact GUID relationship between a definition node, its stored form and fresh instances remains D19. [U33]

18 Node dependencies and package summaries

N56 Declare dependencies at the node level

Basis: [U33] manual declarations and an exploratory dependency model

Treat GUID references as dependency relationships for tracking and checking. Allow the user to mark a node as dependent and define its dependency tree. These declarations describe what the node requires to operate equivalently; they are not containment, an automatic inference from every visual line, or permission to mutate the referenced original. The dependency design remains under discussion, particularly automatic discovery and validation.

Acceptance [P]: Mark a node as dependent, declare the nodes or context it needs, and inspect that declaration during module creation or canvas packaging. A generic visual relationship alone does not silently become an executable dependency.

N57 Account for dependencies of referenced nodes

Basis: [U33, U34] transitive context and chosen inclusion

Dependency tracking must account for a referenced node that itself relies on surrounding nodes, links or context to work equivalently. Track those transitive requirements rather than treating the referenced GUID as an isolated self-sufficient value. A dependency declaration identifies required context; whether external content is downloaded follows N62–N63 rather than automatic inclusion from discovery alone. Shared dependencies, cycles, optional requirements and completeness checks remain D21.

Acceptance [P]: Declare a node that depends on another node whose behavior relies on further links and context. The dependency view identifies that additional required context and does not claim the first referenced node alone is sufficient.

N58 Summarize package contents and required dependencies

Basis: [U33, U34] truthful package and dependency summary

When creating a module or preparing a canvas download, provide a package summary identifying relevant referenced GUIDs and canvases. The user suggested a “This package includes:” note; that label must describe content actually included. Distinguish included content from dependencies that are required but absent or unresolved, so an external reference is not falsely presented as bundled. The summary reflects the chosen external inclusion and required-canvas defaults. Exact metadata, identity mapping and automatic discovery remain open.

Acceptance [P]: Inspect a package with local content and external requirements under both inclusion choices. The summary labels actual contents accurately and identifies requirements that are missing, unavailable or excluded. It does not promise equivalent operation without the required context or silently expand the chosen package.

Export policy is now user controlled

U34 resolves the earlier inclusion question. The downloader chooses external-reference inclusion, and an author can mark an external canvas as required while making a nodal link so it is included by default. This supersedes U32’s absolute local-only restriction. A dependency list still describes what is needed; the chosen download determines what is actually present. [U33, U34; N62–N63]

19 Discovery and canvas packages

N50 Search text aliases metadata and collapsed descendants

Basis: [U32] approved search scope

Search reaches node text, aliases, media metadata and descendants hidden by collapse. Users can discover content inside canvases rather than relying only on the repository canvas or spatial memory. Search result scope and permission behavior remain to define.

Acceptance [P]: Find a text phrase, alias or media-metadata value inside a collapsed branch and navigate to the matching node without manually expanding every parent.

N51 Navigate backlinks original contexts and a synchronized outline

Basis: [U32] approved backlinks and outline navigation

Provide “Where is this referenced?” and “Open original context” actions for available GUID-based references. A synchronized tree outline and keyboard navigation support exploration of dense canvases. These tools respect read-only reference ownership and do not promise resolution of excluded external references in a downloaded package.

Acceptance [P]: Locate a node’s available referencing contexts, open its original context and navigate between the tree outline and canvas by keyboard. Explain missing or unavailable context instead of fabricating a target.

N52 Check canvas package completeness and dependencies

Basis: [U32–U34] package checks and downloader-controlled inclusion

Before sharing or downloading, check for missing files, external dependencies and private material so the user understands the package’s actual contents and remaining requirements. U34 replaces the blanket local-only rule with “Download External References?” and author-designated required-canvas defaults under N62–N63. Dependencies not selected for inclusion stay out of the package. A dependency declaration or summary must not silently expand the selected content or imply permission to access it.

Acceptance [P]: Review a canvas containing local content and external dependencies. The package follows the downloader’s choice and the stated defaults, and distinguishes included dependencies from those absent or inaccessible. It does not claim complete functionality when a necessary requirement is missing.

Existing same-workspace live references and saved cross-canvas snapshots still work within the user’s existing environment under N17–N18. They are not portability guarantees. A future, distinct truly universal reference ID for publicly hosted canvas nodes is only a possible later direction. Local asset packaging and the boundary of launched subcanvases remain D19. [U32]

20 External reference downloads and viewer snippets

N62 Let the downloader choose external reference inclusion

Basis: [U34] explicit download option

Offer the exact option “Download External References?” when downloading a canvas. The downloader chooses whether to include external references: some canvases need them for basic functionality, while other users may prefer to avoid unnecessary data. This replaces the earlier absolute local-only policy. Excluded dependencies remain outside the package and are identified as missing requirements where relevant. Defaults for optional unmarked references are not yet specified.

Acceptance [P]: Download the same canvas with external inclusion enabled and disabled. The resulting contents follow the chosen policy, and the summary explains any required content left out without silently substituting it or expanding the download.

N63 Default author designated required canvases to included

Basis: [U34] required external canvas at nodal link creation

While creating a nodal link, the author can explicitly designate an external canvas as a requirement. A download then includes such required linked canvases by default, while retaining the downloader’s external-inclusion choice. This default applies to author-marked requirements, not every visual link or every unmarked reference. Missing/inaccessible requirements and permissions still need an explicit product policy.

Acceptance [P]: Mark an external canvas required while making its link. A subsequent download initially includes that required canvas, and the downloader can review the inclusion choice and dependency summary. An unrelated visual link alone does not make its target a required download.

N64 Create downloadable snippets while viewing a canvas

Basis: [U34] selected nodes or region copied to a new canvas

In viewing mode, provide a tool to select multiple nodes or define a canvas region for download. Create a blank canvas and place the copied selection in that new space. Copied nodes receive a unique new set of GUIDs, as with module instances; included internal links and references are remapped to the corresponding new nodes. The viewed source remains unchanged. Apply the general export controls consistently without silently deciding the unresolved selection-boundary cases in D22.

Acceptance [P]: While viewing, select several nodes or a region and create a downloadable snippet. Open the new canvas with the copied content and fresh node GUIDs; internal relationships stay within the new copy. Creating or editing that copy does not modify the source canvas.

D22 Snippet selection and boundary details

Which partially intersecting nodes and contained children are copied by a region selection? What happens to crossing links, hidden descendants or children outside the region? Define external-dependency selection and required-canvas defaults for snippets, plus layout/origin details. General downloader control is confirmed; these boundary choices and permissions are not silently inferred. [U34]

21 Deliberate execution and dependable recovery

N53 Offer explicit Run and always accessible Stop All

Basis: [U32] approved deliberate run state

Opening a community canvas must not unexpectedly begin sound or expensive processing. Give viewers an explicit Run control and an always-accessible Stop All control. These controls provide deliberate activation and a way to stop active work. Ordinary calculations and continuous signal circuits retain their established evaluation rules when permitted to run. The exact scope, rearming behavior and result state of Stop All remain open; stopping is not automatically defined as opening every switch.

Acceptance [P]: Open a community canvas containing a closed signal circuit and confirm no unexpected output starts merely on opening. Start it deliberately with Run and stop active output with Stop All. The stop control remains accessible while the canvas is busy.

N54 Support undo named checkpoints and working state recovery

Basis: [U32] approved editing and collaboration safeguards

Provide dependable undo, named checkpoints and recovery of a previously working arrangement for editing and collaboration. Users can explore a change and restore an earlier useful state. History granularity, retention, shared versus personal undo and the relationship to live output state remain to define.

Acceptance [P]: Make a reversible edit, undo it, save a named checkpoint and recover that working arrangement after later changes. In shared work, make the affected scope understandable rather than silently choosing whose edits are rolled back.

N55 Preview the impact of inherited rule changes

Basis: [U32] approved descendant impact preview

Before changing a parent’s inherited rules, preview which descendants will be affected. The preview helps the user understand the consequences of the proposed change; it does not permit descendants to override inheritance. The existing requirement for non-conflicting added rules remains binding.

Acceptance [P]: Propose a parent-rule change and identify the descendants it would affect before applying it. Preserve existing inheritance constraints and explain any conflict rather than silently replacing descendant behavior.

Approval boundaries

The approved additions are reusable assemblies, search/backlinks/outline navigation, package checks with U34’s external-reference choice, and deliberate execution/recovery controls. The earlier advanced result inspector, pause, single-step and value-history proposal was not accepted as a bundle. The broader copy/reference workflow and standalone starter-template list remain proposals unless separately confirmed; U34 separately confirms viewer snippet copies. [U32, U34]

22 Structural and computational decisions

The following questions are open. Their order is a proposed order for the next discussion, not a development sequence. No answer is assumed from an earlier recommendation.

D01 Containment and hierarchy

Nested containers and nested canvases are confirmed. Can a child belong to more than one container simultaneously? How do “every node is a parent and child,” canvas roots and leaves fit together? Define reparenting, parent deletion and how generic groups relate to containers. Packing containers and fresh module instantiation are now confirmed in N47–N49. Single containment parent plus reference appearances was recommended but not selected.

D02 Identity of references joints and variables

Does a reference receive its own GUID? Is a joint a node, and does it have a GUID? Must a variable be its own node, and how does a variable GUID relate to a node GUID? Define alias scope, renaming and duplicate identity. Universal node GUIDs do not answer these questions.

D03 Reference lifecycle and cross canvas use

What snapshot is exposed if a source closes without saving? How do local reference appearance, broken targets, original deletion, duplication and moving an original behave? Save-triggered automatic refresh and no writeback are already settled. Whether a child can belong to several contexts is still D01.

D04 Circuit completion and feedback

What makes a circuit complete? How do several joints, cyclical feedback, missing variables and invalid inputs behave? How is undefined shown, and what ordering or conflict rules matter for dependent calculations? Initial evaluation, operand-change updates and downstream undefined clearing are already confirmed.

D05 System rules and connection compatibility

What counts as a conflict with an inherited rule? What happens when rules change or disappear? Which joint operations and specialized link types are available in each system? How are incompatible cross-system connections explained? Non-conflicting additions are allowed; inherited rules cannot be overridden.

Questions for the next agent

- Can one child belong to two containers at the same time, or does it have one containment parent?
- Is every variable a separate node with that node’s GUID, or can a node expose several GUID-addressed variables?
- What should happen when a calculation feeds back into its own inputs: reject it, evaluate under a chosen feedback rule, or something else?

23 Experience and scope decisions

D06 Layout and interaction

At what scope are sticky edges, overlap and cells configured? Do snapped peers move together? Can cells hold several nodes or span positions? Define parent movement/resizing, child clipping and expand/collapse/select gestures. Revisit how links to hidden children appear; that earlier question was paused, not answered.

D07 Media viewing and document editing

Which text, image, video and SVG capabilities are needed beyond viewing? How should complete DOCX documents open and edit? Are media edits written to originals or kept inside the project? Define viewer-pane navigation, playback behavior and missing-media recovery.

D08 Saving portability and exchange

External inclusion is now chosen by the downloader, with author-marked required canvases included by default. Which media assets and file formats travel with included nodes? Define local document/sample packaging, autosave and unavailable-file handling. The precise treatment of unselected, missing or inaccessible dependencies and linked subcanvases remains D19/D21; no canonical file schema is chosen. Snippet selection boundaries remain D22. [U32–U34]

D09 Reversibility and platform experience

Undo, named checkpoints and recovery are confirmed for editing and collaboration. Define undo/redo granularity, history retention, persistence across reopening and which actions belong to each participant’s history. [U32] The local web-editor direction is confirmed. Which platforms and completely offline use cases must be supported? Shared-canvas collaboration is confirmed; permissions and concurrent interaction remain D16. Historical notes disagree on undo readiness; no working behavior is assumed from them.

D10 First useful system and fidelity

Which research project or functional system should validate the product first? What operations, scale and fidelity make it useful? Logic, circuits and coding-related uses express the broader ambition; they do not promise engineering-grade simulation, a full language runtime or a delivery order.

24 Signal decisions and scope boundaries

D11 Continuous signals and ordinary calculations

Confirmed: a parent-scoped signal circuit produces continuous output while closed; opening stops it and clears dependents. Ordinary calculations retain their established behavior. Still define how scalar controls enter signals, how observed values return, and how start/reconnection affects phase and transport. Feedback, global timing and scheduling remain unspecified; inherited rules stay binding. [U27]

D12 Audio references and state ownership

Does a local read-only reference expose the original live stream without creating a second voice or playhead? What belongs to original state: phase, playback position, envelope, transport and recording? What does an external saved snapshot contain: configuration, a sample asset, a bounded recording or a control value? It must not silently become a live cross-canvas signal dependency.

D13 Library selection symbols outputs and safeguards

Which waveform shapes, effects and output types matter first? Which wiring-symbol standard and palette should be selectable? Define Boolean versus bitwise gates and multi-input XOR/XNOR. Explicit Run and Stop All are confirmed. Still agree feedback controls, output levels, resource visibility, limits and device permissions. Advanced integrations and expensive visual/spectral features remain optional proposals.

Listening controls

[U32] Deliberate Run and always-accessible Stop All are confirmed in N53. [P] Add visible level, instant mute and clipping feedback. Show undefined in the graph when disconnected; an audio destination may silence invalid output without treating semantic undefined as numeric zero. Any limiting or smoothing behavior needs agreement and is not a hearing-safety guarantee. [D13]

Scope of this revision

This is a product and functional specification. It records confirmed behavior, review examples and open choices. It deliberately leaves the technical approach for a later discussion. It does not claim that the recovered scaffold implements these behaviors or that the missing local application has been verified.

Historical contradictions remain evidence

The recovered project descriptions disagree about undo/redo status, identifier representations and document schema details. Current user decisions establish universal node GUIDs, but do not choose migration rules or a canonical file schema. Those technical contradictions stay unresolved in the separate recovery pack rather than being silently converted into product decisions. [D1–D4]

25 Distribution and collaboration decisions

D14 Local web editor and deferred integrations

Resolved direction: the web editor is hosted/distributed at atlasuniversalis.com, can be downloaded and run locally, and supports local project storage without requiring a site visit for every editing session. Still define installation/update experience and any offline limits caused by external resources. Hypershell/MyNet/Hyperlib details are deliberately deferred for a future agent, not a blocking question or an invented architecture. [U29]

Local authoring and publication

The user compares the desired flow to creating Hyperlib pages locally and selectively publishing them to Atlas archives. That analogy establishes author-then-publish intent; it does not establish offline support, a shared storage format or an existing integration. U29 clarifies a web editor hosted/distributed at atlasuniversalis.com that can be downloaded to run locally, with projects stored locally and publication selected separately. Integration details are deferred. [U28, U29; D14]

D15 Viewing interaction and published behavior

Explicit Run, always-accessible Stop All and creating downloadable snippets from selected nodes or a region are confirmed for viewers. Snippets create a new canvas and do not edit the viewed source. Which other controls are allowed: navigation, expand/collapse, media playback, input changes and switches? Are interactions transient or saved? What distinguishes authoring from an interactive published canvas? Define the remaining Run/Stop state details and device access for standalone files and webpage embeds without treating an embed as unrestricted execution permission.

D16 Collaboration audience and authority

Who may view, edit, invite, publish or withdraw a canvas? What does community availability mean, and how is audience selected? How do simultaneous edits, personal versus shared views and joint properties behave? For a running signal system, whose inputs, transport and output devices are authoritative? Original-context state ownership remains binding; collaboration does not silently redefine it.

D17 Downloaded files references and publication lifecycle

U34 allows chosen external-reference downloads and defaults author-marked required canvases to included. Whole-canvas and included-external-package GUID handling, reconnection and lineage remain open; this does not decide them by analogy with module or snippet copies, whose fresh node GUIDs are confirmed. Define media packaging, embeds, absent-dependency presentation, versions, forks, attribution/reuse permissions and republishing. A truly universal public-hosted reference ID remains a future possibility. No automatic sync, default public audience or executable file format is assumed. [U32–U34]

D18 Canvas representation and repository boundaries

How does a canvas-node appearance identify the canvas root it opens: the same GUID, its own GUID, or a reference? How do repeated appearances, moving or deleting a canvas node and unavailable canvases behave? Can a canvas contain or link back to an ancestor? Which links, if any, can attach to an embedded canvas node in an ordinary workspace? Define repository browsing versus personal/community publication, permissions and return navigation. U30 confirms the navigation layer and visual-only repository links; these identity and lifecycle details remain open.

26 Module boundaries and deliberate run decisions

D19 Saved modules and export boundaries

Module definitions are nodes, and stored modules have browsable names, contents and GUIDs. How does their node GUID relate to the instantiation lookup ID, storage scope and versioning? What exactly is saved: configuration only, current values or runtime state too? Node-level dependency declarations now identify needed context under N56–N58. All new node GUIDs and internal remapping are confirmed, but joint and variable identity still need D02. How should a module that depends on an outside node be saved or instantiated while meeting “otherwise identical” behavior? Do not assume external cloning, automatic rebinding or imported snapshots. Destination inheritance remains binding; define the user experience when a saved module is incompatible. [U32, U33]

U34 confirms external inclusion choice and author-marked required-canvas defaults. An ordinary visual link does not by itself mark another canvas required. Still define which media and launched workspace dependencies a package contains, how missing or excluded inputs appear, and how module definitions with outside requirements are stored or instantiated. Including external canvases must respect the chosen scope rather than silently traversing every visual link. [U32–U34]

D20 Run Stop All and recovery semantics

Which active systems and windows does Stop All stop? What result state does it leave, and what action permits output to resume? How does Run interact with completed circuits, switches, ordinary operand changes and continuous signals? Opening a circuit still clears its solution and dependents to undefined; Stop All has not been declared equivalent to that action. Define device activation and recovery of running states without automatically restarting costly or audible work. [U22–U27, U32]

Later pitches remain proposals

[U33] A browsable module library with names, contents and GUIDs is confirmed. [P] Tags, thumbnails and versions that do not silently change existing instances remain proposed. Sibling output nodes now address the collapsed-display case; no specialized wrapper display is required. A fixed palette of frequency/amplitude controls or document slots remains a proposal.

[P] A detailed preflight view could show what remains and what is removed. One proposed handling of an excluded reference is a marked disconnected input, rather than a substituted value. U34 confirms the inclusion choice and required-canvas defaults, but the exact placeholder and launched-workspace treatment remain open.

D21 Dependency declarations and external packaging

External inclusion is resolved by U34: downloader choice plus default inclusion of author-marked required canvases. Still define the default for optional unmarked references, inaccessible/missing requirements, permissions, transitive and cyclic dependencies, shared requirements and whole-package GUID mapping. How are declarations validated, and which surrounding links, rules or state are essential? “Tree” does not yet require a strict tree rather than a graph. Metadata and warning details remain open; a required marker is not an access grant. [U33, U34]

27 Source basis and revision history

The 3 October 2026 all-node correction, subsequent Q&A, signal expansion, local-editor direction and canvas-repository clarification and 4 October module, dependency, export-choice and snippet clarifications govern this handoff. The conversation appendix records the historical discussion verbatim; this current-concept section identifies the governing decisions. Source labels identify the relevant decisions; later clarifications supersede conflicting earlier wording.

Source	Confirmed decision
U1–U2	Original media-rich 2D canvas; research example, zoom and complete PDF/DOCX/TXT documents; product focus without build details
U9	Everything is a node; canvas root, aliases and GUIDs, nested nodes, collapse, siblings, layouts, relation forms and live systems
U10–U13	Parent-scoped systems, non-overridable inheritance, compatible added rules and nested containers
U14	Original-context state ownership; references expose content models, with no competing input or writeback
U15–U16	Local continuous references; external saved/closed snapshots automatically refreshed on source save
U17–U18	Combining capability; generic visual and specialized value-reactive links; later role clarification governs calculation wording
U19–U21	Joints meet two or more links and define operations; nodes define values/destinations; user configures joint properties; crossing requires explicit joining
U22–U23	Circuit completion evaluates once; switches and GUID-addressable variables; connected operand changes immediately update the solution
U24–U25	Opening clears the solution to undefined, which propagates and clears dependent downstream results

28 Recent direction and supersessions

The following later decisions extend or narrow the foundation above. U35 requests a plain whole-product opening before component detail; it adds no product behavior. The current requirements and open decisions are unchanged in this editorial revision.

Source	Confirmed decision
U26	Pure tones/waveforms, layers/effects, visual output, gates, wiring symbols and varied outputs; resource-bounded Max-inspired breadth
U27	Parent-scoped signal circuits produce continuous output while closed; opening stops output and clears dependents; ordinary calculations retain their rules
U28	View/edit modes, interactive files/embeds, download-to-edit, selective repositories and collaboration; U29 clarifies local web-editor access
U29	Hosted/distributed downloadable local web editor; local projects; selective publication; deferred integrations; current-concept handoff plus complete Meridian-only transcript
U30	Navigable canvas repository; special titled canvas nodes with optional images, child-canvas expansion and new-window opening; visual-only repository links; canvas nodes inside ordinary workspaces
U31	Removes an illustrative domain example from planned product scope; built-in templates for that example are not requested
U32	Fresh-GUID packing modules; named I/O; discovery and recovery; initial local-only export policy is superseded by U34
U33	Node dependency declarations and transitive context; truthful summaries; sibling outputs; module-definition nodes and browsable names/contents/GUIDs; inclusion question resolved by U34
U34	Downloader chooses external references; author-marked required canvases default included; viewer snippets create a blank canvas with fresh copied-node GUIDs

29 Recovered sources and Max research

Superseded discussion

The earlier subject/display-entry/catalog model and title/thumbnail-only collapsed appearance were withdrawn. U19's node/link/joint roles govern prior calculation-node wording. U31 removes an earlier illustrative domain example from planned scope; the original wording and correction remain in the transcript. Only the current requirements apply.

Recovered project background

The separate recovery pack preserves 59 original hash-verified repository files, including the documents below. The described 47-file local application was not recovered. Historical records do not verify this revised product model. Repository links may require access.

D1 Overview | D2 Inventory | D3 Checklist | D4 Project log
Official Cycling '74 references ground the optional feature vocabulary; they do not establish Meridian implementation, compatibility or accepted scope. Research checked 3 October 2026.

M1 How MSP works

Event messages and continuously evaluated signal networks are different concepts. This distinction informs the confirmed parent-scoped signal behavior and its remaining timing questions.

M2 Oscillator reference

Periodic tone and waveform generation, including wavetable use.

M3 Matrix routing and mixing

Routing and mixing motivate explicit mix/selection operations.

M4 Sample playback

Sample playback, looping and playback-rate capabilities inform the optional media family.

M5 Filter processing

Filter processing is one effect family; choosing exact effects remains a product decision.

M6 Envelope generation

Attack, decay, sustain and release are a useful control-envelope vocabulary.

M7 Waveform observation

An oscilloscope-style display provides time-domain observation.

M8 Spectrum observation

A spectrum display offers a complementary proposed view.

M9 Expressions and logic

Arithmetic, comparisons, logical and bitwise operations inform the library; Meridian keeps its own operand-update behavior.

M10 Optional audiovisual expansion

Broader audiovisual and 3D capabilities are optional inspiration, not a baseline obligation.

Part Two Meridian conversation transcript

This appendix preserves the Meridian-related conversation in chronological order, including the earlier search for the canvas application. Text is verbatim, including typos and ideas later superseded. Two mixed-topic messages are explicitly marked as excerpts; unrelated discussion is omitted. Times are UTC. Historical conversation is not a statement of current requirements. Attachment contents are not reproduced.

Coverage: 115 messages, comprising 44 user messages and 71 assistant messages, from 30 September 2026 through the opening rewrite request and follow-up on 4 October 2026. The record includes 113 full messages and 2 labeled excerpts. Available attachment captions remain in their original messages; attachment files are not reproduced.

The source conversation was retrieved to its beginning and checked for overlapping records and continuity. No gap was identified in the returned Meridian discussion. Completeness refers to the visible conversation text available in that record, not hidden or deleted events or attachment contents.

01 User

30 September 2026 | 22:33:34 UTC

Cool - thank you. I forgot... most of my apps live on my desktop pc =(Can you take a look at my github and give me just a really brief 2 sentence description of each of the apps in my repo? github.com/idiehl is me. I'm looking for one application in particular and I want to know how it's coded (what language/toolset). It's an app that contains a flexible dynamic canvas space that allows users to create linked visual graphs in a 2d canvas environment where each node on the graph can be a large variety of things like images, text blocks, videos, or graphics. Each node can be connected via a visual "branch" and nodes can act like containers to other nodes. I think it might be on my github. Can you try and track it down for me?

02 Assistant

30 September 2026 | 22:33:39 UTC

I’ll check your GitHub projects and give each app a two-sentence summary, including its language and framework. I’ll look especially for that canvas app with connected multimedia nodes and nested containers

03 Assistant

30 September 2026 | 22:36:30 UTC

The closest match is your Unified Canvas Platform family: [desktop_setup](https://github.com/idiehl/desktop_setup) has draggable text/image nodes in C# Windows Forms, while [DirectX_WIndow_Setup](https://github.com/idiehl/DirectX_WIndow_Setup) uses C# WPF/.NET 8 and defines nested nodes. Neither uploaded version contains the full working combination of connections, video, and nested containers you described, so the more developed version may still be on your desktop

04 Assistant

30 September 2026 | 22:37:29 UTC

Excerpt: the opening and four canvas-app candidates are retained verbatim. Unrelated repository descriptions and the closing inventory summary are omitted.

I found 19 app/prototype repos among your 22 repositories. Here’s the two-sentence rundown, with the canvas candidates first:

1. [desktop_setup — Unified Canvas Platform](https://github.com/idiehl/desktop_setup) is a canvas prototype with movable text/image nodes, grid snapping, and property editing. It uses C# and Windows Forms on .NET Framework 4.8.1.

2. [DirectX_WIndow_Setup — Unified Canvas Platform](https://github.com/idiehl/DirectX_WIndow_Setup) sketches a modular canvas with nested nodes, physics, and animation, but rendering remains unfinished. It uses C#, WPF, .NET 8, and CommunityToolkit.Mvvm; DirectX rendering is still a placeholder.

3. [WPF_desktop](https://github.com/idiehl/WPF_desktop) is a lightweight node canvas with pan/zoom, property editing, and JSON save/load. It uses C#, XAML, WPF, and .NET 8 with MVVM.

4. [unfinished — GraphDesk](https://github.com/idiehl/unfinished) is a drawing-canvas prototype with layers, text, shapes, pan/zoom, and exports. It uses Python, Streamlit, streamlit-elements, and JavaScript/Fabric.js.

05 User

30 September 2026 | 22:45:19 UTC

Excerpt: the canvas-search acknowledgment is retained verbatim. The subsequent unrelated email request is omitted.

I think it is, unfortunately.

06 User

1 October 2026 | 19:06:50 UTC

Excellent. I've managed to track down the app I was looking for yesterday. It's called Atlas Meridian and it's featured on [atlasuniversalis.com](https://atlasuniversalis.com/ProjectLab) Can you check out the codebase contained at the link to my github and see if it lines up with the app I described? I'm a little skeptical but it could be right. Again it should be an app containing a canvas space where the user can create linked nodes containing a variety of object types (images, videos, text, svg) and build a visual graph.

07 Assistant

1 October 2026 | 19:07:00 UTC

I’ll follow Atlas Meridian’s GitHub link and check the implementation for connected nodes, multimedia content, and nested containers, rather than relying on the project description

08 Assistant

1 October 2026 | 19:10:27 UTC

Atlas Meridian is a much stronger match, but your skepticism is justified: its documentation describes the app you remember, while the linked GitHub code is only an unfinished scaffold.

What’s actually in the code:
- C# and XAML, using Avalonia 11.3.11 on .NET 10
- Models for nodes and connections, including arrowheads and curved/straight branches
- Generic content fields and parent-frame relationships
- JSON saving/loading logic
- A static window with no working canvas, media rendering, connection editing, or nesting interaction

The important discovery is in its [overview](https://github.com/idiehl/site_repo/blob/480cd5bfbd4a5f418e487e44e32404228bd5e5a6/docs/master_log/Meridian_Overview.md): it points to a separate local codebase at D:\local\Atlas_Meridian, with a solution named AtlasUniversalisAvalonia.sln. Its [inventory](https://github.com/idiehl/site_repo/blob/480cd5bfbd4a5f418e487e44e32404228bd5e5a6/docs/master_log/Meridian_Inventory.md) describes 47 authored files, a custom canvas renderer, media overlays, and nested frames, with Skia-backed Avalonia rendering and LibVLCSharp for video.

Those fuller implementation files weren’t in the branches and history I checked. SVG support also remains unverified.

My take: probably the right project, but the website's GitHub link leads to an incomplete copy. D:\local\Atlas_Meridian is the folder to look for on your desktop

09 User

1 October 2026 | 19:12:58 UTC

That folder likely lives on my other pc that I don't have access to. How much of the project's documentation are you able to salvage? Can you pull from the README? It'd be nice to at least gather enough to avoid having to retype the entire concept from scratch again.

10 Assistant

1 October 2026 | 19:13:06 UTC

Yes. The README, archived project notes, and remaining code should let me recover a useful specification. I'll gather them into a brief that preserves the concept and clearly marks what was implemented, planned, or still uncertain

11 Assistant

1 October 2026 | 19:16:56 UTC

There's enough to recover a useful starting point: four substantial project documents, a detailed inventory of the missing source files, and map-generator instructions with an example .atlas file. I also found references to ten original concept PDFs, though those references don't include the PDFs themselves

12 Assistant

1 October 2026 | 19:24:40 UTC

I recovered 59 original files and put together a seven-page reconstruction brief plus a reusable Codex restart prompt. I'm saving the files to your Library now

13 Assistant

1 October 2026 | 19:25:37 UTC

Recovered enough to avoid starting from scratch:

- Editable brief: the concept, canvas interactions, media nodes, connections, nested frames, architecture, and unresolved design choices
- Recovery ZIP: 59 original files, an example .atlas file, and a Codex restart prompt

The brief distinguishes working scaffold code from documented plans and flags conflicting specifications. The full app from your other PC is still missing, but its design is now preserved

14 User

2 October 2026 | 11:21:32 UTC

Ok back to Atlas Meridian. So can you gather all the information you can on the project and draft a new project specification sheet that I can use to jump start the development process when I decide to pick it back up? Maybe once you've collected all the information available we can fill in the gaps with a Q and A session and I can provide the rest of the information required to begin work.

15 Assistant

2 October 2026 | 11:21:47 UTC

Yes. I'll build on the recovery pack to draft a development-ready specification, with confirmed requirements, proposed choices, and unresolved gaps clearly marked. Then we can work through the gaps in a focused Q&A.

16 Assistant

2 October 2026 | 11:31:49 UTC

The draft now has 18 requirements with acceptance tests and a prioritized decision list. One important gap: the recovered documents describe nested frames, while you described nodes that can themselves contain other nodes. I've kept that distinction open for our Q&A rather than assuming they're the same.

17 Assistant

2 October 2026 | 11:34:42 UTC

The specification is ready. I'm saving an editable Word copy and a PDF to your Library.

18 Assistant

2 October 2026 | 11:35:50 UTC

Here's Atlas Meridian's draft project specification, in editable Word and PDF formats. It includes 18 requirements with acceptance criteria, the canvas and content model, architecture, development phases, and an 11-item decision list for our Q&A.

Recovered facts and proposed choices are labeled separately, so it's usable as a starting point without locking you into assumptions.

19 Assistant

2 October 2026 | 11:36:11 UTC

Let's start with the purpose: what's the first real thing you'd want to create in Meridian? For example, a research map, an interactive presentation, a project plan, or something else. Your example will help us decide what the first version actually needs.

20 User

2 October 2026 | 14:50:07 UTC

No need to include stuff about development cycles or details about how to build it. I want to rework that part of the project a bit anyways. I picture using Meridian for research - I imagine for example mapping out the genealogy of of Greek gods but to be able to do more than that. To be able to create entries for each god that contains a variety of media. An extensive text entry describing their influence. Images of artistic representations and statues. Most important though would be the gods' link to other gods. I envision utilizing both visual links between nodes, and linking conceptually to nodes from other canvas spaces. I want to be able to collapse/expand nodes. Create "node groups" where nodes bind to other nodes forming modules. Need to be able to zoom in/out on the canvas workspace. Want to be able to include complete documents of a variety of types (pdf,docx,txt). There are a few more important details but I'll digress for now and continue the Q/A! My apologies, I actually forgot I was answering a question there for a moment.

21 Assistant

2 October 2026 | 14:50:23 UTC

That gives Meridian a clear purpose: a research workspace with rich entries, meaningful relationships, and connections across canvases. I'll remove the build instructions and development-cycle sections, and capture your requirements for mixed media, full documents, collapsible nodes, bound node groups, and canvas zoom.

Next question: if Zeus appears on both a genealogy canvas and a Trojan War canvas, should those be two views of the same entry, so editing his description updates it everywhere, while each canvas keeps its own layout and visible connections?

22 User

2 October 2026 | 16:01:18 UTC

Yes exactly - I want the visual aspect of the canvas to be local and the descriptive aspect to be universal (spanning across canvas spaces).

23 Assistant

2 October 2026 | 16:01:30 UTC

Got it: one shared entry for Zeus, with his description and media available across canvases, while each canvas controls its own visual arrangement.

Should relationships also be universal? For example, if you link Zeus as Athena’s father on one canvas, should that relationship be available on every canvas containing them, with each canvas choosing whether to display the connection?

24 User

2 October 2026 | 16:14:16 UTC

No - each canvas will explicitly define its visual relationships. One canvas may depict genealogy of gods while the other tells a chronological story containing the same characters. Each canvas should have an isolated set of visual relationships. A node or node group's subject should persist though - Zeus should have an encyclopedia-like entry associated with him outlining his connections/relationships/associations across all canvas spaces.

25 Assistant

2 October 2026 | 16:14:26 UTC

Understood: each canvas defines its own connections, while Zeus’s shared encyclopedia entry brings together his relationships and associations across canvases. That shared entry can belong to a node or a node group.

Should that entry automatically list relationships drawn on canvases, with links back to their source canvases, or should you curate which relationships appear there?

26 User

2 October 2026 | 16:16:33 UTC

It should automatically list them - it should act like a subject catalog. A quick way to accomplish deep research on a topic. This app is all about condensing an enormous amount of information and presenting it in a way that makes it easier to digest and comprehend.

27 Assistant

2 October 2026 | 16:16:45 UTC

Then the subject catalog will automatically gather those relationships across canvases, with links back to their context. Meridian’s core purpose is to make a large body of research easier to explore and understand.

When you collapse a node or group, what should remain visible: just its title/icon, a short summary or thumbnail, or a preview you choose for each one?

28 User

2 October 2026 | 16:18:03 UTC

A collapsed node would display either a title, a thumbnail, or both (at maximum).

29 Assistant

2 October 2026 | 16:18:16 UTC

Collapsed nodes show a title, a thumbnail, or both, with no additional content.

When you expand a node, should its content unfold directly on the canvas, open in a separate reading panel, or let you choose either?

30 User

3 October 2026 | 04:10:45 UTC

There should be two types of entry - a display entry (items shown on the canvas in expanded view) and normal entries. Single clicking a node should expand it. Right click > open should generate a pop up window with the subject's complete catalog of entries of various types.

31 Assistant

3 October 2026 | 04:11:19 UTC

Got it:
• Display entries appear on the canvas when a node is expanded
• Normal entries are available in the subject’s complete catalog
• A single click expands the node; right-click → Open brings up the full catalog in a pop-up

Should each canvas choose its own display entries for a subject, or should Zeus show the same display entries wherever he appears?

32 User

3 October 2026 | 04:22:53 UTC

No... display entries will be all the media associated with subject "Zeus" on a canvas. The display entries for Zeus could be a collection of pictures, a video, and a block of text. These are the elements that will appear on the canvas in its default view (with the entire network of nodes/subjects visible with connections). I think it's also important to mention that a node/subject can contain other nodes so that when it expands, so too do its contained collection of sub-nodes appear on the canvas. This is a key aspect of the design. This is really the core functionality of Meridian (not the display entries concept). Display entries are simply the way a node appears in its expanded view... the set of media the user has assigned to that particular subject.

33 Assistant

3 October 2026 | 04:23:02 UTC

I had put too much emphasis on display entries. The core is nested nodes: expanding Zeus reveals his assigned media and his contained sub-nodes directly on the canvas, within the larger connected network. Display entries simply define his expanded appearance.

When Zeus is collapsed, should connections from his hidden sub-nodes to outside nodes remain visible as connections to collapsed Zeus, or disappear until he expands again?

34 User

3 October 2026 | 05:09:35 UTC

I've got this wrong - I'm just now remembering some of the critical infrastructure of the design. In Meridian, everything is a node. Scratch everything about this "display entry" nonsense and subjects. The root node of a canvas is the canvas object itself - its title and any image associated with it. Nodes can have a variety of relationships with other nodes. They can be sub-nodes contained within a parent node. They can be adjacent nodes in the same canvas space with no association whatsoever except that they reside in the same canvas. They can be siblings - both nodes

are visible regardless of whether one or the other is collapsed. A collapsed node doesn't display its title and an image as a requirement as I said before. A collapsed node is one where its children nodes are hidden. At minimum a node is a single piece of media in any form placed on the canvas. Certain media types (like a PDF or text document) can be represented in the canvas by their alias - a text title assigned to the media that acts as a hyperlink to launch the appropriate viewer/loader in a separate pane. Another important concept - a node is at minimum a single piece of media placed in canvas space. Any node can also be given an alias (like its handle/title), and every node is assigned a GUID that can be used to reference the node universally throughout Meridian (across canvas). On sub nodes: One important aspect of Meridian is that nodes can be related in a variety of ways and relations can be represented visually on the canvas in a variety of ways. The user can define sub nodes to appear and behave in the canvas in a variety of ways. Sub nodes may have sticky edges so that when the user places them adjacent to another sub/child node of the same generation, their edges snap together. Or the user may allow them to overlap. Or another important concept - the parent node may itself be a container. Container nodes can act like spreadsheets with sub nodes placed in 'cells' with 2 dimensional coordinates. Back to relations - the links between nodes play an important role in expressing the information the user needs to convey. Links can be causal (an arrow extending from one node to another), mutual (a line extending between nodes with arrows on both ends of the line), cyclical (a pair of arrows pointing in either direction indicating a 'feedback loop'). It's important that the lines between related nodes are dynamic and flexible - the user should be able to extend lined from a set of nodes that all meet before extending as a single line to a single related node. Lines extending from nodes should be able to 'snap' to any point on the edge of a destination node or to any point on a separate line segment. Prior to this, I was working to develop a power toolset in Meridian where a canvas space could be used to represent a variety of working systems. Nodes/relations could be used to express things like logic gates, electric circuits, plumbing, coding languages, and more. The plan was to build in a library of node types with unique functions and a collection of relations that executed operations in real time in the canvas space. It was more than simply a means to plot stories or information systems.

35 Assistant

3 October 2026 | 05:09:51 UTC

That changes the foundation substantially. I'll replace the subject/display-entry model with this node-based design:

- Everything is a node, including the canvas root; nodes have universal GUIDs and optional aliases
- Collapsing hides children, rather than prescribing how the parent looks
- Media can appear directly or through an alias that opens a viewer pane
- Children support configurable layouts: snapping, overlap, or spreadsheet-like container cells
- Connections support direction, feedback loops, shared junctions, and attachment to node edges or other lines
- Libraries of functional nodes and relations can turn a canvas into a working system with real-time operations

Next question: should ordinary research canvases and executable systems use the same underlying nodes and connections, with behavior enabled when needed, or should a canvas have a distinct mode that determines whether its connections execute anything?

36 User

3 October 2026 | 05:19:04 UTC

Ooh I like that idea. A parent node should be able to define the available node/relation types available to descendents and define the behavior of the system. Lets avoid locking in behavior at the root node (canvas) layer though so images and text can be contained in the same workspace as other types of node systems for informative purposes.

37 Assistant

3 October 2026 | 05:19:15 UTC

Yes: a parent node can define the node types, relation types, and system behavior available to its descendants. That lets a circuit, explanatory text, and images coexist on one canvas without a canvas-wide mode.

If a descendant is itself a parent, should it inherit those rules by default and be able to extend or override them for its own descendants?

38 User

3 October 2026 | 05:20:24 UTC

Every node is a parent and a child - no node should be able to override its inheritance.

39 Assistant

3 October 2026 | 05:20:37 UTC

Understood: every node participates in the parent–child hierarchy, and inherited rules cannot be overridden.

Can a node add rules for its descendants as long as they don’t conflict with inherited rules, or must it pass down exactly the same rules?

40 User

3 October 2026 | 05:22:06 UTC

A node can add rules

41 Assistant

3 October 2026 | 05:22:27 UTC

Then rules accumulate down the hierarchy: a node can add rules, but cannot contradict or override inherited ones.

Can a node have more than one containment parent, or exactly one? This is separate from having any number of visual connections.

42 User

3 October 2026 | 05:23:08 UTC

Yes - a container can have a container as a child

43 Assistant

3 October 2026 | 05:23:21 UTC

Yes, containers can nest inside other containers. To clarify my question: can the same child node belong to two containers at once, or does it have one containment parent?

44 User

3 October 2026 | 05:30:11 UTC

This is an interesting question. What do you think? How do you think Meridian should behave and what do you think the effects of either decision would be? I cant quite tell? Consider a few cases where this would be applied.

45 Assistant

3 October 2026 | 05:30:21 UTC

I’d lean toward one containment parent per node, with references that let the same node appear elsewhere. I’ll walk through what that means for research, nested containers, and working systems before we settle on it.

46 Assistant

3 October 2026 | 05:31:59 UTC

My recommendation is one containment parent per node, while allowing that node to be referenced and shown in other places.

The key distinction is between “this node belongs inside this system” and “I want to use or view this node here too.” In Meridian, a parent defines inherited rules and behavior, so belonging to two parents would have consequences beyond appearing in two places.

1. Research: the same image in two canvases

Suppose an image of Zeus belongs in a mythology canvas, but you also want it in a canvas about Greek sculpture.

With one parent, the sculpture canvas could contain a reference node that displays the original image. Editing the original would update both appearances, while each appearance could have its own position, size, and surrounding connections.

With two parents, the image would inherit both containers’ rules. That could work, but changing the sculpture container’s rules might suddenly affect a node used in the mythology canvas.

For this case, references seem more predictable.

2. Spreadsheet-like containers

Imagine one node occupying a cell in two different containers. Its value might be shared, but its coordinates and formula context could differ: what does “the cell to my left” mean?

I’d distinguish two useful actions:
• Show the same node elsewhere: a reference to its existing identity
• Create another independent cell: a new node with its own GUID, optionally linked to the original value

True multiple-parent containment would need explicit rules for which container supplies each piece of context.

3. Working systems: one component versus two

If a resistor appears in two circuit diagrams, do you mean one physical component shown twice, or two separate resistors?

A reference would mean the same component and state, shown twice. It should not run twice simply because it has two appearances. A duplicate would get a new GUID and operate independently.

Connections into another operating system would still need explicit compatibility rules. References shouldn’t bypass either system’s restrictions.

4. Collapsing, moving, and deleting

I’d propose these predictable behaviors:
• Collapsing one appearance hides its descendants there without collapsing every other view
• Moving a reference changes its placement; moving the actual node to another parent changes its inherited context and requires validation
• Deleting a reference leaves the original intact
• Deleting an original warns about references that would become broken

With multiple parents, we’d need to distinguish removing one membership from deleting the node itself, and decide which changes affect every location.

Multiple parents aren’t inherently wrong. They’re useful when an object genuinely belongs to several structures, but

your non-overridable inheritance makes conflicting parent rules particularly important. A node would have to satisfy all of them, and some combinations would need to be rejected.

So I'd offer three clear operations: Move, Reference, and Duplicate. A reference could itself be an ordinary node with its own GUID pointing to the original GUID, preserving your "everything is a node" model. That gives you reuse across canvases while keeping containment and inherited behavior understandable.

These are proposed behaviors, not decisions I've locked into the specification.

47 User

3 October 2026 | 06:02:13 UTC

Ok I see what you mean. With a variety of systems being modeled it becomes difficult to account for every situation. For example a transistor: say the user places a transistor in a live system and the value of the transistor changes according to some input. Now the user references that transistor in a separate system. Its value should be preserved across systems. However, the user should not be able to simultaneously define input to the referenced copy that competes with the original system. The reference should act more like an impression or image of the original. A digital copy. Only the original context should directly effect its value. We can accomplish this by associating the GUID with the node object itself rather than its contents. Let the node preserve and deliver a model of its content - not the content itself.

48 Assistant

3 October 2026 | 06:02:24 UTC

That gives references a clear boundary: the original node owns the live state, and other contexts receive a read-only representation. They can't feed competing inputs back into it.

The GUID identifies the node itself; references use that identity to obtain its content model. We'll also need the explicit no-write-back rule, since identity alone doesn't prevent changes.

Should a reference continuously reflect changes in the original, or capture a snapshot that updates only when requested?

49 User

3 October 2026 | 06:11:06 UTC

A reference should only reflect change continuously if it is embedded in the same canvas workspace. If a reference comes from outside the canvas, it should only contain a snapshot of the last time the canvas was saved/closed.

50 Assistant

3 October 2026 | 06:11:14 UTC

Got it: references within the same canvas update continuously; references from another canvas use its last saved/closed snapshot. Both remain read-only.

When that source canvas is saved again, should other canvases automatically load the newer snapshot, or keep their existing snapshot until the user refreshes it?

51 User

3 October 2026 | 06:13:29 UTC

It doesnt make much difference to me. Lets say it automatically updates when saved.

52 Assistant

3 October 2026 | 06:13:36 UTC

We'll use automatic refresh when the source canvas is saved; unsaved changes stay local.

Next: when several connection lines merge into one, is that junction simply a visual arrangement, or can it perform an operation, such as combining signals, depending on the parent system's rules?

53 User

3 October 2026 | 06:14:11 UTC

Yes - it should be capable of combining signals or values.

54 Assistant

3 October 2026 | 06:14:20 UTC

Then a junction can combine signals or values according to the system's rules.

Since everything in Meridian is a node, should that junction itself be a node with a GUID and configurable behavior, even if it appears only as a small connection point?

55 User

3 October 2026 | 06:22:07 UTC

There should be a special link type that enables this type of behavior. Let there be a generic link that is insensitive its endpoints and is simply visual for things like infographs. Then let there be a specialized link that responds to the value of its input nodes. There should also be designated input > output nodes for things like calculations.

56 Assistant

3 October 2026 | 06:22:20 UTC

That gives us three distinct elements:
• Generic links: visual connections that don't react to endpoint values
• Specialized links: respond to values from their input nodes
• Input → output nodes: perform calculations or other defined operations

Should specialized links only carry values between nodes, with calculations handled by nodes, or can the links themselves also transform or combine values?

57 User

3 October 2026 | 06:24:48 UTC

Lets differentiate between links and ioints. A joint is the meeting of two or more links. Joints define operations. Nodes define input values and output destination.

58 Assistant

3 October 2026 | 06:25:00 UTC

Got it: nodes define input values and output destinations; links connect them; joints are where two or more links meet and define the operations. I'll revise the earlier calculation-node wording accordingly.

Should each joint's operation be chosen explicitly by the user, or supplied automatically by the parent system's rules?

59 User

3 October 2026 | 06:25:47 UTC

User selected / configurable using the joints properties.

60 Assistant

3 October 2026 | 06:25:57 UTC

Each joint’s operation will be user-selected and configurable through its properties, within inherited rules.

When two links simply cross visually, should they remain separate unless the user explicitly joins them?

61 User

3 October 2026 | 06:26:14 UTC

Yes

62 Assistant

3 October 2026 | 06:26:21 UTC

Crossing links stay separate unless explicitly joined.

For a working system, should calculations run continuously as inputs change, or should the user have Run/Pause controls and the option to advance one step at a time?

63 User

3 October 2026 | 06:31:56 UTC

Let operations occur like circuits - as soon as the circuit is complete, the operation occurs a single time. Lets also define switch objects to toggle open/closed circuits. Also, include variables that can be given guid's. Let GUID's act as the input to mathematical operations.

64 Assistant

3 October 2026 | 06:32:04 UTC

A completed circuit will trigger its operation once, and switch objects will open or close the circuit. Variables can have GUIDs, which operations use to look up their input values.

If a variable changes while the circuit stays closed, should the operation run again, or wait until the circuit is opened and closed again?

65 User

3 October 2026 | 06:36:08 UTC

Changing the value of an operand in a connected circuit should immediately effect the solution.

66 Assistant

3 October 2026 | 06:36:21 UTC

Then a completed circuit evaluates once initially and recalculates whenever an operand changes, without needing the switch toggled.

When a switch opens the circuit, should the output keep its last calculated value, or become unavailable until the circuit closes again?

67 User

3 October 2026 | 06:37:38 UTC

Opening the circuit should clear the solution or equal undefined.

68 Assistant

3 October 2026 | 06:37:50 UTC

Opening the circuit clears its solution to undefined.

If that output feeds another calculation, should undefined propagate downstream and clear those results too?

69 User

3 October 2026 | 06:39:06 UTC

Yes

70 Assistant

3 October 2026 | 06:39:14 UTC

Undefined will propagate through dependent calculations, clearing their results too.

We've settled a substantial part of the model. I'll consolidate these decisions into the revised product specification now and leave the remaining questions clearly marked.

71 Assistant

3 October 2026 | 06:50:56 UTC

The revised specification now captures the all-node model, inherited rules, references, links and joints, and circuit behavior. I'm saving the updated Word and PDF files to your Library.

72 Assistant

3 October 2026 | 06:52:00 UTC

Here's the revised Meridian specification, incorporating our decisions through undefined-value propagation. It replaces the earlier subject/display-entry model and removes build instructions and development-cycle details.

The remaining design questions are marked explicitly, including feedback loops and containment-parent rules. We can pick those up whenever you're ready.

73 User

3 October 2026 | 13:46:52 UTC

I want you to research a coding language I used to use several years ago. The entire language was created specifically for creating audio output. It was a visual language - it used a canvas workspace (similar to Meridian). Users were able to build wave forms from scratch, or embed audio files. Within the canvas space, there were a variety of tools available that the user could drop into the canvas space and connect into the system they built. They may for example, drop a mixing board into the canvas, a microphone, l/r stereo outputs, choose a file to sample and feed it into the system, then add a gain module into the system. The user would manually connect the audio cables between components. The whole language was designed to be intuitive and easy to learn and would allow the user to quickly set up any audio arrangement or recreate a professional workspace from scratch within a relatively short time frame. Can you figure out what language/platform this was?

74 Assistant

3 October 2026 | 13:47:03 UTC

That sounds a lot like Max/MSP or Pure Data. I'll compare their interfaces and audio tools against what you remember

75 User

3 October 2026 | 13:47:34 UTC

It was Max/MSP.

76 Assistant

3 October 2026 | 13:47:46 UTC

That gives us a useful reference for Meridian: the hands-on experience of placing components and wiring them together. Was there anything about Max/MSP's workflow you especially want Meridian to preserve or improve?

77 User

3 October 2026 | 14:07:45 UTC

Yeah I loved the idea of creating raw wave forms and being able to stack them up, display them, add layers and effects, and control input and output so directly. I'd definitely be interested in building this sort of functionality into Meridian. Especially the ability to generate pure tones, stack them up, and get a visual display of their output. I want you to include as much of Max/MSP into Meridian as possible. I want all basic types of logic gates to be included, I want to utilize the symbols used in wiring diagrams, I want a variety of output types. I honestly can't recall the variety of components used in Max/msp but I can safely say that we should aim to recreate as much as possible as long as it doesn't become burdensome or resource intensive in some way.

78 Assistant

3 October 2026 | 14:07:51 UTC

I'll expand the specification with Max/MSP-inspired signal tools, waveform generation and displays, logic gates, wiring symbols, and output options. I'll also check which features need limits or optional activation to keep large Meridian canvases responsive

79 Assistant

3 October 2026 | 14:08:47 UTC

One important decision: generating a tone needs a signal that keeps running over time. I'd give a parent container a continuous-signal mode, so its oscillators and effects run while its circuit is closed; ordinary calculations elsewhere would keep the behavior we already defined.

Should closing that kind of circuit start continuous output, and opening it stop the signal and clear its dependent results?

80 Assistant

3 October 2026 | 14:14:59 UTC

The [Max documentation](https://docs.cycling74.com/learn/articles/03_msphowmspworks/) gives us a good starting palette:

- Oscillators, noise sources, samples, looping and recording
- Mixers, gain, panning, filters, delays and modulation
- Envelopes, clocks, triggers and sequencers
- Waveform and spectrum displays, level meters and clipping indicators

- Audio, MIDI, numeric, text and Boolean outputs
- AND, OR, NOT, NAND, NOR, XOR and XNOR gates

I’m including that breadth in the draft, with heavier capabilities such as video processing and large banks of simultaneous voices marked as optional. I’m also proposing master mute, output-level controls and visible resource usage

81 User

3 October 2026 | 14:21:11 UTC

Excellent - yes a closed circuit should create continuous output not a single arbitrary length output.

82 Assistant

3 October 2026 | 14:21:17 UTC

Closed signal circuits will keep producing output until opened or stopped, with changes to their inputs taking effect while they run. I’ll make that explicit in the specification

83 Assistant

3 October 2026 | 14:28:45 UTC

The expanded specification is ready, including your continuous-output decision. I’m saving the updated Word and PDF copies to your Library

84 Assistant

3 October 2026 | 14:29:32 UTC

Here’s Meridian v0.3, with waveform generation, layering and effects, visual displays, logic gates, wiring symbols, and continuous signal output. The broader component catalog and resource safeguards are marked as proposals where we still have choices to make.

85 User

3 October 2026 | 17:41:13 UTC

Another key aspect of Meridian: There will be two modes. Viewing mode and editing mode. I want canvases to be created and altered in editing mode which is only accessible using the stand alone app. I want that stand alone app to be embedded in MyNet on Hypershell (do you know what I'm referring to?). Finished canvases should be viewable and interactive as stand alone files. I want to be able to embed a finished canvas in a webpage. I want users to be able to download a Meridian canvas from a website, then open it in editor mode and make changes. Actually thinking back, my original concept included an online repository for community canvas files and the ability to work together in a shared canvas workspace to collaborate. Lets try to preserve that - so just like Hyperlib pages are created locally then published to atlas archives, Meridian files will be created in a user's editor (ran in a browser) then selectively published in their personal repository online once their ready. Meridian's editor and the repository system will live at atlasuniversalis.com, once were ready to finalize.

86 Assistant

3 October 2026 | 17:41:22 UTC

I know Hypershell and Hyperlib, but I don’t yet have enough context to confidently identify MyNet. I’ll check what we’ve recovered and add this publishing and collaboration model to the specification.

One detail to clarify: does your later browser-editor description replace the standalone-only editing restriction, or should the browser editor be accessible only through Hypershell/MyNet?

87 User

3 October 2026 | 17:54:59 UTC

Just make a note of the hypershell/hyperlib details for now and a future agent will understand. Well help me flush this out. I don't necessarily want users to have to go to atlasuniversalis.com every time they want to create/edit pages. However, I do want the editor to be hosted there. I like the idea of having the editor itself be downloadable as a web app that can be run locally and to allow users to store their Meridian projects locally instead of all files being hosted online. I think this is where I want to pass this project over to another agent and pick up with them. Thank you for your help - please create a Meridian project .pdf that's updated with everything we've discussed about the project up to this point. Please include a detailed overview of the concept in its current form and include a complete transcript of this conversation. No need to include unrelated portions when talking about emails or other projects - only include messages discussing Meridian.

88 Assistant

3 October 2026 | 17:55:05 UTC

I'll put together a handoff PDF with the full current concept and a verbatim Meridian-only transcript. I'll include the downloadable, locally run web editor and local project storage, and leave the Hypershell/Hyperlib integration notes for the next agent

89 Assistant

3 October 2026 | 18:06:41 UTC

The handoff is ready: 36 pages covering the current concept, remaining decisions, and 87 Meridian-related messages. I've checked the transcript against the conversation and am saving the finished PDF to your Library

90 Assistant

3 October 2026 | 18:07:58 UTC

Here's your Meridian handoff PDF, ready to give the next agent. It includes the detailed current concept, local web-editor and publishing model, open decisions, and the Meridian-only transcript starting on page 21.

Earlier ideas remain in the transcript for context; the overview reflects your latest decisions. Unrelated discussions are excluded.

91 User

3 October 2026 | 20:59:42 UTC

Something worth adding the PDF: So the canvas itself is the root node. To flush this out a little - I want to include a canvas space layer which will act as the canvas repository where the user will browse their saved canvases and choose which to open. This will take place inside a canvas comprised of canvas nodes. Canvases can contain sub - canvases and canvases can be linked. Opening a canvas will then launch the primary canvas workspace where users will spend most of their time. At the canvas layer, links are only visual and cannot be of the complex types (like those performing operations or carrying electric current or logic gates) they simply indicate a relationship between the canvases. At the canvas layer, canvas nodes may still collapse and expand, revealing child canvas nodes but beyond this canvas nodes contain no distinct functionality. They should be regarded as a special type of node. Canvas nodes posses a title and an optional descriptive image. They can collapse/expand or launch a new window containing the canvases contents. Canvas nodes can also be placed inside a normal canvas workspace alongside any other type of node. For example, a node containing text 'Aphrodite' may expand to reveal several child nodes: a picture, several documents, a video, and a canvas node that launches a workspace that plots a story about Aphrodite using a collection of relational nodes.

92 Assistant

3 October 2026 | 20:59:46 UTC

I'll add the canvas repository as a navigable canvas of special canvas nodes, with visual-only relationships. I'll also include your Aphrodite example to show how those same nodes can sit inside a regular workspace and open another canvas in a new window

93 Assistant

3 October 2026 | 21:07:14 UTC

The canvas repository layer and Aphrodite example are now included, along with this latest exchange in the transcript. I'm saving the revised PDF to your Library

94 Assistant

3 October 2026 | 21:08:16 UTC

Updated PDF attached. The canvas repository, special canvas nodes, visual-only repository links, and Aphrodite example are on pages 3–4. The transcript includes your latest addition.

95 User

3 October 2026 | 21:32:28 UTC

Get rid of the stuff about plumbing. That was more of an example to describe what a user could do. I dont want to build in template objects for plumbing. Thats unnecessary.

96 Assistant

3 October 2026 | 21:32:34 UTC

I'll remove plumbing from the planned features and examples in the specification. Your original wording will remain only in the historical transcript, followed by this correction, so it's clear there's no plumbing template library planned

97 Assistant

3 October 2026 | 21:38:47 UTC

Plumbing has been removed from the current specification. I'm saving the corrected PDF to your Library

98 Assistant

3 October 2026 | 21:39:30 UTC

Here’s the corrected PDF, with plumbing removed from the planned scope and your clarification added to the transcript

99 User

4 October 2026 | 01:50:16 UTC

So before I wrap up this conversation with you - having helped me throughout the process of planning and laying out a conceptual framework: Is there anything you would consider adding or changing about the project? What are some templates, tools, or capabilities you'd build on to Meridian that I might be overlooking?

100 Assistant

4 October 2026 | 01:50:23 UTC

Yes. I’ll give you a prioritized critique of what would make Meridian more useful and easier to work with, including a few reusable templates that fit its core design.

101 Assistant

4 October 2026 | 01:52:07 UTC

Yes. Meridian already has plenty of expressive power. My biggest addition would be tools that help people understand, reuse, and troubleshoot what they create as their canvases grow.

These are recommendations for you to consider; I haven’t added them to the PDF.

1. Reusable assemblies with clear inputs and outputs
Let users turn a collection of nodes, links, and joints into a reusable module. Give it named inputs, outputs, and adjustable controls without requiring users to expand every internal component.

For example, someone could assemble a waveform generator, modulation controls, and a visual display, then reuse that whole arrangement elsewhere. The operations would still belong to joints inside the assembly. Visual links could retain your free edge-snapping, while functional connections would identify exactly which input or output they connect to.

2. A “why am I seeing this?” inspector
Click a result and trace the inputs, joints, inherited rules, and references responsible for it. If something becomes undefined, highlight the open switch or missing input that caused it.

I’d also include pause, single-step, and value history. A working system becomes much easier to understand when you can watch how its result develops.

3. Search, backlinks, and a navigable outline
The repository canvas helps users find canvases. They’ll also need to find things buried inside them.

Search should reach text, aliases, media metadata, and collapsed descendants. “Where is this referenced?” and “Open original context” would make GUID-based identity useful in everyday work. A synchronized tree outline and keyboard navigation would also help users navigate dense canvases without relying entirely on spatial memory.

4. Very clear copy and reference behavior
Users should be able to distinguish:
- Another view of the same original
- A saved cross-canvas snapshot

- An independent copy they can change freely

Show readable names and source information alongside GUIDs. For automatic snapshot refreshes, show which saved revision is being displayed and provide a way to inspect changes. Copying an entire assembly should also make its remaining external references obvious.

5. Self-contained canvas packages
A downloaded canvas should tell you whether everything it needs came with it: images, documents, samples, reusable modules, and saved reference data.

I'd add a "package for sharing" check that identifies missing files, external dependencies, and private material before publication. This would be particularly valuable for your locally stored projects and webpage embeds.

6. A deliberate run state and dependable undo
Opening a community canvas shouldn't unexpectedly start sound or expensive processing. Give viewers an explicit Run control and an always-accessible Stop All control.

For editing and collaboration, I'd prioritize undo, named checkpoints, and recovery of a previously working arrangement. Changing a parent's inherited rules should preview which descendants will be affected.

For starter templates, I'd choose a few broadly reusable arrangements:

- Research and comparison board: sources, claims, alternatives, evidence, and conclusions
- Interactive explainer: an overview with expandable details, media, and a small working demonstration
- Decision checker: visible conditions flowing through logic joints, with explanations for each outcome
- Signal playground: generators, mixing, effects, waveform/spectrum displays, and controlled output
- Assembly test bench: sample inputs, expected results, and inspection tools for testing a reusable module

The main change I'd make to the direction is to prioritize reusable assemblies over a growing catalog of specialized templates. Let users create and share the specialized systems. Meridian's strongest foundation would be excellent composition, navigation, inspection, and portability.

102 User

4 October 2026 | 02:22:38 UTC

I really like a lot of what you've outlined here. Go ahead and work #6, 5, 3, and 1. I especially like #1. Independent of referencing a GUID, there should be a packing container that allows the user to save a collection of nodes. The wrapping container should have an ID that, when searched in a dedicated tool, would generate a new instance of the node module with a unique set of new GUID's but which is in every other way identical to the original. Regarding #5 - downloading and reinstantiating a canvas on a new device should not necessarily preserve external links and references. This is a slippery slope and may have unintended consequences. Downloading a canvas should only capture nodes and links local to that canvas and shouldn't attempt to include external GUID's. Later on in the lifestyle of Meridian we may eventually build in a new type of reference ID that is truly universal once we flush out the online/collaborate aspect of Meridian. Users may eventually reference nodes embedded in canvases hosted publicly online. These are good though. If you have more thoughts or ideas to potentially add in, pitch!

103 Assistant

4 October 2026 | 02:22:46 UTC

I'll work those four into the specification. The module ID will act as a recipe: each insertion creates a fresh collection with new node GUIDs and the same internal relationships. Downloaded canvases will contain only their local nodes and links; external references won't be carried over.

104 Assistant

4 October 2026 | 02:23:56 UTC

Your module idea gives Meridian a useful distinction: a reference observes an existing object; a module ID creates an independent assembly. Its internal connections would be reassigned to the new GUIDs so they stay inside the new instance.

Three more additions I'd pitch:

1. A module shelf with previews and versions
Search by module ID, but also browse by name, tags, thumbnail, and exposed inputs/outputs. Saving an improved module could create a new version without unexpectedly changing instances people already placed.

2. Exposed controls on the packing container
Let a creator choose which controls appear on the outside of a collapsed module. A waveform assembly might expose frequency, amplitude, and output, while a research template exposes a title and document slots. This would make a saved collection feel like a usable tool immediately.

3. A preflight preview for publishing or downloading
Show exactly what the exported canvas will contain and what will be removed. If a calculation depends on an external reference, flag it before export. My recommendation is to leave a clearly marked, disconnected input with no external GUID, rather than silently substituting a value. That would respect your local-only rule while making any lost dependency understandable.

One important boundary: I'd treat canvas nodes that open other workspaces as potential external dependencies too. A download shouldn't quietly pull in an entire network of linked canvases.

105 Assistant

4 October 2026 | 02:34:24 UTC

The approved additions are incorporated, including fresh-GUID module instances and local-only downloads. I'm saving the updated PDF; the newer suggestions remain marked as proposals.

106 Assistant

4 October 2026 | 02:35:28 UTC

Updated PDF attached. The approved module, search, packaging, and safety/recovery capabilities are on pages 17–19. Your local-only download rule and fresh-GUID module behavior are included, along with the latest conversation.

107 User

4 October 2026 | 03:02:46 UTC

Yeah I think you've hit on a couple important points regarding dependency. Maybe we could treat references to GUID's as dependencies. Maybe even build in dependency tracking - when I user attempts to create a module or download a canvas we could include a footnote "This package includes:" with a list of referenced GUID's and Canvases. That introduces another issue, however. If a referenced GUID node is itself dependent on its surroundings or links to be equivalently operable then we must capture that as well. It seems we may need to introduce dependency at the node level. We can allow users to mark a node as a dependent and to define its dependency tree. Regarding the display of certain objects when a node is in its collapsed state - this is another issue and the ability to define display nodes that show special output is something that should exist whether a node has been packages into a template or not. I think the solution already exists. Consider a scenario - a node has a collection of child nodes that together generate a waveform. The user wants to hide all the children and so collapses the parent node. The user also wants to display a set of properties of the wave independently (even when the parent is collapsed). The user would create a set of output nodes that are siblings of the parent node (despite being linked to its children and

displaying their output) so that the parent can be collapsed without also collapsing the sibling display nodes. If this doesnt quite address the issue youre concerned with let me know. I do want to avoid building in too many specialized objects that are not nodes and dont follow the same ruleset as everything else on the canvas. In fact, I think that even the tool we use to define a reusable module in a system of nodes should itself be a node capable of linking and itself forming a system of modules to be stored and loaded together. And yes I agree that the user's library of stored modules should be loaded as a browsable list with stored modules also carrying names and listing their contents in addition to possessing a GUID.

108 Assistant

4 October 2026 | 03:02:55 UTC

Yes, sibling output nodes address the collapsed-display issue neatly: the parent hides its children while those sibling displays stay visible. There’s no need for a special display object. I’ll update the specification with that, node-level dependency declarations, and module-definition nodes that can themselves form linked systems.

One boundary needs your decision: should packaging now let users explicitly include external dependencies, overriding the earlier local-only rule, or should it only list them as missing requirements without copying them?

109 User

4 October 2026 | 03:42:33 UTC

I think the cleanest solution is to leave it up to the user when downloading. Include an option "Download External References?". I think this is the only way to ensure every case is accounted for. In some cases it may be a requirement in order to preserve basic functionality. In other cases it may lead to unnecessary inclusion of large amounts of data irrelevant to the users needs. Lets make the solution 2-fold. Allow the user downloading the canvas to choose (as mentioned before) but also allow the user to explicitly define an external canvas as a requirement when creating a nodal link. This would mean a user downloading the canvas would by default download linked canvases. That gives me another idea 💡 Let's allow users to create download able snippets while viewing a canvas. Build in a tool that allows the user to select multiple nodes or define a region of the canvas to download. This would create a blank canvas and paste the selection in the new space. Like using a template module, the copy would possess a unique set of GUID's.

110 Assistant

4 October 2026 | 03:42:40 UTC

I’ll add “Download External References?” and make author-designated required canvases included by default. I’ll also add downloadable selections in viewing mode: select nodes or a region, then create a new canvas with fresh GUIDs and the copied internal links preserved.

111 Assistant

4 October 2026 | 03:54:28 UTC

The dependency choices, node-based modules, sibling displays, and downloadable snippets are now incorporated. I’m saving the updated PDF, with the earlier local-only restriction replaced by your new download options.

112 Assistant

4 October 2026 | 03:55:35 UTC

Updated PDF attached. Dependency tracking and node-based modules are on pages 18–19; external-reference download options and fresh-GUID snippets are on page 21. The transcript includes both latest additions.

113 User

4 October 2026 | 05:51:15 UTC

In the opening statement of the handoff sheet can you add a paragraph that plainly describes the project to completion without going into detail about features? The way its worded now... you try to include a lot of specific features that are requiremenets while at the same time try to describe the product purpose. I want to make the product itself clear before honing in on features. You can add sections for the details weve outlined for each component latwr on in the specification. Heres a sample of how I kind of want you to start: Atlas Meridian is a 2D canvas that allows the user to create vast networks of information systems with a variety of characteristics. Objects placed by the user on the canvas are called nodes. Each node corresponds to a piece of media. Nodes can be linked, nested, and arranged in a variety of ways that enable the user to express different systems of information. Nodes can be combined into functional live systems representing things like electrical circuits, waveforms, and mathematical expressions using specialized links.... this is kind of the way we should start off the description. Lay out the whole concept, then gradually go into detail about different features and components.

114 Assistant

4 October 2026 | 05:51:23 UTC

I'll rewrite the opening as a plain-language introduction to what Meridian is and what people can create with it, following your example. Then I'll move gradually into the component descriptions and detailed requirements.

115 User

4 October 2026 | 05:51:51 UTC

I guess it'll probably end up being more than a paragraph, but you get the idea.